Universidad de las Ciencias Informáticas

Facultad 10

Título: Metamodelado para la construcción de software en entornos libres.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autoras: Lázara Lladianes Llanes Martell
Yaimet Isla Rodríguez

Tutor: MSc. Maidely Calderón Montero.
MSc. David Leyva Leyva.
MSc. Daymy Tamayo Ávila.

Ciudad de la Habana
Junio / 2008

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber".

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmamos la presente a los __ días del mes de __ del año _____.

Yaimet Isla Rodríguez

_____

Firma del Autor

Lázara Lladianes Llanes Martell

_____

Firma del Autor

MSc. David Leyva Leyva

_____

Firma del Tutor

MSc. Maidely Calderón Montero

_____

Firma del Tutor

MSc. Daymy Tamayo Ávila

_____

Firma del Tutor

Dedicatoria

Dedico este trabajo a mi familia que me ha apoyado en los momentos más difíciles por los que he pasado, por ser pacientes y consecuentes conmigo, por seguirme en mis ideas.

A mí querida abuela paterna por su incondicionalidad y su eterno amor.

A mi padre, por tu ejemplo, por su ayuda financiera, por querer que fuese siempre alguien en este mundo, por sus consejos y sobre todo por su plena confianza en mí.

A mi madre por estar siempre pendiente de mi presencia.

A mi hermana la que me guió y me ofreció las fuerzas necesarias para seguir adelante, por entenderme y la que me soporta todos mis egoísmos y malcriadeces.

A todos mis amigos por apoyarme en todo momento y confiar en mí, gracias por compartir lo mejor de sí mismos y por aceptarme en sus vidas, los quiero.

A mis queridas amigas de toda una vida universitaria, Liudmila, Dayli, Joannis, a mi compañera de tesis por soportarme tanto tiempo sin quejarse y quererme como yo la quiero; gracias.

A mi pareja por admitirme como soy, por ser consecuente, compañero ante todo, pacienzudo en los momentos más tensos de la carrera y por tenderme la mano cada vez que tropezaba para así no dejarme caer. A Esperanza por estar siempre pendiente de mí y ayudarme cuando lo necesité.

A todos que de una manera u otra nos ayudaron a todo lo largo de esta carrera, unos ya no están, otros sí, muchas gracias por estar siempre presente.

Yaimet Isla Rodríguez

Dedicatoria

A mi madre (Olguita) por estar siempre pendiente de mí, por ser la inspiración de mi vida, una gran amiga, la mejor madre del mundo, por haberme alentado en aquellos momentos en los que no tenía ni fuerzas para levantarme, por haberse mantenido a mi lado todo este tiempo luchando por mí y dando todo de sí.

A mi abuela (Olga) mi segunda madre, otra de las personas que son especiales en mi vida, y en mi corazón, por todos los regaños que me has dado, por estar siempre pendiente de mis estudios, por aconsejarme en todos los momentos que lo necesité, por haberme guiado por un buen camino; a ti, dedico en especial este trabajo.

A mi padre por haber confiado ciegamente en mí, por su apoyo incondicional, su cariño, su amor, por ser un ejemplo a seguir en todos los aspectos.

A mi prima Keilyn, por haberme ayudado cuando lo necesité, por sus consejos, por haber sido siempre para mí la hermana mayor.

A mi hermano Darián que lo adoro mucho, a mis dos pequeña (Yari y Wendy) que son una parte de mí y que son los angelitos que más quiero en esta vida.

A mi futuro esposo Jorge y su familia, que han sido para mí, amigos, compañeros y que me han apoyado mucho.

A mi compañera de tesis por haber sido todos estos años de carrera, amiga, compañera y hermana incondicional, y por último agradecerle también a todos mis amigos que compartieron los cinco años de carrera conmigo, a todos ellos decirle que nunca los olvidaré y que los quiero mucho.

Lázara Ll. Llanes Martell

Agradecimientos

A cada uno de nuestros profesores que a lo largo de nuestra carrera nos dieron lo mejor de sí, en conocimiento y talento personal, combinando brillantemente, maestría y sencillez.

A nuestros padres y hermanos que sacrificaron momentos de sus vidas para que pudiéramos seguir adelante en nuestros estudios y en la vida.

A nuestros amigos y hermanos de estudios que nos han ayudado para salir adelante y que siempre han confiado en nosotros para culminar esta tarea.

A todas aquellas personas que brindaron su colaboración y ayuda desinteresada.

Quisiéramos expresar nuestro cordial agradecimiento a nuestros tutores, por su dedicación y ayuda constante.

Agradecer también a nuestras parejas que nos han sido fieles en todos los momentos difíciles de nuestra carrera, brindándonos un hombro en el cual apoyarnos, siendo para nosotras amigos y compañeros ante todo.

Resumen

En el Desarrollo del Software Dirigido por Modelos (DSDM), los artefactos de software son representados mediante modelos abstractos. A través de los años, se ha utilizado el modelado por grandes empresas que desarrollan software, así como el uso de herramientas y técnicas de modelado avanzado.

En el presente trabajo se analiza el metamodelado, una de las técnicas del DSDM y la Arquitectura Dirigida por Modelos, una de las arquitecturas propuestas por el Grupo de Administración de Objetos u Object Management Group. Este nuevo paradigma se ha denominado “Ingeniería de Modelos” o “Desarrollo basado en Modelos”. La técnica que se analiza juega un papel importante dentro de este paradigma, y aporta un gran beneficio para los desarrolladores de software, esta posibilita el desarrollo de software con calidad.

En este trabajo se realiza un análisis de herramientas libres que permiten efectuar el metamodelado y lenguajes que son utilizados en el mismo. Se presentan además los estándares utilizados para el metamodelado. Se propone un lenguaje y herramientas de metamodelado para la producción industrial de software, tanto en la Universidad de las Ciencias Informáticas (UCI), como en el país.

Palabras Claves: MDA, Metamodelado, Lenguajes de Metamodelado, Estándares, Herramientas.

Índice de Figuras

Índice

Índice

Introducción

La Ingeniería de Software (IS), es una disciplina que ofrece métodos y técnicas para desarrollar y mantener software con calidad y resolver problemas de todo tipo. Hoy en día es cada vez más frecuente que se considere a la Ingeniería de Software como una nueva área de la ingeniería, y el ingeniero de software comienza a ser una profesión implantada en el mundo laboral internacional, esta disciplina trata áreas muy diversas de la informática y de la ciencia de la computación, abordando todo el ciclo de vida de desarrollo de cualquier tipo de sistemas de información.

La importancia fundamental de esta disciplina se está manifestando de modo destacado, en el currículum de informática y ciencias de la computación de la mayoría de las universidades de todo el mundo, y seguirá creciendo a medida que se acerque al tercer milenio. Debido a esto las organizaciones profesionales internacionales y departamentos educativos, se decidieron estandarizar los programas curriculares de las diferentes carreras, los planes de estudios de Facultades y Escuelas de Ingenieros de todo el mundo. Dos de las organizaciones que se ha preocupado por este proceso son la Association of Computing Machinery (ACM) e Institute for Electrical and Electronics Engineers (IEEE).

La recomendación dada por estas organizaciones es que divide el currículum en nueve áreas diferentes, donde una de ellas es la Ingeniería del Software, la cual incluye diferentes temas y dentro de ellos el Proceso de desarrollo del Software, como parte de este proceso se encuentra el Desarrollo de Software Dirigido por Modelos (DSDM), donde los artefactos software son representados mediante modelos a un nivel más abstracto que el propio código de las aplicaciones y los modelos juegan un papel fundamental en el proceso. Mediante el modelo se puede obtener una realidad física y abstracta o hipotética, utilizando un cierto lenguaje cuya semántica incluye la información necesaria que permite generar de forma automatizada el código de una aplicación a partir de un modelo de la misma.

El modelado en general permite enfatizar los elementos importantes y oculta los irrelevantes, es preciso, genera importantes ahorros en tiempo y recursos destinados al mantenimiento de software, permite el desarrollo de nuevas aplicaciones y nuevas características al software existente.

Introducción

Durante los últimos años, el interés de las principales compañías de desarrollo de software en el campo del DSDM ha crecido notablemente, el éxito de esta iniciativa ha dado origen a la evolución de distintos paradigmas como: Arquitectura Dirigida por Modelos (MDA), Factorías de Software (SF), Desarrollo de Software por modelos (DSM), Agil-Model-Driven Development (AMDD) y por último Domain Oriented Programming (DOP), cada una de ellas aborda el proceso de manera diferente. La iniciativa MDA establecida por Object Management Group (OMG), cubre un amplio espectro de áreas de investigación (metamodelos basados en MOF, perfiles UML, definición de lenguajes, construcción de modelos PIM - PSM y transformaciones entre ellos). MDA utiliza estándares y lenguajes como Facilidad de Meta-Objeto (MOF), XMI (XML1 Metadata Interchange), Lenguaje Unificado de Modelado (UML) y Object Constraint Language (OCL).

En la actualidad esta arquitectura ha tomado gran auge por las ventajas que ofrece, según (Ubeda, 2006), MDA define de manera clara la tecnología que recomienda utilizar para aplicar su enfoque. Esto facilita la tarea de construcción de un método a los desarrolladores que quieren aplicarlo. Se puede contar con lenguajes ampliamente conocidos que, en gran parte, no necesitan de descripción y disponen de abundante documentación. Está más tiempo en el mercado, ya que fue presentado con anterioridad a las Factorías de Software. Debido a esto, MDA ha sido más estudiado, discutido y aplicado. Este aspecto facilita la aceptación de un método por parte de la comunidad de la Ingeniería de Software. Ofrece un mayor soporte de herramientas que están empezando a madurar. El uso del estándar MOF permite definir en MDA lenguajes mediante el metamodelado. La experiencia de grandes empresas con el metamodelado como Model Integrated Computing (Modelo Integrado de Computación) ha demostrado un incremento de la productividad en el desarrollo de sistemas a partir de modelos y un consecuente descenso en los costes de desarrollo.

El metamodelado es uno de los principales temas de investigación en el campo de las bases de datos y desarrollo de software, el concepto nace con la misma filosofía que en su tiempo motivó a desarrollar computadores (hardware), de propósito general, en lugar de seguir con el esquema de construir un computador para cada aplicación en particular. Este concepto al aplicarse en el desarrollo de aplicaciones de software permite diseñar un modelo de datos (el metamodelo) con la capacidad de almacenar otros modelos de información.

[1] Extensible Markup Language o lenguaje de etiquetado extensible.

Introducción

El metamodelado es complejo en su fabricación, pero que una vez terminado, incrementa la velocidad con la cual los desarrolladores de software pueden atender cambios y modificaciones al software, unifica la interfaz para las personas que acceden a los sistemas de información y les entrega la posibilidad de generar consultas sobre la información registrada, acorde a sus necesidades y no sujetos a un conjunto limitado de posibilidades.

Por otra parte, debido a las ventajas que posee el software libre, la Universidad de las Ciencias Informáticas (UCI) está llevando a cabo un proceso de migración, por lo que son muchas las herramientas y lenguajes de modelado y metamodelado que aún se desconocen, que podrían servir de ayuda para obtener una mejora en el modelado de sistemas de software libre a cualquier nivel de abstracción. Con anterioridad se desarrolló una investigación (Hechavarría, 2007) donde se introdujeron temas de metamodelado. Esta investigación se enfocó sólo a la etapa de Requisitos. Por todo lo planteado anteriormente, se plantea el siguiente **problema**: ¿Cómo introducir el uso de técnicas avanzadas de Ingeniería de Software para mejorar la producción de software libre en la UCI?

De acuerdo a lo planteado en el problema anterior se tiene como **objetivo general** proponer una herramienta y un lenguaje de metamodelado para el proceso de desarrollo de software libre en la UCI.

Se plantea como **objeto de estudio** los lenguajes y herramientas de metamodelado para el proceso de desarrollo de software libre, y su **campo de acción** son los lenguajes y herramientas de metamodelado en Análisis y Diseño.

Se plantean las siguientes **preguntas científicas**:

1. ¿Cuáles son los lenguajes de metamodelado y cuáles son sus posibles aplicaciones al desarrollo de software libre?
2. ¿Cuáles herramientas de libre distribución permiten el metamodelado?
3. ¿Cómo pueden ser utilizadas las herramientas de metamodelado, así como los lenguajes y estándares asociados a ellas a la producción industrial de software libre?

Para dar respuesta a las preguntas anteriores se realizarán las siguientes **tareas** investigativas:

1. Estudio de los lenguajes y herramientas para el metamodelado.
2. Estudio de los estándares para el metamodelado.
3. Realizar el análisis de los lenguajes y herramientas para el metamodelado, así como su vinculación.
4. Elaborar una propuesta de un lenguaje con una herramienta de metamodelado que permita una mejora del desarrollo de software en entornos libres en la UCI.

Para la realización de este trabajo se utilizan diferentes métodos de investigación. El método histórico-lógico se utiliza para determinar la evolución de conceptos como modelado, metamodelado y MDA; el analítico-sintético y la revisión de documentos, para analizar la teoría relacionada con el metamodelado, y extraer información relevante.
Además se utiliza se utiliza la encuesta, para indagar sobre el grado de conocimiento del metamodelado en la UCI.

Este trabajo consta de una introducción, dos capítulos, conclusiones, recomendaciones y anexos. En el Capítulo 1 se abordarán los conceptos básicos para el desarrollo de este trabajo, se expondrá además un breve resumen de los lenguajes y las herramientas de metamodelado más extendidos privativas o libres, por último una breve caracterización del Análisis y Diseño de sistemas.
El Capítulo 2 englobará los lenguajes y herramientas de metamodelado, así como los estándares escogidos, se hará un análisis riguroso mediante una serie de parámetros, los cuales arrojarán un resultado para la propuesta de un lenguaje y una herramienta para la producción de software en entornos libres utilizando el metamodelado.

Capítulo1: Fundamentación Teórica

Introducción

En este capítulo se analizan conceptos básicos para la comprensión de este trabajo. Consta de un primer epígrafe donde se definen conceptos como: Ingeniería de Software, Proceso de Desarrollo de Software, MDA, modelo y metamodelado, en un segundo epígrafe se realiza un resumen de los lenguajes de metamodelado, en el tercero se mostrará una síntesis de los estándares que utiliza el metamodelado, seguidamente por otro epígrafe donde se realizará un compendio de las diferentes herramientas de metamodelado que se utilizan, además se hará mención de otras herramientas para el metamodelado, privativas y libres; y un último epígrafe donde se realizará una caracterización del Análisis y Diseño de sistemas.

1.1.1 La Ingeniería de Software

En la actualidad los conceptos de Ingeniería de Software son muy variados, algunos de los conceptos son los planteados por (Bohem, 1976), donde define que la Ingeniería de Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software.

Otro concepto es el que plantea (Cota, 1994), donde puntualiza que la IS es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software", es decir, "permite elaborar consistentemente productos correctos, utilizables y costo-efectivos".

De acuerdo a los conceptos antes expuestos se puede concluir que es un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad.

1.1.2 El Proceso de Desarrollo de software

El Proceso de Desarrollo de Software es considerado por (Wesley, 2000), como un proceso que define “quién” está haciendo “qué”, “cuándo” y “cómo” para alcanzar un determinado objetivo. Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los

esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto.

Las piedras angulares del proceso de desarrollo del software son: el proyecto, las personas y el producto; siendo las características del cliente, el entorno de desarrollo y las condiciones del negocio, elementos que influyen en el proceso. Por otra parte (Quispe-Otazu, 2007), define el Proceso de Desarrollo de Software como: Proceso de negocio, o caso de uso de negocio, de un negocio de desarrollo de software. Conjunto total de actividades necesarias para transformar los requisitos de un cliente en un conjunto consistente de artefactos que representan un producto software y es punto posterior en el tiempo para transformar cambios en dichos requisitos en nuevas versiones del producto. Para este trabajo se tomará el concepto de (Wesley, 2000), para definir el Proceso de Desarrollo de Software.

1.1.3 Modelos y Modelado

Existen autores con perspectivas diferentes acerca del concepto de modelo, como (Rumbaugh, Jacobson y Booch, 1998): que plantean que un modelo es una representación, en cierto medio, de algo en el mismo u otro medio. Según estos autores, el artefacto más utilizado en el Proceso de Desarrollo del Software es el modelo, pues cada trabajador necesita una perspectiva diferente del sistema, por tanto la construcción de un sistema es un proceso de construcción de modelos, utilizando distintos modelos para describir todas las perspectivas del sistema, a todo este proceso se le llama proceso de modelado.

El modelo capta los aspectos importantes de lo que se está modelando, desde cierto punto de vista, y simplifica u omite el resto. Un modelo se expresa en un medio adecuado para el trabajo. Un modelo de un sistema software está construido en un lenguaje de modelado, como UML. El modelo tiene semántica y notación y puede adoptar varios formatos que incluyen texto y gráficos. El modelo pretende ser más fácil de usar para ciertos propósitos que el sistema final. Además de estos autores otros como (Bellinger, 1997) considera que un modelo es una representación simplificada de un sistema desde un punto de vista particular en el tiempo y el espacio para proporcionar el entendimiento del sistema real.

En general el modelo debe expresar de manera clara y correcta lo que se quiere obtener, sin sobrecargar de información al modelo en sí, éste debe ser completo. Los modelos son utilizados en

la ingeniería para varios propósitos, dentro de ellos se puede mencionar que son utilizados para captar y enumerar exhaustivamente los requisitos y el dominio de conocimiento, de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos, ayudar a pensar el diseño de un sistema, para capturar decisiones del diseño en una forma mutable a partir de los requisitos, otros de sus usos es el generar productos aprovechables para el trabajo y “domesticar” los sistemas complejos.

La acción de modelar (modelado) es una técnica de sistemas software para tratar con la complejidad inherente a estos sistemas. El uso de modelos ayuda al ingeniero de software a "visualizar" el sistema a construir. Además, los modelos de un nivel de abstracción mayor pueden utilizarse para la comunicación con el cliente. Por último, las herramientas de modelado y las de Ingeniería de Software Automatizada, pueden ayudar a verificar la corrección del modelo.

En (Portillo, 2004) se aborda el modelado como un ejercicio de abstracción, entendida ésta como una simplificación y generalización de la realidad. Abstraer significa ignorar, excluir o esconder ciertos detalles de un conjunto de elementos con el objetivo de capturar y resaltar las características comunes a todos esos elementos. El uso de la abstracción permite describir, representar, manejar y resolver problemas complejos, pudiendo después aplicar los resultados a casos concretos. Normalmente, se construyen jerarquías de abstracción, en las que cada nivel de abstracción se apoya en los inferiores.

Niveles de abstracción en el modelado

El modelado presenta cuatro niveles de abstracción, en la que cada nivel se apoya de su nivel inferior, a continuación se explica en que consiste cada uno de estos niveles y como están relacionados según (Portillo, 2004).

1. **Nivel de Información**. Agregación ‘informal’ de datos a manejar en un entorno concreto (aplicación). Se separa un subconjunto de datos con características comunes o interesantes desde un determinado punto de vista.

2. **Nivel de Modelo**. Agregación ‘informal’ de metadatos (datos sobre los datos) que describen una información concreta. Se describen las características comunes de los datos dando

lugar a metadatos que se agrupan, describiéndose las relaciones entre ellos, para formar un modelo.

3. **Nivel de Metamodelo**. Agregación 'informal' de modelos o meta-metadatos (descripción de metadatos). Descripciones que definen la estructura y la semántica de los metadatos. Se describen las características comunes de subconjuntos de metadatos dando lugar a meta-metadatos o modelos que se agrupan, describiéndose las relaciones entre ellos, para formar un metamodelo.

4. **Nivel de Metametamodelo**. Definición de la estructura y la semántica de los meta-metadatos. Se capturan las características comunes de subconjuntos de modelos dando lugar a metamodelos que se agrupan, describiéndose las relaciones entre ellos, para formar un metametamodelo.

En la siguiente figura se puede visualizar mejor los niveles de abstracción.

NIVELES DE ABSTRACCIÓN	ENTIDADES MANEJADAS	LENGUAJES DEFINIDOS
Meta-metamodelo	Meta-modelos	Lenguaje para la descripción de diferentes tipos de modelos. Lenguaje de modelado
Meta-modelo	Meta-metadatos ó Modelos	Lenguaje para la descripción de diferentes tipos de datos (meta-datos)
Modelo	Meta-datos	Lenguaje para la descripción de datos concretos
Información	Datos	

Figura 1: Niveles de abstracción del modelado (Portillo, 2004).

1.1.4 MDA (Model-Driven Architecture). Arquitectura de software dirigida por modelos.

La comunidad de la Ingeniería de Software está prestando en la actualidad, un especial interés a uno de los enfoques propuesto por la OMG para el desarrollo de software: MDA. A continuación se detalla en que consiste esta arquitectura.

Según (Anaheim, 2003), la principal característica que diferencia a MDA respecto a los enfoques tradicionales para el desarrollo de software se encuentra en el uso de los modelos como el recurso principal en el proceso de desarrollo. MDA propone que los sistemas software sean generados directamente a partir de modelos de dicho sistema, es decir, es un enfoque para el desarrollo de sistemas que ofrece una forma, que los modelos dirijan el proceso de comprensión, diseño, construcción, instalación, operación, mantenimiento y modificación. OMG propone para este paradigma y promueve, el uso de diversos lenguajes y estándares relacionados con la creación y gestión de modelos (UML, MOF, XMI y OCL) como mecanismos básicos para soportar esta estrategia. En (Hechavarría, 2007) se expresa que en MDA existen dos ideas principales:

1. Separar la especificación de la funcionalidad del sistema de su implementación, sobre una plataforma en una tecnología específica.
2. Controlar la evolución desde modelos abstractos a implementaciones tendiendo a aumentar el grado de automatización.

1.1.4.1 Modelos y MDA

(Hechavarría, 2007) distingue que MDA presenta cuatro tipos de modelos los cuales se explican a continuación:

1. **Modelo Independiente de Computación (Computation Independent Model (CIM))**

   Es una descripción de la lógica del negocio desde una perspectiva independiente de la computación. Es un modelo del dominio.

2. **Modelos Independientes de la Plataforma de Implementación (Platform Independent Model (PIM))**

   Es una descripción de la funcionalidad del sistema en forma independiente de las características de plataformas de implementación. Estos son modelos de sistema que no

contiene información acerca de la plataforma o la tecnología que es usada para implementarlo.

3. **Modelos Específicos de la Plataforma de Implementación (Platform Specific Model (PSM))**

   Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma específica. Es una descripción del sistema en términos de una plataforma específica. Por ejemplo, .NET y J2EE. Modelos reusables a distintos niveles de abstracción.

4. **Modelo especifico de implementación (Implementation Specific Model (ISM))**

   Es una descripción (especificación) del sistema a nivel de código. Por ejemplo, Java, C# entre otros.

En MDA es crucial mantener trazas y relaciones entre diferentes modelos, así como lograr interoperabilidad en diferentes niveles. Los metamodelos son importantes en MDA porque actúa como mecanismo para definir lenguajes de modelado, de forma que su definición no sea ambigua. Que no sea ambigua permite que una herramienta pueda leer, escribir y entender modelos como los construidos con UML.

Por otro lado, las reglas de transformación usadas para transformar un modelo en lenguaje A en otro modelo en lenguaje B, usan los metamodelos de los lenguajes A y B para definir las transformaciones.

Los metamodelos en MDA, aportan grandes ventajas como por ejemplo, permiten el intercambio de modelos entre herramientas de modelado, permiten la representación específica de elementos de dominio, uso de una terminología común, reduce las ambigüedades, permite producir documentación completa, chequeo de la consistencia de los modelos, trazabilidad de los elementos del modelo.

1.1.4.2 Arquitectura MDA

MDA es una arquitectura de lenguajes de modelado que distingue 4 niveles de aplicación principales, así lo considera (Rosario, 2007).

1. Nivel de Metametamodelo (M3): Proporciona un lenguaje para especificar metamodelos (MOF).
2. Nivel de Metamodelo (M2): Proporciona una instancia de M3 que conforma un lenguaje de modelado (UML).
3. Nivel de Modelo (M1): Proporciona los modelos generados con el lenguaje de modelado (Modelos UML).
4. Nivel de Objetos (M0): Proporciona el modelo en tiempo de ejecución, la aplicación conforme al modelo especificado en M1 (Aplicación).

1.1.5 Metamodelado

El metamodelado se comprende a partir de una estructura de cuatro niveles (M0, M1, M2, M3). En esta estructura, el nivel M1 establece las primitivas de metamodelado, esto es, el lenguaje de metamodelado. En el nivel M2, se definen los metamodelos en sí con las primitivas de M1. En M3 se aplican los metamodelos para generar instancias, que se denominan modelos. Por último, la instanciación de los modelos, lleva al nivel donde se tiene la información que se manejará en el sistema.

Para entender lo que es un metamodelo, es necesario entender la diferencia que hay entre un metamodelo y un modelo, un metamodelo es un modelo de un lenguaje de modelado y un modelo es confeccionado por un lenguaje de modelado. Aunque un metamodelo también es un modelo, un metamodelo tiene dos características principales. Primeramente, debe capturar los rasgos esenciales y propiedades del lenguaje que está siendo modelado. Así, un metamodelo debe ser capaz de describir la sintaxis concreta de un lenguaje, sintaxis abstracta y semántica. Como segunda característica, un metamodelo debe ser parte de una arquitectura del metamodelo, una arquitectura de metamodelo permite ver un metamodelo como un modelo que a su vez se describe por otro metamodelo. Esto permite describir todos los metamodelos por un solo metamodelo, este solo metamodelo, a veces conocido como metametamodelo, es la clave para meta-modelar, ya que él permite describir todos los lenguajes de modelado, una forma de definir el metamodelado podría ser como lo plantea (De Lara, Vangheluwe y Alfonseca 2003) que definen el Metamodelado como

el “proceso de modelado de formalismos”, el cual se suele realizar para determinar si una instancia de un modelo particular es consistente con su especificación en forma de metamodelo (Sourrouille, J. & Caplat, G. 2003). La técnica de metamodelado permite definir lenguajes de modelado, a través de un metamodelo, en (Hechavarría, 2007) se expresa que un metamodelo es un modelo con la capacidad de almacenar diferentes modelos, sin embargo otros autores como (Favre 2004) y (Clark, Evans, Sammut, Willans, 2007), coinciden en que un metamodelo es un modelo de un lenguaje de modelado.

De acuerdo con las definiciones antes planteadas se puede resumir que el metamodelado no es más que una técnica que permite la creación de lenguajes de modelado, mediante la creación de metamodelos, ya que estos como bien se dice anteriormente, no es más que un modelo de un lenguaje de modelado.

1.1.5.1 Áreas del Metamodelo

El metamodelo se divide en dos áreas principales, así se plantea en (Hechavarría, 2007):

Área de metadatos

Esta área almacena el metamodelo en sí, en un conjunto de tablas relacionales se almacena una descripción detallada del modelo de los datos a ser almacenados en el área de datos. Este modelo parte del principio que toda información para ser registrada en un sistema de información, debe ser estructurada en uno o varios formatos que puede hacerse siguiendo una metodología denominada "Metodología para la estructuración de formatos en un metamodelo", la cual permite la creación de uno o varios formatos en un contexto determinado.

Área de datos

Esta área almacena la información acorde a la estructura definida en el área de metadatos. Es aquí donde se registra la información proveniente del mundo real, estructurada de acuerdo a los formatos que hacen parte del modelo almacenado en el área de metadatos

Los metamodelos son útiles dentro de la IS, después de realizar un estudio acerca del metamodelado, se puede plantear que los metamodelos se utilizan para gestionar sistemas complejos más fácilmente, definir lenguajes de modelado, permite interoperabilidad entre herramientas apoyándose en estándares como XMI, entre otros.

1.2 Lenguajes de Metamodelado

Un lenguaje de metamodelado es un lenguaje para describir otros lenguajes. Los lenguajes de metamodelado más extendidos son MOF de OMG, Ecore para Eclipse (asociado al EMF[2]) y el lenguaje de modelado UML 2.0. Una definición de un lenguaje a través de un lenguaje de metamodelado se denomina metamodelo. A continuación se expondrá una breve explicación de cada uno de los lenguajes.

1.2.1 MOF

Meta Object Facility o Facilidad de Meta-Objeto, es el lenguaje de metamodelado propuesto por el OMG. Es utilizado para crear metamodelos (por ejemplo, el metamodelo de UML ha sido definido con MOF), y es, por tanto, un elemento básico de MDA, es un lenguaje "Medio universal para definir lenguajes de modelado", permite expresar metadatos (igual que XML), es independiente de la plataforma, es descrito con la notación UML, OCL y texto informal, la notación para metamodelos MOF es la sintaxis concreta de UML, utiliza OCL para expresar la semántica estática y además usa paquetes si el metamodelo que se va a desarrollar es muy grande, obtenido de (García, 2008).

1.2.2 Ecore

Ecore es un lenguaje común basado en EMOF[3], que es parte de la especificación MOF 2.0, es usado para la definición de metamodelos. El lenguaje Ecore es usado por EMF (Budinsky, 2003) y (Moore, 2004). Los metamodelos y modelos usados por EMF se representan con documentos XML.

1.2.3 UML 2.0

El Lenguaje Unificado de Modelado 2.0 es el resultado de la evolución de UML que nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y el OMT (Object Modeling Technique, Técnica de Modelado de Objetos). Más tarde se les unió Ivar Jacobson, creador del método Object-Oriented Software Engineering, Ingeniería de Software Orientada a Objetos (OOSE). UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Está pensado

[2] Eclipse Modelling Framework: herramienta para el metamodelado.

[3] Essential MOF: Es una especificación de MOF, también un lenguaje de metamodelado.

para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios.

En UML la expresividad de dicho lenguaje no es suficiente para cubrir todo tipo de situaciones. Nacen así extensiones a UML para modelar otros elementos, tales como persistencia, seguridad e interfaces gráficas. Ante la proliferación de tantas extensiones, y la carencia de un estándar en que dichas extensiones sean especificadas, nace la necesidad de definir claramente un Metamodelo para UML. Un metamodelo es un modelo de un modelo; en el caso de UML, una definición de la estructura del lenguaje de modelado. Aunque desde un principio la especificación de UML contenía un metamodelo, éste era bastante ambiguo a la hora de extender sus capacidades.

Después de varios años de desarrollo apareció un nuevo estándar para UML, la versión 2.0, la cual incluye no sólo un nuevo metamodelo para el lenguaje de modelado en sí, sino que además incluye un metametamodelo llamado MOF. La idea detrás de esto es: proveer un lenguaje que permita a cada grupo de desarrolladores definir sus propios lenguajes de modelado. Si alguien necesita desarrollar un software que trabaje con páginas web, a través de este metametamodelo puede especificar qué cosas necesita en su lenguaje de modelado, por ejemplo: páginas HTML, formularios, enlaces entre páginas, etc., para luego modelar directamente el software utilizando estos elementos. Si otra persona necesita modelar un sistema orientado a objetos, a través del metametamodelo también puede especificar qué cosas necesita: clases, métodos, atributos, relaciones de herencia, etc., y luego crear un sistema basado en estos elementos.

1.3 Estándares de Metamodelado

1.3.1 XMI

Nació de la propuesta realizada por IBM, Unisys, y otras industrias líderes para realizar un nuevo estándar que combinase los beneficios de XML para la definición, validación y estandarización del formato de los documentos en la web con los beneficios de UML. XMI es un formato de intercambio para modelos basados en MOF y XML, es un estándar de OMG para el intercambio de información y meta-información entre herramientas, repositorios y aplicaciones, que proporciona un formato de intercambio para entornos distribuidos. XMI fue presentado en Junio del 98 como el pilar del intercambio de modelos de información, actualmente ha sido adoptado como una recomendación tecnológica de OMG desde el 23 de marzo de 1999. La versión de XMI 1.1 apareció el 1 de febrero

del 2000, actualmente se está desarrollando la versión 1.2 y se está trabajando para que XMI permita la generación automática en XML del Document Type Definition o Definición de tipos de documentos (DTD) para cada meta-información del modelo, esta es la evolución del estándar XMI , planteado en (Pérez, García, 2006).

1.3.2 UML

Originalmente, UML fue concebido por Grady Booch, James Rumbaugh e Ivar Jacobson de Rational Software. Posteriormente obtuvo el apoyo de la industria vía el consorcio de socios de UML, y fue presentado al OMG y aprobado por éste como estándar (17 de noviembre de 1997). Este lenguaje fue detallado en la sección 1.2.3.

1.3.3 OCL

En (Gómez, Sánchez, 2004) se plantea que es un lenguaje de especificación que permite definir modelos más precisos y completos, mediante expresiones que pueden establecer el cuerpo de una operación de consulta, un invariante de clase, las pre y poscondiciones de una operación, o especificar las reglas de derivación para atributos o asociaciones, que además es un lenguaje para expresar condiciones adicionales que no se pueden expresar con diagramas y cardinalidades.

Además de los estándares antes mencionados existen otros que no son los más utilizados mundialmente pero que sirven también para el metamodelado, entre ellos están:

QVT

OMG ha definido el lenguaje QVT para expresar transformaciones. La propuesta de estándar QVT, propone tres lenguajes de características diferentes: Relations, Core y Operational Mappings. El lenguaje Relations es un lenguaje declarativo, Operacional Mappings es imperativo, y ambos pueden implementarse sobre el lenguaje Core, que es un lenguaje de transformación con primitivas de bajo nivel. Es un estándar actualmente en desarrollo, que define el modo en que se lleva a cabo las transformaciones entre modelos cuyos lenguajes han sido definidos usando MOF, tomado de (Gómez, Sánchez, 2004).

CWM

Common Warehouse Metamodel o Metamodelo común de depósito (CWM). En (Súnico, 2007) se plantea que este estándar proporciona un mecanismo para la estandarización de los modelos de datos habituales en un determinado dominio a través de bases de datos y repositorios de modelos.

1.4 Herramientas de Metamodelado

Todas las herramientas de metamodelado poseen una característica en común y es que todas permiten definir metamodelos y generan editores de modelos, en esta sección se enumeran las herramientas libres y comerciales que pueden ser utilizadas para el metamodelado, estas son:

1.4.1 Herramientas MDA

AndroMDA, en (Mathhias, 2000) se plantean algunas características de esta herramienta, una de ellas es que esta herramienta es open source basada en plantillas para la generación de código J2EE desde modelos UML/XMI.

En [Hechavarría, 2007] se presentan una extensa lista de herramientas, que también pueden ser encontradas en la página de OMG sobre MDA, algunas de estas son:

Herramientas open source MDA

OpenMDX, un entorno MDA open source que genera código para las plataformas J2EE y .Net

VisualWADE, una herramienta desarrollada por el grupo de Ingeniería Web de la Universidad de Alicante. VisualWADE permite el diseño y generación automática de aplicaciones web siguiendo una aproximación Model Driven Development (MDD). Combina diagramas de dominio UML con nuevos modelos para representar la interacción con el usuario sobre un entorno hipermedia. El código intermedio generado es XML. En la versión actual, se proporciona un compilador de modelos que produce entregables en PHP a partir del código intermedio XML.

Herramientas comerciales MDA:

ArcStyler, herramienta comercial de Interactive Objects. Utiliza MOF para soportar estándares como XMI y UML, e Interfaz de Meta-dato de Java o Java Meta-data Interface (JMI). Integra

herramientas de modelado (UML) y desarrollo (ingeniería inversa, explorador de modelos basado en MOF, construcción y despliegue).

Codagen Architect, producto comercial integrado con varias herramientas comerciales UML.

OptimalJ de Compuware, usa notación de patrones para definir las transformaciones PSM[4] y MOF para soportar estándares como UML y XMI. Se trata de un entorno de desarrollo que permite generar aplicaciones J2EE completas a partir de un PIM[5].

También existen herramientas Case (Computer Aided Software Engineering) que permite realizar metamodelos, pero estas poseen una serie de limitaciones, ellas ofrecen posibilidades de dibujo para elaborar diagramas y facilidades de análisis para verificar el grado de corrección de los mismos. Una limitación de las herramientas Case es que permiten manipular sólo un conjunto limitado de tipos de diagrama y asimismo las herramientas Case que trabajan con UML poseen versiones inferiores a la actual 2.0 de ese paradigma6.

1.4.2 Case

ArgoUML Versión 0.24 (actualmente): Herramienta de código abierto y licencia libre, proyecto de Jason Robbins, 1999, es una herramienta para el modelado de sistemas, que se apoya en diagramas UML. Herramienta dirigida al apoyo de fases de planificación, análisis de requisitos y estrategia de desarrollo como se plantea en (Portelli, Bellissimo, 2006).

BOUML: En (Pérez, 2007) se despliega una serie de características como son: es una herramienta CASE gratuita (licencia GPL o Generic Public License), permite trabajar con UML 2.0, soporta gran cantidad de diagramas , es rápida y apenas consume memoria, es sencilla de utilizar, se puede generar código para Java, C++ e IDL (y controlar bastante la generación), y hacer reingeniería inversa (a partir del código sacar el modelo), también es capaz de generar documentación en varios formatos (HTML y XMI), puedes trabajar en grupo con sus módulos "Project Control" y "Project Synchro", es multiplataforma: Linux, MacOS y Windows.

[4] Modelos Específicos de la plataforma de Implementación, uno modelos de los modelos pertenecientes a MDA.

[5] Modelos Independientes de la plataforma de Implementación, uno modelos de los modelos pertenecientes a MDA.

[6] Conjuntos de conceptos y modelos.

Enterprise Architect: Enterprise Architect combina el poder de la última especificación UML 2.1 con alto rendimiento, interfaz intuitiva, para traer modelado avanzado al escritorio, y para el equipo completo de desarrollo e implementación. Es propietaria, se encuentra bajo la licencia EULA, solo para los sistemas operativos NP/Vista.

Como se expuso anteriormente las herramientas CASE presentaban limitaciones por lo que surgieron las METACASE que resolvían dichas restricciones, estas herramientas se le exponen a continuación:

1.4.3 Herramientas MetaCase

AToM3: (A Tool for Multi-Formalism Modelling and Meta-Modelling), Según (Zapata, Tamayo, Arango, 2007) , esta herramienta está escrita en lenguaje Python, es una herramienta usada para describir formalismos de cualquier tipo de esquema conceptual; esta herramienta MetaCase permite la creación, edición, transformación, simulación y optimización de metamodelos y modelos, y además la generación de código en lenguaje Python, AToM3 utiliza técnicas de reescritura gráfica y gramática de grafos para realizar las transformaciones entre formalismos.

DOME: (Domain Modeling Environment o Entorno de Modelado de Dominio): es una herramienta gratuita creada por Honeywell Technology Center. El formalismo que utiliza es una modificación del diagrama de clases de UML a nivel gráfico, que se especifica empleando código en los lenguajes Alter y Smalltalk, así lo expone el grupo de desarrollo de esta herramienta en (Grupo de Desarrollo, 2000).

MetaEdit+: de MetaCase, la herramienta Metaedit+ desarrollada por la empresa MetaCase, y destinada a la creación de Lenguajes Específicos del Dominio (DSL). Proporciona un entorno gráfico, amigable y fácil de usar, integra diferentes editores, navegadores de datos y otras herramientas. En Metaedit+, el lenguaje de metamodelado se denomina GOPPRR, que proviene de Graph, Object, Property, Port, Relationship y Role, que son los meta-tipos que proporciona la herramienta para describir los lenguajes de modelado. Se les llamarán a estos meta-tipos, Grafo, Objeto, Propiedad, Puerto, Relación y Rol, extraído de (Gómez, Sánchez, 2004).

GME: (Generis Models Environment o Entorno Genérico de Modelado) igualmente utiliza una simbología similar al diagrama de clases para la definición de los metamodelos, e incorpora la posibilidad de interpretación de restricciones utilizando OCL. El GME una excelente herramienta para la elaboración de modelos basados en UML; sin embargo, el modelado conceptual posee diagramas que no pertenecen al UML y que son igualmente importantes en el análisis, extraído de (Zapata, Álvarez, 2005).

1.5 Análisis y Diseño

El análisis de los requisitos es una tarea de ingeniería de software que cubre el hueco entre la definición del software a nivel sistema y el diseño del software El análisis de requisitos permite al ingeniero de sistemas especificar las características operacionales del software (función, datos y rendimientos), indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software.

El diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas (diseño, generación de código y pruebas) que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que de lugar por último a un software de computadora validado.

1.5.1 Análisis

En el análisis se pueden observar el producto obtenido (Pressman, 2001): Las descripciones de los objetos de datos, los diagramas entidad-relación, los diagramas de flujo de datos, los diagramas de transición de estados, las especificaciones del proceso y las especificaciones de control son creadas como resultado de las actividades del análisis.

Objetos de datos: Un objeto de datos es una representación de cualquier composición de información compuesta que deba comprender el software. Por composición de información, se entiende como todo aquello que tiene un número de propiedades o atributos diferentes. Por tanto, el

<<ancho>> (un valor sencillo) no sería un objeto de datos válido, pero las <<dimensiones>> (incorporando altura, ancho y profundidad) se podrían definir como objeto.

La especificación de control (EC): representa el comportamiento del sistema (al nivel al que se ha hecho referencia) de dos formas diferentes. La EC contiene un diagrama de transición de estados (DTE) que es una especificación secuencial del comportamiento. También puede contener una tabla de activación de procesos (TAP).

El diagrama entidad-relación: se centra sólo en los datos (y por consiguiente satisface el primer principio operacional de análisis), representando una <<red de dato>> su que existe para un sistema dado. El DER es específicamente útil para aplicaciones en donde los datos y las relaciones que gobiernan los datos son complejos.

El diagrama de flujo de datos (DFD): es una técnica que representa el flujo de la información y las transformaciones que se aplican a los datos al moverse desde la entrada hasta la salida.

El diagrama de transición de estados: representa el comportamiento de un sistema que muestra los estados y los sucesos que hacen que el sistema cambie de estado. Además, el DTE indica qué acciones se llevan a cabo como consecuencia de un suceso determinado.

Se usa la especificación de proceso (EP): para describir todos los procesos del modelo de flujo que aparecen en el nivel final de refinamiento. El contenido de la especificación de procesamiento puede incluir una narrativa textual, una descripción en lenguaje de diseño de programas (LDP) del algoritmo del proceso, ecuaciones matemáticas, tablas, diagramas o gráficos.

El modelo de análisis debe lograr tres objetivos primarios: (1) describir lo que requiere el cliente. (2) establecer una base para la creación de un diseño de software, y (3) definir un conjunto de requisitos que se pueda validar una vez que se construye el software.

1.5.2 Diseño

El diseño es una representación significativa de ingeniería de algo que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente, y al mismo tiempo la calidad se puede

evaluar y cotejar con el conjunto de criterios predefinidos para obtener un buen diseño. En el contexto de la ingeniería de software, el diseño se centra en cuatro áreas importantes: Datos, Arquitectura, Interfaces y Componentes. En estas áreas se aplican los principios del diseño.

El diseño del software se encuentra en el núcleo técnico de la ingeniería del software (Pressman, 2001) y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas: diseño, generación de código y pruebas que se requieren para construir y verificar el software.

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un plano para construir el software, inicialmente, el plano representa una visión holística del software. Esto es, el diseño se representa a un nivel alto de abstracción, un nivel que puede rastrearse directamente hasta conseguir el objetivo del sistema específico y según unos requisitos más detallados de comportamiento, funcionales y de datos.

A lo largo de todo el proceso de diseño, la calidad de la evolución del diseño se evalúa con una serie de revisiones técnicas formales o con las revisiones de diseño que sugiere tres características que sirven como guía para la evaluación de un buen diseño (Pressman, 2001):

1. El diseño deberá implementar todos los requisitos explícitos del modelo de análisis, y deberán ajustarse a todos los requisitos implícitos que desea el cliente.
2. El diseño deberá ser una guía legible y comprensible para aquellos que generan código y para aquellos que comprueban y consecuentemente, dan soporte al software.
3. El diseño deberá proporcionar una imagen completa del software, enfrentándose a los dominios de comportamiento, funcionales y de datos desde una perspectiva de implementación.

El diseño de software es tanto un proceso como un modelo (Pressman, 2001). El proceso de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario, es un conocimiento creativo, experiencia en el tema, un sentido de

lo que hace que un software sea bueno, y un compromiso general con la calidad son factores críticos de éxito para un diseño competente.

Existe una serie de principios para el diseño del software (Pressman, 2001): en el proceso de diseño no deberá utilizarse conjeturas, el diseño deberá poderse rastrear hasta el modelo de análisis, no deberá inventar nada que ya esté inventado, deberá minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara y presentar uniformidad e integración, tiene que estructurarse para admitir cambios, el diseño no es escribir código y escribir código no es diseñar.

En el diseño el producto obtenido: es una especificación del diseño el cual se compone de los modelos del diseño que describen los datos, arquitectura, interfaces y componentes. Cada una de estas partes es lo que forma el producto obtenido del proceso de diseño.

La Especificación del diseño aborda diferentes aspectos del modelo de diseño y se completa a medida que el diseñador refina su propia representación del software. En primer lugar, se describe el ámbito global del esfuerzo realizado en el diseño. La mayor parte de la información que se presenta aquí se deriva de la Especificación del sistema y del modelo de análisis (Especificación de los requisitos del software).

Resumiendo, el diseño es el núcleo técnico de la ingeniería del software. Durante el diseño se desarrollan, revisan y documentan los refinamientos progresivos de la estructura de datos, arquitectura, interfaces y datos procedimentales de los componentes del software. El diseño da como resultado representaciones del software para evaluar la calidad.

Durante las cuatro últimas décadas se han propuesto diferentes principios y conceptos fundamentales del diseño del software. Los principios del diseño sirven de guía al ingeniero del software a medida que avanza el proceso de diseño. Los conceptos de diseño proporcionan los criterios básicos para la calidad del diseño.

Capítulo 2: Metamodelado en Análisis y Diseño

Introducción

En el Desarrollo del Software Dirigido por Modelos se requieren lenguajes apropiados para describir todos los aspectos de los sistemas y a diferentes niveles de abstracción. Los lenguajes de modelado son específicos del dominio (DSL), y pueden ser definidos formalmente mediante un metamodelo, sintaxis abstracta y restricciones; o sintaxis concreta y semántica.

Las herramientas son el soporte que les permiten visualizar el metamodelo que se construye y los estándares permiten realizar una serie de acciones como el intercambio de modelos entre aplicaciones.

Para llevar a cabo la realización de un metamodelo existen un conjunto de elementos que son muy importantes y que no pueden dejar de estar presente, en este capítulo se analizarán elementos que tienen gran influencia en la elaboración de metamodelos, como los lenguajes, estándares y herramientas de metamodelado.

Este capítulo está dividido por cuatro epígrafes, en el primero se realizará un análisis general de cada uno de los lenguajes de metamodelado más extendidos, seguidamente en el segundo epígrafe se efectuará un análisis de los estándares más importantes de metamodelado. Posteriormente se plasmará en el epígrafe tres, una comparación de las herramientas para el metamodelado más usadas y extendidas y en un último epígrafe se confeccionará una propuesta de un lenguaje de metamodelado, así como las herramientas que permitan efectuar un buen metamodelado.

2.1 Análisis de los lenguajes de metamodelado

AL igual que los modelos que necesitan de un lenguaje de modelado para ser representados, también los metamodelos utilizan un lenguaje para ser confeccionados, estos son los lenguajes de metamodelado; lo que difieren entre ellos es que los lenguajes de modelado permiten la elaboración de modelos, los cuales son instancias de los metamodelos y son utilizados para visualizar el sistema que se quiere construir, sin embargo los lenguajes de metamodelado son utilizados para definir lenguajes de modelado, lo cual es realizado mediante un metamodelo. Se realizó una comparación de dos de los lenguajes más extendidos a través de los siguientes parámetros:

versión, herramientas que lo soportan, arquitectura e integración con otros estándares, además de otras características adicionales.

2.1.1 MOF

Un lenguaje de modelado se define mediante un metamodelo o descripción de los elementos del lenguaje y de las relaciones existentes entre ellos. MOF es un lenguaje para crear metamodelos (por ejemplo, el metamodelo de UML ha sido definido con MOF), y es, por tanto, un elemento básico de MDA ya que es uno de los pilares en que se apoya esta arquitectura. MOF es un ejemplo de un meta metamodelo o un modelo del metamodelo orientado a objetos por naturaleza. Define los elementos esenciales, sintaxis y estructuras de metamodelos que son utilizados para construir modelos orientados a objetos de sistemas MOF permite a los desarrolladores definir nuevos lenguajes a partir de metamodelos. Además de ser un lenguaje de metamodelado, también es un entorno para la especificación, análisis, generación y gestión de metamodelos independientes de la tecnología de implementación.

2.1.1.1 Versión

Versión 2.0.

2.1.1.2 Herramientas que soportan el lenguaje

Herramientas MDA como el AndroMDA (conocida también como Andrómeda), ya que este enfoque incluye este lenguaje como uno de sus pilares fundamentales.

2.1.1.3 Arquitectura de MOF

MOF establece una arquitectura de 4 capas, las cuales no son fijas, debido a que esto depende de como se quiera utilizar MOF, pues el uso de estas capas es solo un mecanismo que ayuda a entender las relaciones entre los diferentes tipos de datos y metadatos.

- M3: Metametamodelo: en esta capa es donde se define un lenguaje para especificar metamodelos.
- M2: Metamodelo: en esta capa se define el lenguaje para especificar modelos, cada elemento es una instancia del metametamodelo.
- M1: Modelo: cada elemento es una instancia del metamodelo.
- M0: Datos: aquí se tratan las instancias de elementos definidos en el modelo.

Para un mejor entendimiento ver la figura que se presenta a continuación:

Nivel	MOF	Ejemplo
M3	Modelo MOF (Meta-metamodelo)	
M2	Meta-modelo	UML
M1	Modelo	Modelo de Coche en UML
M0	Datos	Datos de un coche concreto

Figura 2: Ejemplo de arquitectura MOF.

2.1.1.4 Integración con otros estándares

UML 2.0: La alineación del metamodelo UML 2.0 con el metamodelo MOF simplificará el intercambio de modelos vía XMI y la interoperabilidad cruzada entre herramientas.

OCL: En MOF, como en UML, se emplea OCL para expresar requisitos adicionales sobre los datos. Un requisito se compone de nombre, lenguaje, expresión, política de evaluación y conjunto de elementos implicados.

XMI 2.0: Es un formato de intercambio para modelos basados en MOF y XML. Esto permite el intercambio de modelos entre aplicaciones.

2.1.1.5 Características adicionales

Es un lenguaje que utiliza un conjunto de construcciones que pueden ser usados para definir lenguajes de modelado, las construcciones básicas son las que se mencionan a continuación, extraído de (Súnico, 2007):

1. **Clases:** usadas para definir tipos de elementos en un lenguaje de modelado. Por ejemplo, la relación de dependencia de UML es una clase definida en MOF, que representa el tipo de todas las dependencias que pueden crearse en un modelo UML.

2. **Generalización:** define herencia entre clases. Por ejemplo UML::Classifier es una generalización de UML:: Class. La subclase hereda todas las características de la clase padre.
3. **Atributos:** usados para definir propiedades de elementos del modelo. Los atributos tienen un tipo y una multiplicidad.
4. **Asociaciones:** definen relaciones entre clases. Una relación tiene dos extremos, cada uno de los cuales puede tener definido un nombre de rol, navegabilidad y multiplicidad.
5. **Operaciones:** definen operaciones dentro del ámbito de una clase, junto con una lista de parámetros.

MOF define un entorno para la generación de repositorios para albergar los metadatos descritos por los metamodelos. Este entorno hace uso de transformaciones estandarizadas para convertir metamodelos MOF en API's7 para metadatos haciendo uso de (Interface Definition Language o Lenguaje para la definición de interfaces) IDL8.

Según (Súnico, 2007) a la hora de establecer un modelo MOF se suelen tomar en consideración dos perspectivas:

1. Perspectiva de Modelado de la Aplicación: Donde se define el modelo de información y donde, de alguna forma, se mira hacia las capas de menor nivel de abstracción.
2. Perspectiva de Modelado de Datos: Donde se define el modelo de datos y donde, de alguna forma, se mira hacia las capas de mayor o igual nivel de abstracción.

MOF tiene como responsabilidad permitir el intercambio de modelos y proporcionar una infraestructura que permite la realización de transformaciones de modelos. Con este fin se deben aportar medios para facilitar la gestión de todo modelo de datos asociado a un determinado modelo MOF. El OMG aporta con este fin un estándar de modelado, un metamodelo para aplicaciones de almacenaje y gestión de los metadatos asociados al metamodelo, CWM.

[7] Interfaz de Programación de Aplicaciones es el conjunto de funciones y procedimientos

[8] Un lenguaje para la definición de interfaces procedente de la especificación CORBA; en MOF se usa un subconjunto del lenguaje que en teoría no condiciona el modelado.

2.1.1.6 Ventajas de MOF

Por todas las características vistas anteriormente se puede expresar que este lenguaje representa una gran ventaja para los sistemas de modelado simple, pero además de esto permite el intercambio modelos y metamodelos entre elementos que usen el mismo metametamodelo, ayuda a los desarrolladores a añadir de forma incremental metamodelos y nuevos tipos de metadatos, ofrece la ventaja que soporta cualquier tipo de modelo y paradigma de modelado imaginable.

2.1.2 UML

Es un lenguaje de modelado visual, que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software, cuyo propósito general es el modelado orientado a objetos. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.

2.1.2.1 Versión:

Versión 2.0, aunque actualmente se está desarrollando una versión más avanzada, UML 2.1.

2.1.2.2 Herramientas que soportan el lenguaje

Existen un sinnúmero de herramientas que soportan este lenguaje, como son las MetaCase y las MDA. Muchas de las herramientas que se mencionan en este trabajo utilizan el UML2.0 como lenguaje de modelado, entre ellas están AndroMDA, DOME, el AtoM3 utiliza algunos de sus diagramas.

2.1.2.3 Arquitectura de UML

Se trata de alinear el metamodelo UML con el metamodelo MOF, para simplificar el intercambio de modelos, así como la interacción entre esas herramientas. Para esto MOF define una Arquitectura de lenguajes de modelado con cuatro capas o niveles:

- Nivel M3: Metametamodelo
- Nivel M2: Metamodelo
- Nivel M1: Modelo
- Nivel M0: Datos

En el UML 2.0 el metamodelo está perfectamente adaptado al metamodelo MOF.

2.1.2.4 Integración con otros estándares

XMI 2.0: Permite serializar de manera automática los modelos UML en forma de ficheros XML y de los metamodelos en forma de XML Schemas[9] (Esquema XML). Se recomienda su uso para el intercambio estándar de información entre herramientas UML y para el almacenamiento de los modelos en repositorios XML.

OCL: Como se había comentado anteriormente en UML, también se emplea OCL para expresar requisitos adicionales sobre los datos.

2.1.2.5 Características adicionales

El lenguaje de modelado UML 2.0 según (Portillo, 2004)**,** plantea que este lenguaje trae algunas características que lo hacen superior a sus otras versiones, por ejemplo: simplifica la sintaxis y semántica del lenguaje, presenta una alineación con MOF, se refina el modelo de desarrollo y el de proceso, gracias a la mejora de los diagramas de actividad, de interacciones y máquinas de estados, permite la definición de perfiles, hay una mejor representación de las relaciones.

Al igual que las versiones inferiores los diagramas de **UML 2.0** se agrupan en tres tipos:

1. Diagramas de comportamiento: Permiten exhibir comportamientos de un sistema o de los procesos de las organizaciones. Incluyen los diagramas de actividad, estado, caso típico y de interacción.
2. Diagramas de interacción: Es un subconjunto de los diagramas de comportamiento que permiten enfatizar las interacciones entre los objetos. Incluyen comunicación, vista general de interacciones, secuencia y diagrama de tiempo.
3. Diagramas de estructura: Muestran los elementos de una especificación que sean independientes del tiempo. Incluyen clase, estructura de componentes, componente, despliegue, objeto y diagramas de paquetes.

El metamodelo de UML2.0

El metamodelo de UML está dividido en tres paquetes principales:

[9] Es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa.

1. El paquete de fundamentos: define la estructura estática de UML.
2. El paquete de elementos de comportamiento: define la estructura dinámica de UML.
3. El paquete de administración del modelo: define la estructura organizativa de los modelos de UML.

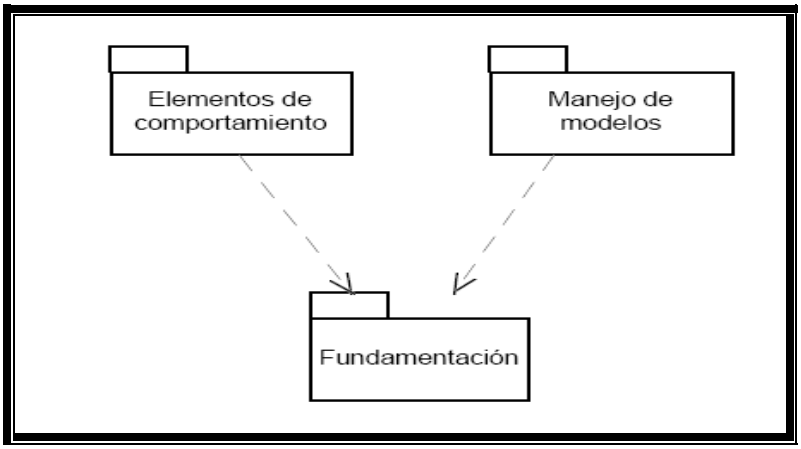


Figura 3: Paquetes del metamodelo de UML.

Paquete de fundamentos: El paquete de fundamentos contiene tres fundamentos.

El **núcleo** es el primer fundamento que describe las principales estructuras estáticas de UML. Entre ellas se cuentan los clasificadores, su contenido y sus relaciones. El contenido incluye atributo, operación, método y parámetro. Entre las relaciones se cuentan generalización, asociación y dependencia. También se definen varias metaclases abstractas tales como elemento generalizable, espacio de nombres y elemento del modelo.

El paquete también define plantilla y distintos tipos de subclases de dependencia así como componente, nodo y comentario. Como segundo se encuentran los **tipos de datos,** que describe las clases de tipos de datos que se utilizan en el metamodelo. El tercer fundamento es el **mecanismo de extensión,** el cual describe los mecanismos restricción, estereotipo y valor etiquetado.

Paquete de elementos de comportamiento

El paquete de comportamiento tiene un subpaquete para cada vista principal y además un paquete para estructuras de comportamiento, dentro de este paquete, se definen los elementos necesarios para representar la dinámica de un sistema, como son los casos de utilización (use cases: que

describe actor y caso de uso), los diagramas de interacción, los diagramas de actividades y los diagramas de estado o maquinas de estado que describe la estructura de una máquina de estados, incluyendo estado y distintas clases de pseudoestado, evento, señal, transición y condición de guarda. También describe estructuras adicionales para modelos de actividad tales como estado de acción, estado de actividad y estado de flujo de objeto, contiene además en paquete de colaboraciones que describe colaboración, interacción, mensaje, rol de clasificador y asociación y el paquete de comportamiento común describe señal, operación y acción. También describe instancias de clases correspondientes a diferentes descriptores.

Paquete de administración de modelos

El paquete de administración de modelos describe los paquetes, modelos y subsistemas. También describe las propiedades y visibilidad de los espacios de nombres y de los paquetes. No posee subpaquetes.

2.1.2.6 Ventajas de UML

El uso de UML 2.0 ha traído grandes ventajas a los desarrolladores ya que permite definir los límites del sistema y sus principales funciones, mediante Casos de Uso y actores, ilustrar el funcionamiento de un caso de uso mediante Diagramas de Interacción, representar la estructura de un sistema mediante Diagramas de Clases de manera más especifica y completa, modela el comportamiento de los objetos mediante Diagramas de Máquinas de Estados, describir la arquitectura de la implementación física con Diagramas de Componentes y Despliegue, permite extender la funcionalidad de elementos estándar mediante estereotipos y además permite que se puedan establecer restricciones adicionales sobre los datos mediante el uso de OCL y provee una base formal para los diagramas (metamodelo, OCL).

En esta nueva versión integra los siguientes diagramas: Diagrama de Estructura Paquete, Diagrama Compuesta, Diagrama Componente, Diagrama Despliegue, Diagrama Clases, Diagrama objeto, Diagrama Caso de Uso, Diagrama Activad, Diagrama de Maquina de Estado, Diagrama Secuencia, Diagrama General de Interacción, Diagrama de Tiempo, Diagrama de Comunicación. Todos estos diagramas brindan la posibilidad de realizar modelos más completos y precisos.

2.2 Análisis de los estándares de metamodelado

2.2.1 XMI 2.0

XMI es el nombre que recibe el estándar para el intercambio de metamodelos, modelos o datos en formato XML, pero según la naturaleza de la información intercambiada se distingue una arquitectura de tres ó cuatro niveles así lo plantea (Lorenzo, Peral, Sánchez, 2006). Su principal objetivo es permitir un intercambio de meta-información entre herramientas de modelado basadas en UML y repositorios de meta-información basados en MOF en entornos heterogéneos distribuidos. Integra tres estándares UML, MOF, XML.

La integración de estos estándares hace más general y factible para su uso. UML es un estándar que define un lenguaje de modelado orientado a objetos que es soportado por una gama de herramientas de diseño gráfico y MOF es un estándar que define un marco de trabajo para definir modelos de meta-información y proporciona herramientas con interfaces programadas para almacenar y acceder a meta-información en un repositorio. Es por esto, que el hecho de incluir tres estándares como XML, UML y MOF, permite a los desarrolladores de sistemas distribuidos compartir modelos de objetos y otra información sobre Internet. De esta forma se consigue un modelado, una gestión y una publicación de meta-información estándar a través de la web, utilizando UML y MOF para el diseño de metamodelos y XML para transferir la información.

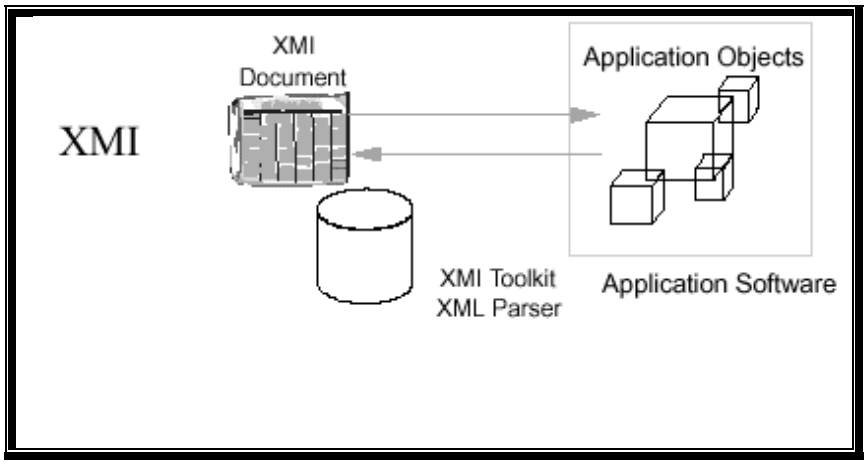


Figura 4: Intercambio con XMI (Pérez, García, 2006).

Arquitectura de XMI 2.0

Según lo plantea (Lorenzo, Peral, Sánchez, 2006), como se había dicho anteriormente, XMI utiliza una arquitectura de tres o cuatro niveles en dependencia de la naturaleza de la información intercambiada:

1. Intercambio de metamodelos o modelos. Se emplea una arquitectura de cuatro niveles (MOF / metamodelo / modelo / datos), donde MOF es un metametamodelo o un modelo de metamodelos. XMI regula, por una parte, el mapeo de metamodelos (definidos según MOF) y Schemas, y por otra parte, entre modelos (que tengan a MOF como metametamodelo) y documentos XML.

2. Intercambio de datos. Arquitectura de tres niveles (MOF / modelo / datos). Para poder emplear XMI, se necesita que el modelo de datos empleado se haya definido de acuerdo a MOF, es decir, se toma MOF como metamodelo de datos. En este caso, XMI permite mapear los modelos (diagrama de clases) a Schemas y los datos (diagrama de objetos) a documentos XML.

La arquitectura de XMI simplifica la comunicación entre diferentes aplicaciones y potencia la reutilización de objetos y componentes.

La comunicación entre n aplicaciones en un sistema que usa XMI utiliza n conexiones entre dichas aplicaciones, mientras que si no usara XMI serían necesarias más conexiones.

Utilizando XMI Sin utilizar XMI

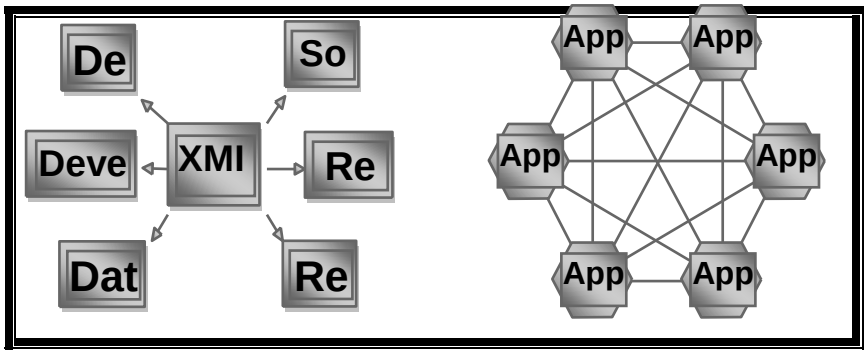


Figura 5: Tecnología con uso y sin uso de XMI (Lorenzo, Peral, Sánchez, 2006).

Ventajas de XMI 2.0

El uso del estándar XMI ha proporcionado un sinnúmero de ventajas que hace que su utilización sea cada vez mayor , algunas de las ventajas que se exponen en (Lorenzo, Peral, Sánchez, 2006) son: trabaja con Internet y estándares industriales como: XML, HTML, UML, MOF, utiliza un único formato de archivo para intercambio para todas las herramientas CASE, permite el intercambio de objetos entre aplicaciones, permite la reutilización de objetos y componentes, contiene un método sencillo de empaquetar información y metainformación, independencia a la hora de decidir la herramienta, el lenguaje o la tecnología que utilizar al desarrollar software.

Existe gran cantidad de herramientas que usan XMI: Rational Rose, Visual Paradigm, Poseidón UML, NetBeans, Metamill Software, ALTOVA, Microsoft Visio, Eclipse UML, entre otras.

Desventajas

El estándar presenta algunas desventajas: no permite recoger datos gráficos, incompatibilidad entre diferentes versiones.

2.2.2 UML 2.0

En (Portillo, 2004) se plantea que este estándar es aprobado por la compañía OMG como estándar de modelado orientado a objeto el 17 de noviembre de 1997, tres años después de su surgimiento durante el cambio de la versión 1.1 a 1.2. Este lenguaje es utilizado como se había dicho anteriormente para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas.

UML 2.0 propone una arquitectura de 4 capas similar a la arquitectura de MOF y MDA y además utiliza XMI para el intercambio de meta-información entre herramientas basada en UML. Debido a las similitudes entre el modelo MOF y los modelos de estructura del UML, los metamodelos MOF son usualmente modelados como diagramas de clase del UML.

Como se mostró en el apartado 2.1.2, la arquitectura de esta versión de UML se encuentra alineada a la arquitectura de cuatro capas definida por OMG, para cumplir con la especificación MOF.

Ventajas

Ofrece una forma estándar de modelar sistemas software, pudiendo utilizarse: con cualquier proceso de desarrollo, a lo largo de todo el ciclo de vida, con distintas tecnologías de implementación, puede usarse también en otras áreas, como la ingeniería de negocio, modelado de procesos, etc. En un lenguaje que se ha estandarizado y gracias a eso los desarrolladores pueden utilizar un lenguaje común para la representación del sistema.

Desventajas

Es un lenguaje que a pesar de tener muchas coincidencias con sus versiones anteriores ha tenido modificaciones que benefician a los desarrolladores.

2.2.3 OCL 2.0

OCL es un lenguaje de especificación formal fácil de leer y escribir, que fue definido por la OMG, se utiliza fundamentalmente en diagramas de clases. Permite expresar restricciones semánticas del sistema que no se pueden expresar a partir de una notación gráfica. De esta forma, los diagramas complementados con expresiones OCL son más precisos, su documentación es más clara, se mejora la comunicación entre desarrolladores (evitando errores producidos por malas interpretaciones) y la compresibilidad del sistema en etapas iniciales del desarrollo de software es mayor. OCL no es un lenguaje de programación, es un lenguaje que especifica cómo establecer restricciones adicionales sobre los datos.

Restricciones en OCL según (Castro, Saporiti, 2007)

Las restricciones que se pueden especificar con OCL son:

- Invariantes
- Definiciones
- Precondiciones
- Postcondiciones

Además de restricciones a nivel modelo, se puede definir también reglas a nivel metamodelo.

Invariante

Un invariante es una restricción que se liga a un Classifier10 (Class, Interface, etc). El propósito del invariante es definir una condición que debe ser válida siempre para todas las instancias de un Classifier. La restricción tiene el estereotipo << invariant >>.

Definición

Una definición es un "Constraint" que se liga a un Classifier. La variable o función definida puede utilizarse como una propiedad o una operación del correspondiente Classifier. El propósito de esta restricción es definir expresiones OCL reusables. La restricción tiene el estereotipo << definition >>.

Precondición

Una precondición es una restricción que se liga a un Operation de un Classifier. Esta restricción establece una condición que debe cumplirse antes de ejecutar la operación. La restricción tiene el estereotipo << precondition >>.

Postcondición

Una postcondición es una restricción que se liga a un Operation de un Classifier. El propósito de esta restricción es definir la condición que debe cumplirse luego de ejecutar la operación. Una postcondición consiste en una expresión OCL de tipo Boolean. La restricción tiene el estereotipo << postcondition >>.

A continuación se expone un ejemplo de cómo expresar una restricción adicional, por ejemplo se tiene el siguiente diagrama de clases Figura 3:

[10] Un Classifier es una clasificación de instancias. Describe un conjunto de instancias que tienen características comunes.

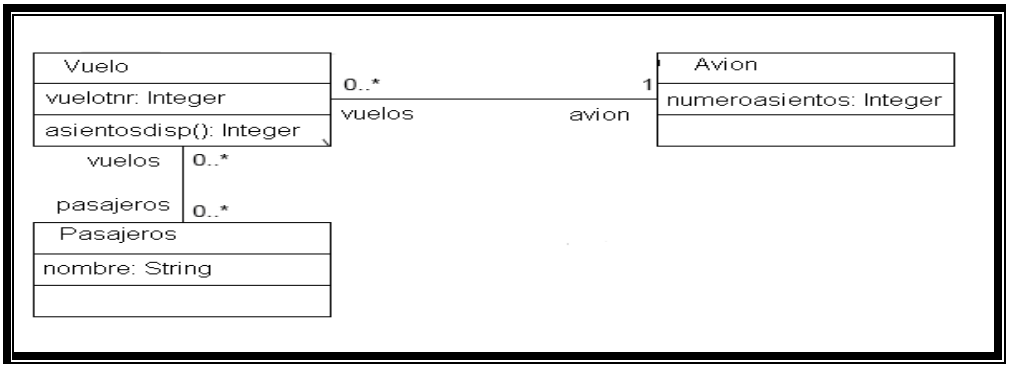


Figura 6: Diagrama de clases utilizado para el ejemplo con OCL.

Si se quiere expresar una condición adicional que establezca que en ningún vuelo puede haber más pasajeros que el número de asientos que tiene el avión, se realizaría de la siguiente manera:

Context Vuelo

Inv: pasajeros->capacidad () <= avion.numerodeasientos

Es un lenguaje con el que se puede definir modelos más precisos y completos. Algunos de los usos que se les da al lenguaje OCL en los modelos son:

1. Especificar valores iniciales de atributos.
2. Definir el cuerpo de operaciones de consulta.
3. Especificar las reglas de derivación para atributos o asociaciones
4. Expresar restricciones sobre clases o atributos.

OCL es un lenguaje para consultar y restringir los elementos de un modelo, y es parte integrante de UML. El término Constraint que aparece en el nombre OCL perdura de la primera versión de OCL, que sólo permitía definir restricciones. Sin embargo, la versión 2.0 de OCL proporciona un lenguaje de consultas mucho más general, con una expresividad similar a la de SQL[11].

En UML una restricción no es más que una declaración textual de una relación semántica expresada en un cierto lenguaje formal OCL.

[11] Lenguaje declarativo de acceso a bases de datos relacionales

2.3 Análisis de las herramientas de metamodelado

Como se ha planteado en la sección 1.5, son muchas las herramientas que de una forma u otra permiten crear metamodelos, entre estas herramientas se escogieron dos tipos: las MDA y las MetaCase. En una búsqueda exhaustiva en Internet y el estudio de documentos relacionados con el tema de las herramientas para el metamodelado, arrojó que las herramientas más utilizadas para entornos libres son: AndroMDA y AtoM3.

2.3.1 Herramientas MDA

Se entiende como herramienta MDA a aquellas que dan soporte al proceso de diseño y construcción de sistemas a través de la transformación automática de modelos.

¿Porque las MDA?

Se escogen estas herramientas porque están basadas en el enfoque MDA, que es el enfoque que se trata en esta investigación, por tanto ellas soportan todos los estándares que propone la OMG para este enfoque, además permite la creación metamodelos.

En los últimos años se han propuesto diversos criterios para la clasificación de las herramientas MDA. De especial interés son las propuestas de (Kent S, 2002), (Jézéquel, 2005), (Czarnecki, Helsen, 2003) y (Tariq, Akhter, 2005).

Características según (Kent S, 2002)

Están orientadas a verificar que los modelos estén bien formados, permiten trabajar con instancias de modelos, ofrecen soporte para la transformación entre modelos, permite la verificación orientada a modelos, apoyan los procesos de desarrollo de software dirigido por modelos.

Características según (Jézéquel, 2005)

Plantea que las herramientas MDA son:

1. Lenguajes de Programación de Propósito General
2. Herramienta de Transformación Genérica
3. Herramientas CASE de Lenguajes Interpretados
4. Dedicadas a la Transformación de Modelos

5. De Meta-Metamodelado

Características según (Czarnecki, Helsen, 2003)

Plantea que son un enfoque Modelo a Código, basado en el patrón "visitor ", basado en plantillas, Modelo a Modelo, enfoque de manipulación directa, relacional, basado en transformación de grafos, centrado en la estructura y un enfoque híbridos.

Características según (Tariq, Akhter, 2005)

Plantea que estas herramientas, son soporte de CIM, PIM, son compatibles los modelos MOF, y que además son soporte para: Diagrama de Caso de Uso, Diagrama de Clase, Diagrama se Secuencia, UML, XMI-XML, soporta estándares como Query-View-Transformation y OCL, realiza ttransformación PIM -PSM y directa de PIM a Código, son herramientas basadas en metamodelos, modelos, patrones, perfiles UML, permite el registro de transformaciones, la trazabilidad de transformaciones, combina 2 o más CIM en 1.

2.3.2 Herramientas MetaCase

¿Por qué las MetaCase y no las Case?

Las herramientas CASE, ofrecen la facilidad de poder elaborar los diagramas y verificar el grado de corrección de los mismos. Una limitación importante de las herramientas CASE es que permiten manipular sólo un conjunto limitado de tipos de diagrama y verificar sólo un conjunto limitado de restricciones sobre los diagramas de estos tipos. Como una respuesta a estas limitaciones han surgido herramientas CASE con facilidades de metamodelado (habitualmente denominadas MetaCases), existen dos tipos de herramientas CASE: las convencionales y las MetaCase.

Herramientas CASE convencionales

Las herramientas CASE son sistemas de software que tienen la intención de proveer soporte automático para las actividades del proceso de desarrollo de software. Bajo esta definición se pueden agrupar diversas herramientas que van desde asistentes para la construcción de gráficos, hasta herramientas que permiten la elaboración de modelos y su conversión a piezas más o menos completas de código ejecutable, como es el caso de productos como el ArgoUML.

La característica fundamental de las herramientas CASE es que sólo se pueden elaborar los diagramas que vienen por defecto definidos internamente en la herramienta de modelado. En una herramienta típica de diagramas UML, tal como el ArgoUML, el analista puede elaborar diagramas de casos de uso, clases, transición de estados, colaboración o secuencias, por mencionar sólo algunos. Por tener tipos de diagramas definidos internamente, no le es posible al analista realizar modificaciones sobre estos tipos, limitando de este modo los diferentes elementos que se hayan definido en cada tipo de modelo. Entre estos elementos se incluyen las diferentes restricciones que pueden ser verificadas en los diferentes diagramas que considere la herramienta.

Sin embargo, en el caso del ArgoUML no se manejan algunas de las restricciones correspondientes a la versión 2.0 de UML, pues en su formalismo se incluyeron las restricciones correspondientes a la versión 1.5 de UML.

Herramientas MetaCase

Las herramientas de Metamodelado (MetaCase) han surgido como alternativas a las herramientas CASE convencionales, con el fin de permitir a sus usuarios la creación de nuevos formalismos para diagramas que no se encuentren disponibles en la herramienta, o van a ser elaborados, además también para la complementación de las diferentes restricciones y reglas de consistencia internas de los diagramas que se pueden elaborar en la herramienta.

Entre las diferentes herramientas de Metamodelado se encuentran Generic Model Environment – GME, el DOME, el AToM3 y el MetaEdit+, entre otros, de las cuales GME y MetaEdit+ son comerciales. Con una herramienta MetaCase es posible describir de manera completa un tipo de diagrama cualquiera, de interés para un problema específico. Una vez el tipo de diagrama ha sido descrito, la herramienta se puede usar para elaborar instancias del mismo.

Ventajas principales de las herramientas de Metamodelado (MetaCase) sobre las herramientas CASE convencionales, según se plantea en (Zapata, Álvarez, Arango, 2005), son las siguientes:

1. Se puede iniciar la construcción de formalismos de modelos desde cero, lo cual implica la posibilidad de correcciones (adición o borrado de elementos, por ejemplo) en modelos que cambian constantemente de versión o en otros que presentan modificaciones sutiles frecuentemente.

2. Facilidad gráfica de expresión del formalismo de los diferentes modelos, lo que reduce la complejidad en la comprensión de los mismos.

3. Posibilidad de creación de instancias de los modelos y de verificación de las restricciones planteadas en el metamodelo sobre esas instancias en particular.

Herramientas seleccionadas:

2.3.1 AndroMDA 3.2

Es una herramienta open source, desarrollada bajo la comunidad AndroMDA, es una herramienta multiplataforma lo cual especifica que lo mismo puede ser instalada en Windows como en Linux, se encuentra bajo licencia BSD, es una herramienta informática que genera código extensible, y en diferentes lenguajes de programación, y que además se adhiere al paradigma MDA.

Debido a que su generador de código soporta plataformas actuales, se ha convertido en la principal herramienta de código abierto de MDA para el desarrollo de aplicaciones empresariales. Utiliza como lenguajes para crear metamodelos el metamodelo de UML y MOF. AndroMDA, también conocida como Andrómeda está compuesta por *cartridges*[12]. Los *cartridges* son un tipo de *plug-ins*[13], que es donde se definen los metamodelos y reglas de transformación para transformar elementos del modelo de acuerdo al metamodelo. En muchos casos, un *cartridge* puede contener solamente los metamodelos, ya que las reglas de transformación pueden ser manejadas por muchos *cartridges* y pueden ser contenidas en un *cartridge* común. La tecnología de *cartridge*, le permite obtener modelos en diferentes plataformas, una vez que se tienen definidos los *cartridges*, las transformaciones entre los modelos de diferentes niveles se hacen automáticamente. El lenguaje utilizado para definir los cartridges es JAVA.

Características generales de la herramienta:

1. Apoyan los procesos de desarrollo de software dirigido por modelos
2. Dedicadas a la Transformación de Modelos
3. Enfoque basado en plantillas

[12] Cartuchos. Se mantiene el término en inglés para no alterar su significado.

[13] Es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

4. Soporte para XMI-XML
5. Transformación basada en patrones.
6. Soporte para UML 2.0 y herramientas basadas en Eclipse EMF.
7. Generación de código para PSM metamodelo de clases.
8. Generación de código para Java.
9. Mejora en la documentación y nuevos tutoriales.
10. Corrección de errores y pequeñas mejoras.

2.3.2 AToM3 0.2.2

Es una herramienta para el multiformalismo y metamodelado. Según (Zapata, Álvarez, 2005) está escrita en el lenguaje de programación Phyton. En (Zapata, Tamayo, Arango, 2007) se plantea que en AToM3, los metamodelos son creados y editados en un ambiente que emplea como formalismo gráfico el diagrama Entidad-Relación. Esta herramienta posee un metametamodelo inicial basado en el modelo Entidad-Relación extendido con restricciones, que permite la definición de los diferentes metamodelos en un entorno gráfico. De esta manera, se puede definir cualquier tipo de metamodelo en términos de las "entidades" que hacen parte del mismo y sus posibles interconexiones o "relaciones". Una vez definido el metamodelo, se puede emplear su definición para construir los modelos pertinentes a un problema específico del mundo.

La Gramática de Grafos en AToM3 permite expresar restricciones y describir las transformaciones entre modelos de manera gráfica. La gramática de grafos incluida en AToM3 define reglas que poseen un lado izquierdo (left-hand side o LHS) donde se definen de manera gráfica las precondiciones necesarias para disparar una regla y un lado derecho (right-hand side o RHS) que contiene el modelo que reemplazará el que equipare el lado izquierdo de la regla. Para las reglas expresadas de esta manera también se deben definir unas condiciones y unas acciones para ejecutar cuando la regla se active. La Gramática de Grafos de AToM3 posee también un mecanismo que va rescribiendo el modelo a medida que las diferentes reglas se van activando hasta que no haya reglas que se puedan ejecutar (De Lara, Vangheluwe, Alfonseca, 2003).

Criterios de comparación

Para la selección de una propuesta es de primera necesidad establecer una serie de criterios que permitan establecer una comparación entre las herramientas. Dichos criterios se agrupan bajo tres puntos de vista: características generales, lenguajes empleados, metamodelo.

Características Generales:

- **Plataforma:** Aquí se verificará en cuales plataformas puede ser instalada la herramienta.
- **Documentación y soporte:** Aquí se sabrá si estas herramientas cuentan con tutoriales y guías de usuarios para su manejo.
- **Donde está enfocada la herramienta:** Aquí se percibirá hacia que parte del proceso de desarrollo de software está enfocada.
- **Soporte de XMI:** Este criterio es para saber si la herramienta soporta el estándar XMI para intercambio de información.
- **Interoperabilidad entre herramientas:** Aquí se apreciará si es posible integrar estas herramientas con otras.
- **Licencia:** Aquí se puede contemplar bajo que licencia se distribuye la herramienta.

Lenguajes empleados:

- **Lenguaje base:** Contempla el lenguaje en que fue creada la herramienta.
- **Lenguaje interno:** Lenguajes con los cuales se pueden realizar acciones al interior de la herramienta.

Metamodelo:

- **Paradigma de modelado:** 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.

Criterios		AndroMDA
Plataforma	Linux, Unix and MS Windows	Multiplataforma
Documentación y soporte	Si	Si
Licencia	No especificada	BSD
Soporte XMI	Si	Si
Interoperabilidad entre herramientas	Integración con otras herramientas	Integración con otras herramientas
Donde está enfocada la herramienta.	Análisis y Diseño	Apoyan todo el proceso de desarrollo de software dirigido por modelos.
Lenguajes Empleados		
Lenguaje base	Python: lenguaje procedural	JAVA
Lenguaje interno	Python: lenguaje procedural	OCL: lenguaje funcional, Utiliza Java para definir cartridges.
Metamodelo		
Paradigma de modelación	Entidad- Relación Extendido	meta-modelo de UML, soporta MOF.
Visualización gráfica	Si	Si

Figura 7: Comparación entre AtoM3 y AndroMDA.

2.4 Propuesta Final

En MDA el metamodelo actúa como un mecanismo para definir lenguajes de modelado, de forma que su definición no sea ambigua. Esta no ambigüedad permite que una herramienta pueda leer, escribir y entender modelos como los construidos con UML 2.0. En correspondencia con la utilidad de los metamodelos en MDA y la utilización de estándares de metamodelados integrados, después de haberse realizado un análisis cada uno de los criterios de comparación, se realizan las siguientes propuestas:

Como primera propuesta se tiene a la herramienta AndroMDA 3.2 con el lenguaje MOF, la selección de esta herramienta como propuesta se basa en que la herramienta es libre, multiplataforma, se encuentra bajo la licencia BSD, apoya todo el proceso de desarrollo del software, utiliza MOF para crear metamodelos y también permite elaborar metamodelos a partir del metamodelo de UML, es una herramienta que integra estándares y lenguajes como UML para modelar, utiliza OCL para las expresar restricciones y el uso del estándar XMI le facilita el intercambio de información y meta-información, abriendo un amplio abanico de posibilidades al trabajo en equipo, permite nuevas colaboraciones entre sectores con diferentes tecnologías, simplifica la comunicación entre aplicaciones. En AndroMDA el código generado puede ser utilizado en diversas aplicaciones, crear diversos *cartridges* o modificar los existentes, todos estos *cartridges* son utilizados para transformar modelos.

Como segunda propuesta se tiene la herramienta AtoM3 con su formalismo (diagrama de entidad-relación extendido), es una herramienta de metamodelado que permite la generación rápida de entornos de modelado especializados y para el modelado multi-paradigma (que incluye metamodelado, modelado multi-formalismo y en múltiples niveles de abstracción). Una de las mayores diferencias con otras herramientas es que en AtoM3 todo es un modelo, o ha sido definido mediante un modelo y por tanto, el usuario puede cambiarlo.

Esta herramienta permite la creación, edición, transformación, simulación y optimización de metamodelos y modelos. En AToM3, los metamodelos son creados y editados en un ambiente que emplea como formalismo gráfico el diagrama entidad-relación, este formalismo propio permite la expresión de diferentes modelos, incluyendo sus restricciones. Esta herramienta no trabaja con los

lenguajes analizados como UML, MOF, pero tiene otras posibilidades que favorecen al desarrollo de sistemas, como es el uso del formalismo antes mencionado, además AtoM3 tiene la posibilidad de realizar conversiones entre modelos empleando para ello la Gramática de Grafos, un mecanismo también programado en el lenguaje Phyton. En la Gramática de Grafos las reglas de transformación se expresan de manera gráfica y no a modo de texto.

Por último se propone la creación de una nueva herramienta, basada en tomar lo mejor de las dos anteriores, es decir, tomar la herramienta más general que en este caso es la herramienta AndroMDA y se le vincule el formalismo que presenta la herramienta AtoM3, el diagrama entidad-relación, para que AndroMDA sea capaz de crear metamodelos mediante MOF y el formalismo (diagrama entidad-relación), puesto que este le proporciona una mejora a esta herramienta en la definición de metamodelos y modelos a cualquier nivel de abstracción, además permitirle a esta herramienta el uso de la “Gramática de Grafos”, dándole la posibilidad de definir una transformación entre dos tipos de modelos y que sus reglas de transformación se expresan de manera gráfica y no a modo de texto.

Trabajos Futuros

Este trabajo abarca dos temas que en el mundo a nivel de grandes empresas se está utilizando por las ventajas y aportes que traen consigo para el desarrollo de software, debido a esto se proponen los siguientes trabajos futuros:

1. Aplicar la arquitectura MDA a un proceso de desarrollo de software.
2. Poner en práctica la técnica de metamodelado en un proyecto de la Universidad utilizando un lenguaje de metamodelado y una herramienta de las que se proponen en este documento.
3. Extender este trabajo hasta la fase de implementación y prueba.
4. Realizar una unión de las dos herramientas propuesta.

Conclusiones

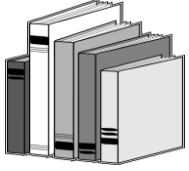
Con el desarrollo de este trabajo de diploma se cumplieron los objetivos propuestos, arribando a las siguientes conclusiones:

1. Se obtuvieron varias propuestas que pudieran servir para mejorar el desarrollo de software en la UCI, logrando así un alcance mayor del previsto inicialmente.
2. El lenguaje MOF es la mejor opción disponible como lenguaje para el metamodelado, pues puede ser utilizado tanto para definir metamodelos, como lenguajes de modelado.
3. La herramienta AndroMDA tiene un uso muy extendido y puede ser utilizada en todo el proceso de desarrollo de software.
4. La herramienta AtoM3 posee un fuerte basamento formal, lo que ofrece la posibilidad de modelar situaciones no previstas inicialmente en la herramienta.
5. La combinación de AndroMDA y AtoM3 puede dar lugar a una herramienta completa y con un formalismo muy fuerte.

Recomendaciones

Para dar continuidad al presente trabajo y teniendo en cuenta las experiencias adquiridas durante el desarrollo del mismo, se recomienda:

1. Realizar un estudio más profundo de la arquitectura propuesta por OMG, la arquitectura MDA, ya que está tomando un gran auge en la actualidad y abarca otras áreas de investigación, como es la transformación entre modelos.
2. Profundizar en los lenguajes MOF y UML, para brindarle la posibilidad a los desarrolladores de poder crear sus propios lenguajes de modelado a partir de estos.
3. Mantener vigilancia tecnológica acerca de nuevos estándares para el metamodelado.
4. Extender la investigación realizada a todo el proceso de desarrollo de software.



Bibliografía

Referencias bibliográficas

ANAHEIM. 2003. Generative Techniques in the context of MDA. California. [En línea] Octubre del 2003. http://www.softmetaware.com/oopsla2003/mdaworkshop.html.

ARIQ, N. A., Akther, N. Comparison of Model Driven Architecture (MDA) based Tools. Department of Computer and Systems Sciences: Royal institute of Technology (KTH). 2005.

BELLINGER, Gene. Modeling & Simulation. Outsights Corp. 1997.

BOEHMB, W. Software Engineering ,IEEE Fansactions on Computers, 12 de diciembre de 1976 C-25, 1226-1241 pg.

BOOCH, Grady, JACOBSON, Ivar y RUMBAUGH, James. Lenguaje unificado de modelado: Manual de referencia, 1998. 528 pg. ISBN: 0-201-57168-4.

BOOCH, Grady, JACOBSON, Ivar y RUMBAUGH, James. El Proceso Unificado de Desarrollo de Software. Addison Wesley. Madrid. 2000. 458 pg. ISBN: 84-7829-036-2

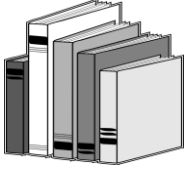
BUDINSKY, Frank. Eclipse Modelling Framework: Developer's Guide, 2003.

CASTRO Manacero, María Noel, SAPORITI, Martin. Generación de Micromundos para Transformaciones de Modelos. Licenciatura en Informática. Argentina. Universidad Nacional de La Plata: Facultad de Informática UNLP. Noviembre del 2007. 126 pg.

CLARK, Tony, EVANS, Andy, SAMMUT, Paul, WILLIANS, James. Metamodelling: A Foundation for Language Driven Development Versión 1.0. Octubre del 2007. <http://www.ceteva.com/book.html>

COLLABNET. Enterprise Edition. 2006. Bill Portelli, Mike Bellissimo. http://www.tigris.org/

Bibliografía



Conversion de Esquemas Preconceptuales a Diagrama de Casos de Uso usando AtoM3. J, Carlos M **ZAPATA, O, TAMAYO, Paula A. y ARANGO, Fernándo.** 2007. 153, Medellín : Revista Dyna, 2007, Vol. 74. 0012-7353.

COTA, A. Ingeniería de Software, Soluciones Avanzadas. Julio de 1994.5-13 pg.

CZAMECKI, K. y HELSEN, S. Classification of Model Transformation Approaches: Workshop on Generative Techniques in the Context of Model-Driven Architecture. University of Waterloo, Canada, 2003.

DE LARA, J, VANGHELUWE, H. y ALFONSECA, M. Using Meta-Modelling and Graph- Grammars to Create Modelling Environments. Electronic Notes in Theoretical Computer .2003. Science, vol. 72, N° 3.

FAVRE, J.M. Towards a Basic Theory to Model Driven Engineering. Workshop in Software Model Engineering. 2004.

GARCÍA, Jesús. Tema 2. Fundamentos del DSDM. Metamodelado. Murcia. 13 de junio del 2008, 99 pg.

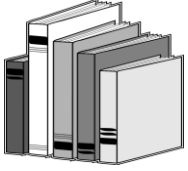
GÓMEZ, Pablo y SÁNCHEZ, Oscar. Herramientas de Metamodelado: Microsoft DSL Tools vs MetaEdit+. Murcia. 2004. 243 pg.

HECHAVARRÍA, Dayra Iris. Modelos Semánticos para la Gestión de Requisitos, Trabajo de diploma para optar por el titulo de Ingeniero Informático. La Habana. Universidad de las Ciencia Informáticas. Julio del 2007.105 pg.

Honeywell. 24 de agosto del 2000.desarrolladores del proyecto. http://www.htc.honeywell.com/dome/description.htm.

INTI, Rosario . Desarrollo dirigido por modelos (MDD). España. Junio 2007. 87 pg.

Bibliografía



JEZEQUEL, J.M. Model Transformation Techniques. Universidad de Rennes. 2005. Disponible en: http://www.irisa.fr/prive/jezequel/enseignement/ModelTransfo.pdf.

KENT, S. Model Driven Engineering: Proceeding of IFM. Springer, 2002.

LORENZO, Sara, PERAL, Almudena y SANCHEZ, Ester. XMI: XML Metadata Interchange.Valencia. 2006. 19 pg.

MATHHIAS, Bohlen. 2006. AndroMDA. Viernes 10 de noviembre del. http://www.andromda.org

Mi e-Oasis. **José María Súnico.** 3 de octubre del 2007. http://jm.sunico.org/2007/10/03/diario-de-a-bordo-fecha-estelar-2007-10-03.

MOORE, Bill. Eclipse Development using Graphical Editing Framework and the Eclipse Modelling Framework. IBM Redbooks, 2004.

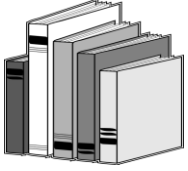
PÉREZ, J. GARCIA, M. XMI: XML Metadata Interchange. Valencia. 2006. 19 pg.

PÉREZ, Alejandro. 2007. Adictos al Trabajo. [En línea] 13 de julio de 2007. www.adictosaltrabajo.com.

PORTILLO, Javier. Entorno Multidisciplinar para el Desarrollo de Sistemas de Control Distribuido con Requisitos de Tiempo Real. Tesis doctoral. Bilbao. Escuela Superior de Ingenieros de Bilbao. 2004. Capítulo 2, 1-99 pg.

PRESSMAN, Roger S. Ingeniería de Software: Un Enfoque Practico, quinta edición, Madrid. 2001. 614 pg.

QUISPE-OTAZU, Rodolfo. Proceso Unificado de Desarrollo de Software, 20 de agosto del 2007.



SOURROUILLE, J. y CAPLAT, G. A pragmatic View about Consistency Checking of UML Models. Proceedings of the Workshop on Consistency Problems in UML-Based Software Development. San Francisco. 2003. 43-50 pg.

UBEDA, Jorge.2006.Hacia la Cuarta Generación del Software. [En línea] 18 de Abril de 2006.[Citado el: 20 de enero de 2008]. Disponible en: http://cuartageneracion.blogspot.com/2006/04/una-evaluacin-integradora-de-mdasf.html

ZAPATA, Carlos M., ALVAREZ, Carlos A. Conversión de diagramas de procesos en diagramas de caso de uso usando AtoM3. Colombia. Julio 2005. 103-113 pg. ISSN 0012-5373.

ZAPATA, Carlos M., ALVAREZ, Carlos A., Y ARANGO I., Fernando. Propuesta para el manejo de restricciones en modelos de clases usando AtoM3.Número 17.Enero-Junio, 2005.17: 130-147 pg. ISSN: 0122-3461.

Bibliografías Consultadas:

Disponible en: http://www.quasartecnologia.com/Metamodelado.htm - 3 de abril del 2008

ZAPATA J. Carlos M. / ÁLVAREZ.Carlos Alberto 2005. Conversión de diagramas de procesos en diagramas de caso de uso usando AtoM3., Dyna, julio, año/vol. 72, número 146 Universidad Nacional de Colombia, Medellín, Colombia.

Disponible en: http://dis.um.es/~jmolina/Pfc/DSLvsMetaedit.pdf

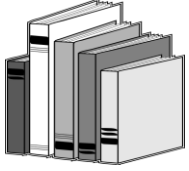
Disponible en: http://alarcos.inf-cr.uclm.es/doc/masi/doc/lec/parte3/fruiz-mifisis04.pdf

Disponible en: http://ieeexplore.ieee.org/iel5/9907/34417/01642458.pdf

Disponible en: http://www.inti.gov.ar/ue/pdf/presentacionJoseba_Laka.pdf

Disponible en: http://cuartageneracion.blogspot.com/2005/06/microsoft-y-mda-mdd-domain-specific.html

Bibliografía



PADRÓN Lorenzo, J., GARCÍA Luna, J.D., SÁNCHEZ Rebull, E.V., GARCÍA. Estévez. A Open Canarias SL, Santa Cruz deTenerife, España, http: //www.opencanarias.com, info@opencanarias.com.

Favre, Liliana. 2006. Arquitectura de software dirigida por modelos (Model-Driven Architecture) UNCPBA, 2006.

JARAMILLO, Juan de Lara. Simulación Educativa Mediante Metamodelado y Gramáticas de Grafos, Escuela Politécnica Superior, Ingeniería Informática, Universidad Autónoma de Madrid, e-mail: Juan.Lara@ii.uam.es .

GONZÁLEZ, Jaime.1998. El discreto encanto del metamodelo de UML, publicado en el número 60, de agosto de 1998, de la revista Soluciones Avanzadas.

GARCÍA, Sara Lorenzo. RODRÍGEZ Peral , Almudena. SÁNCHEZ Jiménez, Ester. XMI, XML Metadata Interchange.

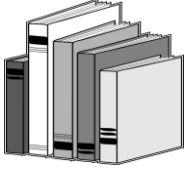
HECHAVARRÍA, Dayra Iris. Modelos Semánticos para la Gestión de Requisitos, Trabajo de diploma para optar por el titulo de Ingeniero Informático. La Habana. Universidad de las Ciencia Informáticas. Julio del 2007.105 pg.

JARAMILLO, Juan de Lara. Modelado de Software con UML2.0, Parte I: Notación, http: //uml.org, jdelara@uam.es Escuela Politécnica Superior, Universidad Autónoma de Madrid.

JARAMILLO, Juan de Lara. Modelado de Software con UML2.0, Parte II: Notación, http: //uml.org, jdelara@uam.es Escuela Politécnica Superior, Universidad Autónoma de Madrid.

Disponible en: http://www.ehu.es/javier.portillo/tesia/2-ESTADO%20DEL%20ARTE.pdf.

J. García Molina, J. Rodríguez, M. Menárguez, M.J. Ortín, J. Sánchez. Un estudio comparativo de dos herramientas MDA: OptimalJ y ArcStyler, Departamento de Informática y Sistemas, Universidad de Murcia, Campus de Espinardo 30071 Murcia, jmolina@um.es.



Disponible en:http://jm.sunico.org/2007/10/03/viabilidad-del-uso-de-mda-para-codiseno-hw-sw-i/#more-232.

FUENTES, Lidia y VALLECILLO, Antonio. Una Introducción a los Perfiles UML, Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Campus de Teatinos. E29071-Málaga (SPAIN), e-mail: {lff,av}@lcc.uma.es.
Dña. Begoña Cristina Pelayo García-Bustelo, Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos, Oviedo mayo 2007.

CASTRO Manacero, María Noel. SAPOTIRI, Martin. Generación de Micromundos para transformaciones de modelos, noviembre del 2007.

INTI, Rosario . Desarrollo dirigido por modelos (MDD). España. Junio 2007. 87 pg.

Disponible en: http://tasof-ucn.blogspot.com/2005/10/de-modelos-metamodelos-y.html.

Disponible en: http://astreo.ii.uam.es/~jlara/doctorado.2006/UML_2.0.pdf/.

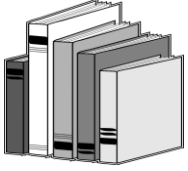
M. Reina, J. Torres, M. Toro. Octubre – 2006, Hacia Lenguajes de Metamodelado Orientados a Aspectos,

ZAPATA, O, TAMAYO, Paula A. y ARANGO, Fernándo. Conversión de Esquemas Preconceptuales a Diagrama de Casos de Uso Empleando AToM3, agosto 08 de 2007.

V. Pelechano, Estévez, y Vallecillo, A. Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM 2005), (Eds.), 13 de septiembre de 2005, http://www.dsic.upv.es/workshops/dsdm05/.

GARCÍA Molina, Jesús. Fundamentos del DSDM. Metamodelado, Departamento de Informática y Sistemas, Universidad de Murcia, http://dis.um.es/~jmolina.

Bibliografía



BOLLATI, Verónica, M, Juan, MARCOS, Vara Belén Vela Esperanza. Una revisión de herramientas MDA. http://www.andromda.org. Página de la herramienta AndroMDA.

CLARK, Tony, EVANS, Andy, SAMMUT, Paul, WILLIANS, James. Superlanguages. Developing Languages and Applications with XMF, First Edition, 2008.

GARCÍA, Iván, GÓMEZ-SANZ, Jorge J. y PAVÓN, Juan. Representación de las Relaciones en los Metamodelos con el Lenguaje Ecore.

Meta Object Facility(MOF) Specification. Abril 2002,OMG,IBM,Oracle Corporation, entre otras.

ZAPATA, Carlos M., ALVAREZ, Carlos A., Y ARANGO I., Fernando. Propuesta para el manejo de restricciones en modelos de clases usando AtoM3.Número 17.Enero-Junio, 2005.17: 130-147 pg. ISSN: 0122-3461.

PONS, Claudia, GIANDINI, Roxana. UML y OCL, Prof. Dra. 2007.

HERRERA, Juan Carlos, MATTEO, Alfredo e DÍAZ, Isabel. Una Caracterización de Herramientas MDA de Código Abierto†, 2005

ZAPATA J, Carlos M, ARANGO I, PÁEZ, Fernando Raquel Anaya. Estudio Comparativo de las herramientas MetaCase, enero 2007.

RODRÍGUEZ Vicente, Jesús, GARCÍA Molina, Jesús J. Estudio Comparativo de OptimalJ y ArcStyler, junio 2004.

Disponible en: http://www.metamodel.com/staticpages/index.php?page=20021010231056977.

Disponible en: http://www.cetus-links.org:80/oo_uml.html/xmi.html

Anexos

Anexo 1

Encuesta de metamodelado

1. **¿Qué conocimientos posees acerca del metamodelado?**

___ Ninguno

___ Poco

___ Mucho

2. **¿Sabes alguna utilidad del metamodelado en la ingeniería de software?**

___ Sí

___ No

Si su respuesta es Sí, ¿Cuál o cuáles?

_____

_____.

3. **¿Conoces alguna herramienta de metamodelado?**

___ Sí

___ No

4. **¿Conoces algún lenguaje de metamodelado?**

___ Sí

___ No

5. **¿Has trabajado con alguna de las herramientas de metamodelado?**

___ Sí

___ No

6. **¿Conoces alguna herramienta CASE que se utilice para el metamodelado?**

___ Sí

___ No

¿Si su respuesta es Sí, cuál o cuáles?

_____

_____.

Glosario de Términos

API: Una API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ATL: Lenguaje de definición de transformaciones, consultas y librerías de funciones.

CWM: Common Warehouse Metamodel o Metamodelo común de depósito (CWM). Proporciona un mecanismo para la estandarización de los modelos de datos habituales en un determinado dominio a través de bases de datos y repositorios de modelos.

DSDM: Desarrollo de Software Dirigido por Modelo.

DSL: Son lenguajes de alto nivel, diseñados para ser eficientes en tareas particulares. Una de las técnicas más habituales para describir la sintaxis de los DSLs es mediante metamodelado.

DTD (Document Type Definition): La DTD es una definición, en un documento SGML o XML, que especifica restricciones en la estructura y sintaxis del mismo. La DTD se puede incluir dentro del archivo del documento, pero normalmente se almacena en un fichero ASCII de texto separado. La sintaxis de las DTD para SGML y XML es similar pero no idéntica.

Ecore: Ecore es un lenguaje usado para la definición de metamodelos. El lenguaje Ecore es el usado por el Eclipse Modeling Framework (EMF).

Framework: Véase como marco de trabajo.

IBM: International Business Machines o IBM (NYSE: IBM) (conocida coloquialmente como el Gigante Azul) es una empresa que fabrica y comercializa herramientas, programas y servicios relacionados con la informática.

Glosario de Términos

JMI: estándar para la administración de metadatos.

MDA: Arquitectura de Software Dirigida por Modelos, es propuesta por OMG como un nuevo paradigma de desarrollo de software.

Metadatos: Significa datos sobre datos. Los metadatos son datos asociados a objetos o sistemas de información para fines de descripción administración, uso, preservación, entre otras. Existen varios tipos de metadatos descriptivos (que sirve para identificación y localización), administrativos (creación, derechos, control de acceso, entre otros) y los estructurales (que relacionan los objetos).

MOF: *Meta Object Facility o Facilidad de Meta-Objeto*) es el lenguaje de metamodelado propuesto por el OMG. Es utilizado para crear metamodelos.

OCL: Object Constraint Language (OCL) es un lenguaje de especificación que permite definir modelos más precisos y completos.

OMG: Es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos.

Plug-ins: es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa. Ésta aplicación adicional es ejecutada por la aplicación principal. Los plug-ins típicos tienen la función de reproducir determinados formatos de gráficos, reproducir datos multimedia, codificar/decodificar emails, filtrar imágenes de programas gráficos.

QVT: es un lenguaje para expresar transformaciones.

SQL: Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar -de una forma

sencilla- información de interés de una base de datos, así como también hacer cambios sobre la misma. Es un lenguaje de cuarta generación.

UML: Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software.

Unisys: Es una organización mundial de servicios y soluciones en tecnología de información y procesos de negocios.

XMI: Es un formato de intercambio de metadatos en XML.

XML: Es un lenguaje de marcado, se trata de una tecnología que permite la compatibilidad entre sistemas para compartir, información de manera segura, fiable y fácil.