MDE; MDA; Transformaciones y DSLs. Una breve introducción.

Orozco M Daniel F; Giraldo O William J†; Trefftz G Helmuth
Escuela de ingeniería, Departamento de Informática y Sistemas, Maestría en Ingeniería, Universidad EAFIT-Medellín, Colombia.
†Programa de Ingeniería de Sistemas y Computación, Facultad de Ingeniería, Grupo SINFOCI, Universidad del Quindío, Armenia, Colombia.

Marzo 2013

Resumen

Con el avance de la investigación académica y científica se han impulsado diferentes técnicas y estrategias en el desarrollo de software. En la década más reciente se ha incrementado el uso y la aplicación del desarrollo de software dirigido por modelos, y a partir de este enfoque se han derivado muchas otras corrientes/vertientes del uso y aplicación de los modelos. En este artículo se realiza una breve aproximación al enfoque del desarrollo dirigido por modelos, su cobertura, la clasificación de las transformaciones para convertir esos modelos en algo físico (y no etéreo) y la manera cómo esos modelos pueden aportar a la construcción de un lenguaje de propósito específico.

Palabras Clave: *MDE; MDA; DSL; transformaciones; modelo; metamodelo; CIM (Modelo Independiente de la Computación); PIM (Modelo Independiente de Plataforma); PSM (Modelo de Plataforma Específica); PIT (Transformaciones Independientes de Plataforma); PST (Transformaciones de Plataforma Específica); dominio.*

1. Introducción

La tendencia actual del desarrollo de software se ha visto impulsada por el uso de modelos y de las abstracciones de los escenarios, en organizaciones y situaciones, para obtener de ellos sistemas y aplicativos informáticos a la medida. Es en este aspecto, en el del modelado, en el que debe concentrarse el esfuerzo inicial requerido para abordar una solución de caracter informático. De manera que, obteniendo una abstracción inicial y planteándola mediante un esquema de modelos se puede conseguir el máximo del entendimiento del problema (conocimiento del dominio); y así, de la mano de expertos del dominio, analistas de negocio y de sistemas, elicitadores de requisitos, arquitectos de software, diseñadores e implementadores, obtener una solución satisfactoria con un esfuerzo reducido, con participación independiente de cada uno de los roles y asegurando cumplir con la construcción de la solución planteada a través de la abstracción inicial. En este artículo se presenta una breve introducción al modelado bajo el contexto de la corriente *MDE* (secciones 2 y 3); y como los modelos por sí solos solo aportan el entendimiento entre las partes y el planteamiento de una solución, también se abordan, en este artículo, las coberturas de los modelos para apoyar el proceso de desarrollo de software (sección 4). En la sección 5 se presenta una introducción a las transformaciones (enfoques) de los modelos y la sección 6 aborda el tema y las características de un lenguaje de dominio específico para tratar la semántica y sintaxis de los modelos declarados en una solución. Finalmente, se presentan las conclusiones de este artículo (sección 7).

2. Definición de MDE

El modelo constituye la base fundamental de información sobre la que interactúan los expertos en el domi-

nio del problema y los desarrolladores de software. Por lo tanto es de fundamental importancia que exprese la esencia del problema en forma clara y precisa [16]. En [12] se afirma que si se trabaja con modelos, se pueden obtener importantes beneficios tanto en la productividad (prestando mayor atención, por parte de los desarrolladores, a la solución del problema a partir del dominio y no tanto en los detalles técnicos), como en la portabilidad (dependiendo de las herramientas de transformación usadas) independencia de la plataforma de sistema de software, en la interoperabilidad (haciendo referencia a los *'brigdes'* puentes – mapeo de elementos entre distintas plataformas y a la independencia de los fabricantes), en el mantenimiento y la documentación (esta se mantiene a un alto nivel de abstracción partiendo desde los modelos conceptual e independiente de plataforma).
El marco *MDE (Model–Driven Engineering)* ayuda a descubrir los elementos de un sistema a partir de la creación de modelos enfocados sobre los conceptos de dominio y no tanto sobre los conceptos de informática. Uno de los objetivos de *MDE* es especificar y explicitar los términos del negocio en modelos durante todo el proceso de desarrollo de software. A partir de este objetivo, este enfoque está centrado en el desarrollo de un *PIM (Modelo de Plataforma Independiente – PIM – por sus siglas en inglés)*, propiciando que los desarrolladores se enfoquen solo en modelos sin considerar aspectos de implementación. Se pueden trabajar independientemente los detalles de las plataformas específicas. Esto hace que el sistema desarrollado se acerque más a las necesidades del usuario final, el cual tendrá mayor funcionalidad en un menor tiempo. Realizar modificaciones sobre los modelos será mucho más rápido y seguro que hacerlo sobre el código final.
MDE, (Model Driven Engineering) es una aproximación de desarrollo de software centrado en la generación de modelos para describir los elementos de un sistema [23]. MDE es la sigla que se usa en el ámbito académico investigativo y no ha sido registrada bajo la batuta del Object Group Management (OMG)[18]. Esta sigla, MDE, es usada actualmente por la comunidad investigadora internacional cuando se refieren a ideas relacionadas con la ingeniería de modelos sin centrarse exclusivamente en los estándares y planteamientos de Object Management Group.

3. Propuesta MDA

MDA Model-Driven Architecture es un concepto promovido por *OMG (Object Management Group)*[17] cuyo propósito es afrontar el desafío de la integración de aplicaciones y cambios tecnológicos. El objetivo de MDA es separar el diseño del sistema tanto de la arquitectura como de las tecnologías de construcción facilitando que, diseño y arquitectura, puedan modificarse independientemente. *MDA* es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

MDA se enfoca en tres principios:

1. *Representación directa*, centrándose en las ideas y conceptos del dominio del problema y disminuyendo la distancia entre la semántica del dominio y su representación aplicando principios de abstracción; separando aspectos relevantes del dominio del problema de las decisiones de tecnología;

2. *Automatización*, promoviendo el uso de funcionalidades como el intercambio de modelos, el manejo de metamodelos, la verificación de la consistencia y la transformación de modelos; y

3. *El uso de estándares abiertos*, cuyo propósito es lograr la interoperabilidad de las diversas herramientas y plataformas apoyando al desarrollo de herramientas robustas, como por ejemplo, UML expresado en XML que resulta útil como mecanismo de transformación de modelos como, por ejemplo, la tecnología XSLT (Extensible Style Language Transformation).

Por ser *MDD (Model– Driven Development)* una marca registrada de OMG, la comunidad científica utiliza el término MDE para referirse a ideas y conceptos relacionados con la ingeniería basada en modelos, sin centrarse, exclusivamente, en los estándares de OMG.

En definitiva, MDD significa lo mismo que MDE, sólo que MDD es una marca registrada, mientras que MDE no. Si habláramos de medicamentos, MDE sería el nombre genérico del medicamento, mientras que MDD sería una marca comercial, cuyo componente principal es el medicamento MDE [3].

4. Cobertura de Modelos con MDA

En [22] se plantean dificultades cotidianas a la hora de utilizar los conceptos de MDA, que pueden hacerse presentes en todas las fases del ciclo de desarrollo, citándolos:

- Los modelos se usan solo como documentación.
- Existen vacíos entre el modelo y su implementación como sistemas resultantes.
- No hay una mezcla adecuada de modelos. Modelos desconectados y vistas desconectadas de un sistema.
- No hay transformación de modelos.

Para mitigar estos problemas en [13] se presenta una cobertura de modelos de MDA que apoyan, desde diversas perspectivas, el proceso de desarrollo de software:

4.1. Separación de Aspectos

[1]. Es una aproximación basada en múltiples vistas. La capacidad de integrar técnicas de múltiples perspectivas como CIM; PIM; PSM. Por ejemplo, para el modelo independiente de la computación (CIM), es posible modelar la lógica del negocio y los requisitos, por los stakeholders (teniendo en cuenta que cada participante trata sus asuntos con su propia apreciacióninterpretación), cada uno desde sus propias perspectivas, proporcionando un modelo separado del sistema más preciso y claro. Estos puntos de vista son el origen de los asuntos que se modelarán en el Modelo Independiente de la Plataforma (PIM) y en el Modelo de Plataforma Específica (PSM). La tendencia de programación sobre esta separación de aspectos es el enfoque Programación Orientada a Aspectos (POA).

4.2. Ingeniería dirigida por modelos reflexivos

[4]. En la que se aplican las nociones de PIT (transformaciones independientes de plataforma) y PST (transformaciones de plataforma específica). Es importante mencionar que las transformaciones de modelos a partir de modelos de dominio en modelos de plataforma si son desarrollados en un lenguaje propietario o inestable, pueden perder la capacidad de reutilización de los modelos de dominio que es una de las principales ventajas extraídas de MDA. La transformación de modelos, con las notaciones PIT y PST, se centra en dos pasos específicos:

1. Expresar las transformaciones de modelos de una manera independiente de la plataforma;
2. Mapear esta expresión de la herramienta independiente en una herramienta real (lo que conlleva a la herramienta o tecnología específica de las expresiones de transformación de modelos)

Las transformaciones de PIT a PST, visualmente podrían presentarse según la figura 1

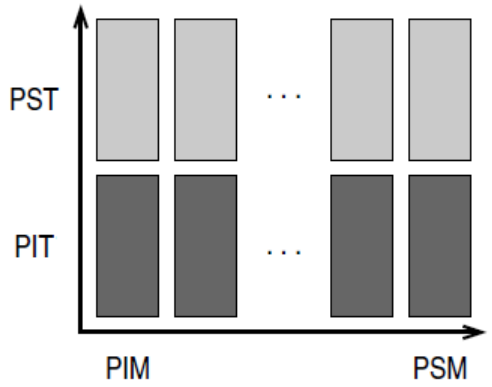

Figura 1: Tomado de [4]

La primera dimensión, el eje horizontal, es la usual disposición de las capas de los modelos de aplicación PIM a PSM (de modelo independiente de plataforma a modelo específico de plataforma) transformaciones de modelos aplicadas en los sucesivos PIM a modelos PSM; en la segunda dimensión, eje vertical, se presentan las expresiones de transformación de modelos a nivel PIT y PST. El proceso de transformación de modelos, bajo la ingeniería dirigida por modelos reflexivos puede representarse de la siguiente manera: (ver figura 2)

La ingeniería dirigida por modelos reflexivos se basa en el enfoque Orientado a Objetos y su metamodelo es centrado en UML para definir lenguajes de transformación de tipo PIT y PST.

4.3. Proceso explícito de Validación y Verificación (V&V) en MDA

[14]. Es la propuesta de un proceso genérico para ser usado en MDA para detectar errores e inconsistencias en etapas tempranas del desarrollo, evitando la propagación

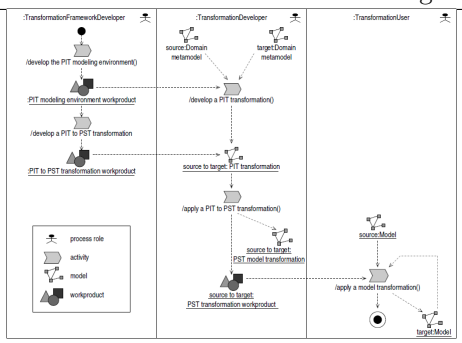

Figura 2: Ciclo de Vida del modelo de Transformación. Tomado de [4]

de estos errores hasta etapas posteriores. Un error en el PIM se arrastra hasta el PSM hasta llegar al código. Si en la etapa de modelado independiente de la plataforma se construyen modelos incompletos, imprecisos o inconsistentes se produce una propagación de estos errores al resto de las etapas.
La verificación intenta mostrar que un modelo satisface una propiedad específica. Esta verificación puede llevarse formal o informalmente. Las técnicas de la verificación formal son la demostración de teoremas y el model checking. Los algoritmos y el checklist son técnicas de la verificación informal. La validación evalúa si el comportamiento observable del modelo se ajusta a los requisitos. Si los modelos están basados en UML esta validación se realiza por medio de simulación de escenarios y casos de uso. Etapas del proceso de verificación y validación:

1. Identificar propiedades que deben cumplir los modelos construidos.

2. Propiedades relacionadas con la semántica estática de los modelos de acuerdo con las reglas de su metamodelo.

3. Propiedades relacionadas con la semántica dinámica de los modelos.

4. Propiedades relacionadas con la consistencia dentro de un modelo o entre modelos.

5. Propiedades relacionadas con la semántica concreta del dominio al que pertenecen los modelos.

6. Especificación/implementación del metamodelo. Resultado: una representación del mismo en un determinado lenguaje.

En la implementación del metamodelo se usan los modelos concretos que definen el sistema en las actividades de validación y verificación.

- La realización de esta transformación es un metamodelo distinto usado en el PIM origen. El PIMR (resultante) de esta transformación es un metamodelo utilizable para realizar las actividades de verificación y validación.

- Implementación de las propiedades.

4.4. Desarrollo incremental consistente integrando MDA y Orientación a Aspectos

[2]. Entre cada nivel de abstracción se hallan las transformaciones, otro de los mecanismos claves de MDA y MDD, cuyo propósito es establecer reglas de transformación entre elementos de un modelo de abstracción fuente hacia otro más refinado o más abstracto [11]. Todo este proceso puede disminuir su esfuerzo en ciertos requisitos no funcionales, trazabilidad, mantenibilidad, escalabilidad (evolución), pues los modelos que especifican el sistema se vuelven excesivamente grandes, complejos, provocando dificultad en el mantenimiento, en el reuso [2]. Además las transformaciones entre diferentes niveles de abstracción se vuelven muy complejas, excesivamente grandes y poco reutilizables [5].
Esta propuesta (Desarrollo Incremental consistente integrando MDA y Orientación a Aspectos) es que dichos modelos, desde el CIM hasta el PSM, puedan ser desarrollados por diferentes equipos de trabajo de forma descentralizada, concurrente y consistente con un mínimo de comunicación entre ellos.
El desarrollo de software orientado a Aspectos (DSOA), supone un avance hacia la modularización del software. Permite aislar propiedades del sistema cuya especificación queda dispersa por el sistema en artefactos (denominados aspectos). Aplicando técnicas de Separación de Aspectos a cada nivel de abstracción se pueden obtener modelos mejor modularizados, como consecuencia, modelos más manejables, con mayor reuso, con mejor mantenibilidad [2].

4.5. Definición y Descripción de PIM

[7]. Trata la separación de aspectos y requerimientos no funcionales; los aspectos de separación de plataforma independiente y plataforma específica. Indica las partes

en las que se define el PIM: *contexto, requerimientos, análisis, diseño* del comportamiento.

4.5.1. El contexto del PIM

es el modelo de presentación de la vista superior del sistema a desarrollar. Su propósito es definir claramente el alcance del sistema a desarrollar. El contexto PIM:

1. Introduce el sistema.
2. Presenta sus objetivos (se tiene el alcance),
3. se describen los principios de negocio que estructuran el sistema.
4. Describe a los actores externos que interactúan con el sistema.
5. Presenta los servicios de alto nivel requeridos para interactuar con el sistema como los comportamientos clave del sistema.
6. Y define los objetos de negocio.

4.5.2. Los Requerimientos PIM

especifican la vista externa del sistema usando una vista 'black box'. Esto es, cómo el sistema está obligado a actuar cuando es estimulado. Aborda:

1. la construcción de un modelo de expectativas de los clientes con un vocabulario claro,
2. disponer de una descripción única de requisitos para que todos los modelos posteriores puedan ser usados. Por ejemplo, un Modelo de Análisis modela la funcionalidad especificada en el modelo de requerimientos; un Modelo de Plataforma Específica puede usar un tipo de Sistema Operativo como se define en el modelo de requerimientos.

El modelo requerimientos está dado por los requisitos funcionales y los requisitos no funcionales. Los primeros: los servicios; los objetos de negocio; los eventos producidos y consumidos por el sistema y los actores interactuando con el sistema y las funciones que son descritas usando capacidades capabilities (casos de uso). Los requisitos No Funcionales como calidad del servicio (tiempo de respuesta, tolerancia a fallos); calidad del desarrollo (mantenibilidad, reusabilidad, flexibilidad y requerimientos específicos de plataforma).

4.5.3. El Análisis PIM

especifica la vista interna del sistema usando una vista 'white box', sin ninguna consideración tecnológica o de software y se sigue cumpliendo con el principio de separación de aspectos. Lo sostiene la especificación funcional del sistema, centrado en el dominio y áreas de aplicación. El Modelo de Análisis se describe y se ve como el modelo más duradero. Está libre de cualquier plataforma incluso de consideraciones de arquitectura 'física'.
El Modelo de Análisis incluye tres actividades:

1. Mantener las interfaces externas.
2. Realizar el análisis de Dominio.
3. Realizar el análisis de calidad del servicio.

4.5.4. Diseño de Componentes PIM

. Representa una plataforma independiente de la solución expresada en términos de componentes de software (componente, interfase, puerto, conector). Esta solución ('el cómo') cumple con los requisitos funcionales y no funcionales modelados en el Modelo de Análisis ('el qué') del sistema.
El Diseño de Componentes PIM incluye cuatro actividades:

1. Partición del sistema.
2. Realizar el diseño de componentes de frontera (boundaries).
3. Realizar el diseño de los componentes internos.
4. Realizar el despliegue de los componentes lógicos.

Esta cobertura de modelos aborda el aspecto de la transformación de modelos indicando un proceso de transformación asistido garantizando la consistencia con modelos de diferentes niveles de abstracción.

5. Transformaciones

Las transformaciones bajo este enfoque (OMG), involucran una especificación del metamodelo llamado *Meta Object Facility (MOF)*[21], un lenguaje de restricciones llamado *Object Constraint Language (OCL)*[19] y un lenguaje de transformación llamado *Query/View/Transformation (QVT)*[20].

La transformación de modelos, por lo general, adopta un enfoque orientado a objetos para la representación y manipulación del tema de sus modelos. La transformación define la generación automática de un modelo a partir de otro modelo. Esta definición es un conjunto de reglas que describen cómo uno o más artefactos en el modelo origen pueden ser transformados en una o más construcciones en el modelo destino [10].

Las transformaciones pueden ser:

1. *Endógenas:* Si están expresadas en el mismo lenguaje que los modelos.
2. *Exógenas:* No están expresadas en el mismo lenguaje que los modelos.
3. *Verticales:* Los lenguajes destino son diferentes entre sí.
4. *Horizontales:* Los lenguajes destino no son diferentes entre sí.

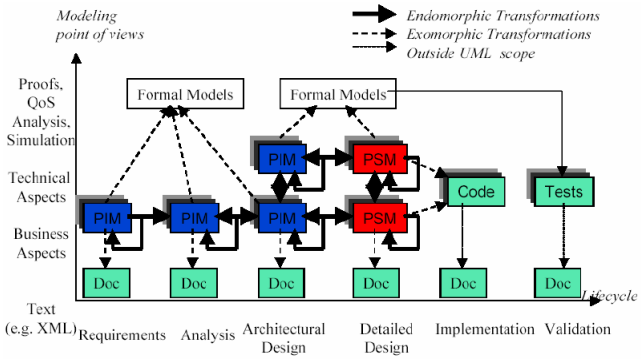

Figura 3: Alcance de las transformaciones. Tomado de [15]

En [6] se exponen aproximaciones de la transformación de modelos, particularmente la transformación de PMI a PSM.

5.1. Enfoque de Modelo a Código *(Model–To– Code)*

También referenciado como Modelo–a– Texto.

1. Enfoque Visitor–Based: Se trata de un mecanismo visitante que recorre la representación interna de un modelo para la generación de código muy básico y su escritura. Jamda, por ejemplo, (herramienta para representar modelos UML) tiene un mecanismo visitador para generar código.
2. Enfoque basado en plantillas (template–based): la mayor parte de las herramientas disponibles para MDA soportan la generación de código sobre este enfoque. Una plantilla, por lo general, consiste en el texto destino que contienen los 'empalmes'(mapeo) del metacódigo para acceder a la información del modelo fuente para realizar la selección de código y la expansión iterativa.

5.2. Enfoque Modelo–a–Modelo *(Model–to–Model)*

Estas transformaciones traducen entre el modelo fuente y los modelos de destino, que pueden ser instancias de los meta–modelos iguales o diferentes. Todos estos enfoques de apoyo introducen sintáctica de variables y de patrones. Por ejemplo, cuando se va de un diagrama de clases a una implementación en EJB, la herramienta (OptimalJ) genera un modelo de componentes intermedio EJB el cual contiene toda la información necesaria para generar todo el código de la clase Java.

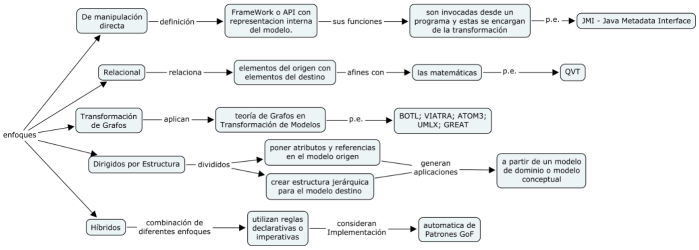

Figura 4: Clasificación de las transformaciones Modelo–a–Modelo. Presentación propia de los autores.

En la figura 4 se presentan las corrientes de este enfoque.
También se mencionan otros mecanismos de transformación, en [15], como CWM Common Warehouse Metamodel [18] cuya transformación se implementa en XSLT (eXtensible Stylesheet Language Transformation), que lo que hace es transformar modelos serializados en XML usando XMI. Lo que hace esta transformación es unir elementos del modelo origen con elementos del modelo destino. Este enfoque conduce rápido a implementaciones no mantenibles y a una dificultad en

su lectura por su nivel de detalle.

En [23] se presentan motores usados para la transformación y generadores de modelos.

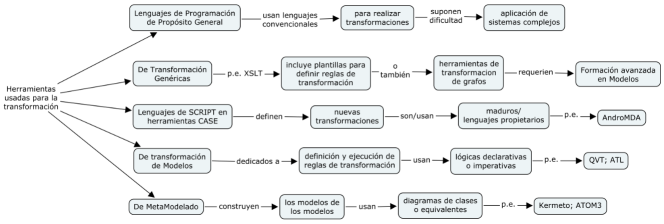

Figura 5: Herramientas para la transformación. Presentación propia de los autores.

Igualmente, en [23], se referencian los enfoques de la transformación de modelos y el número de pasos para ejecutar estas transformaciones.

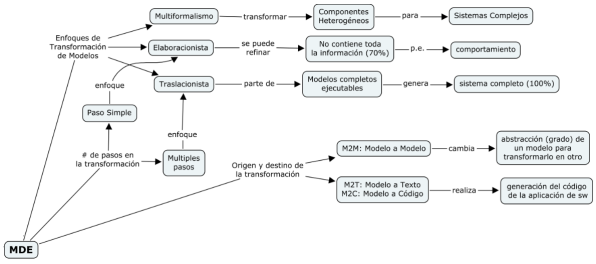

Figura 6: Enfoques de transformación de modelos y número de pasos en la transformación. Adaptado de [23]. Presentación propia de los autores.

Todas estas transformaciones deben, siempre, obtener un resultado. Bien sea un nuevo modelo, metamodelo, o código del aplicativo para el cual se ha modelado un dominio. Pero ese resultado, más allá de convertirse en un modelo de mayor (o menor) nivel de abstracción, debe aportar condiciones únicas para el dominio modelado. Este aporte de condiciones particulares sobre el dominio está dado por el *DSL (Domain Specific Language)*.

6. *DSL – Domain Specific Language*

En [9], DSL se define como artefacto que toma parte importante en el proceso de mapeo entre un problema de dominio y el modelo de implementación de ese dominio de solución. Un DSL contiene la sintaxis y la semántica de los conceptos de un modelo en el mismo nivel de abstracción que el dominio del problema ofrece.
Entiéndase *dominio del problema* como los procesos, entidades, restricciones que forman parte de la empresa (organización) que se está analizando. Es necesario comprender como las diferentes entidades del dominio interactúan entre sí y cumplen con sus responsabilidades. Es importante para comprender un dominio de problema compartir un vocabulario común, nos referimos a conceptualizaciones de términos unificados, para expertos del dominio y modeladores, y sobre todo, lo más importante, hacer fácil la comunicación (entendimiento común).
Un DSL debe dar la posibilidad de diseñar las abstracciones que forman parte del dominio. La abstracción es un proceso cognitivo del cerebro humano que permite centrarnos en los aspectos fundamentales de un tema, haciendo caso omiso de los detalles innecesarios.
El DSL difiere de un lenguaje de propósito general porque se encuentra dirigido a un problema (dominio particular) y porque contiene sintaxis y semántica que los conceptos del modelo en el mismo nivel de abstracción como el problema tiene. Otra particularidad, descrita en [9], es que el DSL tiene que ser comunicativo con su público y permitir que fragmentos de código sean lo suficientemente expresivos para mapear el proceso de pensamiento del modelador de dominio; como consecuencia de esto se obtiene en el diseño el nivel correcto de sintaxis así como de abstracciones semánticas para el usuario.
Cuando se programa usando DSL quien hace el programa puede centrarse en resolver el problema en cuestión y no tiene que preocuparse de los detalles en la implementación u otros elementos no esenciales del dominio de solución; siempre y cuando el DSL tenga el nivel apropiado de abstracción; en este caso se requiere que la mente humana (modelador) entienda su comportamiento.
El DSL, según describe [8], es un lenguage de un propósito determinado, cuya representación puede ser gráfica o textual, adaptado a problemas concretos de un dominio. Un DSL gestiona complejidad a través de abstracción organizada, en software existente y/o en literatura técnica, en la que participan expertos de un dominio. El DSL es un lenguaje de modelado especializado que se orienta hacia un dominio estrecho (específico). Entre estos dominios se pueden mencionar i) Verticales: dominio de negocio; telecomunicaciones; finanzas; banca; seguros; etc., y ii) Horizontales, subdividos en Transacciones y Técnicos: persistencia; interfaces de usuario; telecomunicacio-

nes; etc. También en [8] se aluden las cualidades deseables de un DSL:

1. Expresividad limitada.
2. Integridad conceptual.
3. Manejabilidad.
4. Promoción de la escritura de código conciso y legible.
5. Regularidad.
6. Fiablidad.
7. Consistencia con notaciones aceptadas convencionalmente.

Un DSL puede ser interno (embebido) o externo, en este caso, se representa en un lenguaje diferente al lenguaje de programación principal. Un DSL se traduce, generalmente, en llamadas a bibliotecas de subrutinas. El DSL provee sintaxis (que es gramática) que es la expresión de captura legal. También provee semántica que es lo que se hace cuando se ejecuta. Los diagramas para su representación, usualmente, son máquinas de estado. A todo este conjunto se le denomina Modelo Semántico [8]. Este modelo semántico se acompaña de un analizador (PARSER) que contiene operaciones jerárquicas, estructura de scripts (o también conocido como syntax tree). A toda esta estructura se le conoce como procesador DSL.

El modelo semántico es el mejor lugar para validar comportamiento. Posee dos interfaces:

1. Interface Operacional: que permite usar modelos de ('llenado') en el curso del trabajo, y,
2. llenado ('population') de interface: que se usa para crear instancias de clases en el modelo.

El modelo semántico es una representación del mismo tema que el DSL describe. Es una librería o un framework que el DSL rellena. El modelo semántico aumenta flexibilidad en el PARSER y en la ejecución.

En cuanto a la generación de código, a partir de un DSL, se presenta el concepto *Model–Award Generation* que posee dos estilos:

6.1. *Transformer Generation*

Que es código leído por el modelo semántico, y,

6.2. *Templated Generation*

Que es la especificación de una máquina de estados particular.
En [8] se define el término *Language Workbench* como las herramientas que ayudan a construir DSL's y herramientas de soporte para los mismos DSL's. Estas herramientas, languages Workbenchs, permiten el uso de metamodelos. Estas herramientas se caracterizan por tener los siguientes elementos:

1. Esquema de modelo semántico: se usa en el metamodelo. Por ejemplo, un modelo de datos sin mucho comportamiento.
2. Ambiente de edición de DSL: permite la escritura de DSL's.
3. Comportamiento del modelo semántico: define lo que el script DSL hace por fuera del modelo semántico.

Languages Workbenchs conocidos:

- Intentional Workbench.
- MetaEdit.
- Meta-Programming System (MPS).
- Xtext.
- MsSqlServer Modeling Project (Óslo')
- Clojure.

7. Conclusiones

El aporte que ha brindado el enfoque dirigido por modelos al desarrollo de software ha sido amplio y práctico, en el sentido en que el resultado de esta práctica minimiza el esfuerzo en el desarrollo de productos y garantiza la completitud en el entendimiento, tanto del problema como de la solución planteada. El esfuerzo del desarrollo puede minimizarse debido a que el equipo desarrollador se centra en las ideas y conceptos del dominio del problema y no en los aspectos de la implementación tecnológica, cumpliendo así con el principio de *Representación Directa* de la propuesta MDA; de esta manera se presenta un mayor acercamiento a las necesidades del usuario final.

Este enfoque de desarrollo de software dirigido por modelos invita a la separación de los mismos; es decir, el diseño del sistema puede separarse de las tecnologías de construcción y de la misma arquitectura, aportando como gran ventaja la modificación de cada uno de estos modelos de manera separada e independiente.

La cobertura de modelos, bajo la propuesta MDA, es un mecanismo de apoyo para el proceso de desarrollo de software. Estas diferentes perspectivas intentan llenar los vacíos que se presentan a la hora de manipular y trabajar con modelos. En teoría es fácil iniciar con un modelo, pero a la hora de volver realidad ese modelo pueden presentarse dificultades e inconvenientes que pueden hacer que el resultado esperado no sea lo propuesto en la abstracción inicial; de manera que, esta cobertura de modelos puede guíar, en la construcción del producto software, desde la abstracción inicial hasta la obtención del código final representado en una tecnología específica. Esta cobertura de modelos también invita a que los diferentes modelos puedan ser trabajados por diferentes equipos de desarrollo de forma descentralizada, concurrente y con un mínimo de comunicación, pero efectiva, entre ellos.

Las transformaciones son los resultados de construcciones de modelos iniciales en modelos destinos. Es decir, la generación automática de un modelo a partir de otro modelo. Y ese resultado debe aportar condiciones únicas (semántica y sintaxis) para el dominio que inicialmente se modeló.

Bibliografía

[1] Amaya Pablo; González Carlos; Murillo Juan M. Separación de aspectos en mda: Una aproximación basada en múltiples vistas. In *Actas del I Taller sobre Desarrollo Dirigido por Modelos MDA y Aplicaciones (DSDM'04)*, 2004.

[2] Amaya Pablo; González Carlos; Murillo Juan M. Aspectmda: Hacia un desarrollo incremental consistente integrando mda y orientado a aspectos. In *Actas del II Taller sobre Desarrollo Dirigido por Modelos MDA y Aplicaciones (DSDM'05)*, 2005.

[3] Anacleto Valerio Adrián. Arquitectura dirigida por modelos. *Revista Code*, (31):60–64, 2006.

[4] Bèzivin Jean; Farcet Nicolas; Jèzèque Jean-Marc; Langlois Benoît; Pollet Damien. Reflective model driven engineering. In *Proceedings of UML 2003*, volume 2863 of LNCS, pages 175–189. Springer, 2003.

[5] Clarke Siobhán; Harrison William; Ossher Harold; Tarr Peri. Subject–oriented design: Towards improved alignment of requirements, design and code. In *Object–Oriented Programming Systems, Languages and Applications (OOPSLA)*, 1999.

[6] Czarnecki Krzysztof; Helsen Simon. Classification of model transformation approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model– Driven Architecture*, 2003.

[7] Daniel Exertier; Benoît Langlois; Xavier Le Roux. Pim definition and description. In *Proceedings First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, pages 17–18, 2004.

[8] Fowler Martin. *Domain Specific Languages.* Addison-Wesley Professional, 2010.

[9] Ghosh Debasish. *DSLs in ACTION.* Manning Publications, 2011.

[10] González Rubén Antolín. Aplicación de ingeniería dirigida por modelos (mde) en procesos de negocio (bpm). Master's thesis, Departamento de Ingeniería Informática. Escuela Politécnica Superior. Universidad Autónoma de Madrid, 2008.

[11] IEEE Software. Special issue on model–driven development. Technical Report 5, IEEE, 2003.

[12] Kleppe Anneke. Warmer Jos. Bast Wim. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison Wesley, 2003.

[13] López L. Edna D; González G. Moisés; López S. Máximo; Iduñate R. Erick L. Proceso de desarrollo de software mediante herramientas mda. In *CISCI 6ª Conferencia Iberoamericana en sistemas cibernética e informática.* Departamento de Ciencias Computacionales. Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET). México, 2007.

[14] Lucas Martínez Francisco Javier; Molina Molina Fernando; Toval Álvarez Ambrosio. Una propuesta de proceso explícito de v&v en el marco de mda.

Grupo de Ingeniería del Software. Departamento de Informática y Sistemas. Universidad de Murcia, 157, 2003.

[15] Muller Pierre-Alain. Model transformations. an overview. http://www.irisa.fr/triskell/members/pierre-alain.muller/teaching/aboutmodeltransfo, 2005.

[16] Neil Carlos Gerardo. *Arquitectura de software dirigida por modelos - Diseño de un almacén de datos históricos en el marco del desarrollo de software dirigido por modelos.* PhD thesis, Universidad Nacional de la Plata. Argentina, 2010.

[17] OMG. *OMG: Model Driven Architecture (MDA).*, 2001. Document number ormsc/2001-07-01. (2001).

[18] OMG. *OMG. Catalog of OMG Modeling and Metadata Specifications.*, 2003. OMG. Catalog of OMG Modeling and Metadata Specifications.

[19] OMG. *OMG. Documents Associated With Object Constraint Language, Version 2.0.*, 2006. OMG. Documents Associated With Object Constraint Language, Version 2.0.

[20] OMG. *OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.1*, 2011. OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.1.

[21] OMG. *OMG: Meta Object Facility (MOF) Version 2.4.1*, 2011. Documents Associated With Meta Object Facility (MOF) Version 2.4.1 Release Date: August 2011.

[22] Quintero Juan Bernando & Anaya Raquel. Mda y el papel de los modelos en el proceso de desarrollo de software. *Revista EIA*, (8):131–146, 2007.

[23] Quintero Juan Bernardo; Duitama Muñoz Jhon Freddy. Reflexiones acerca de la adopción de enfoques centrados en modelos en el desarrollo de software. *Ingeniería y Universidad*, 15(1):219–243, 2011.