Capítulo

4

Desenvolvimento de Software Dirigido a Modelos

Almir Buarque[[1]](#footnote-1)

O objetivo geral deste capítulo é apresentar o processo desenvolvimento de software dirigido a modelos (MDD[[2]](#footnote-2)), padronizado pela Arquitetura Dirigida a Modelos (MDA[[3]](#footnote-3)) do grupo OMG, sua relevância para elevação da qualidade do processo de engenharia de software e, consequentemente, do produto. Duas abordagens MDD serão descritas: OO-Método e AndroMDA. O capítulo mencionará ainda os problemas e desafios atuais do processo de desenvolvimento dirigido a modelos. Será apresentada mais detalhadamente, a abordagem OO-Método por ser uma referência na literatura MDD, ter precisão e definição semântica baseada na linguagem formal orientada a objeto chamada OASIS e por ser totalmente suportada pela ferramenta OlivaNova da Care Technologies [OlivaNova 2009].

#  Introdução

Dentro do contexto de que modelar é uma atividade essencial da engenharia de software, desenvolvimento de software dirigido a modelos (**Model Driven Software Development**), cujo acrônimo em inglês é MDD, vem representando atualmente um papel central no processo de engenharia de software. Convém lembrar que essa idéia não é nova. Desde a década de 1970, que os métodos formais difundiram o desenvolvimento de software a partir de modelos formais matemáticos que eram transformados até se gerar o código fonte final do sistema. A partir de um desenvolvimento formal é possível elevar a qualidade do software com técnicas formais de validação e verificação.

 Com o amadurecimento das linguagens de modelagem de software e a complexidade da conjuntura atual da indústria de software, cada vez mais, essa idéia tem se consolidado através de abordagens que adotam MDD como um padrão de desenvolvimento. Em 2001, ao especificar a Arquitetura Dirigida a Modelos-MDA (Model Driven Architecture), o grupo OMG, na verdade, definiu apenas uma nova instância de processo de desenvolvimento de software dirigido a modelos (MDD) que já existia há anos.

Os principais argumentos para a utilização de um processo de desenvolvimento dirigido a modelos são os seguintes: maior produtividade, portabilidade, interoperabilidade, menor custo, mais facilidade na evolução do software, enfim, maior qualidade do produto. Esses benefícios são evidenciados, por exemplo, num estudo [MDA 2003] que comparou uma produção de software usando-se a tecnologia MDD com o mesmo software desenvolvido com tecnologia OO tradicional. Isso ocorre principalmente pelas seguintes razões:

* A principal idéia em MDD é a transformação de modelos de maiores níveis de abstração (domínio do problema) em modelos mais concretos (domínio solução) até se obter, por fim, o código do sistema.
* O paradigma MDD preconiza que o desenvolvimento inicial e modificações futuras da aplicação sejam efetuados apenas no modelo mais abstrato.

Em processos MDD automatizados, esse modelo abstrato do sistema deve representar com precisão o código, ou seja, ele deve ser executável e ter uma equivalência funcional com todos os outros modelos mais concretos. Dessa forma, as modificações no modelo de mais alto nível de abstração são refletidas automaticamente nos modelos de mais baixo nível, tornando a atividade de modelar no nível mais abstrato o centro de todo processo de desenvolvimento do software e dispensando completamente, nos melhores ambientes MDD, a execução de atividades manuais nos modelos de mais baixos níveis de abstração (projeto e implementação).

Entretanto, a indústria de software e empresas de ferramentas CASE[[4]](#footnote-4) têm exagerado nesses benefícios, transmitindo a falsa idéia aos desenvolvedores de que, em MDD, apenas com um *click* ou “passo de mágica”, obtém-se todas as transformações de modelos até se chegar ao produto final de software. Além disso, passa-se a idéia de que gerar código é o principal objetivo MDD quando, de fato, é transformar modelos, confundindo os desenvolvedores com várias ferramentas (ambientes) CASE que apenas geram códigos fontes a partir de técnicas diversas e que, na verdade, não transformam modelos conforme a arquitetura MDA. Ademais, existem ainda problemas semânticos, complexidades e imprecisões (ambiguidades) inerentes aos modelos atuais que tornam esse processo de transformação e mapeamentos de modelos uma tarefa árdua e propensa a falhas.

Por outro lado, não há um consenso na comunidade acadêmica sobre qual modelo de maior nível de abstração é mais adequado (necessário e suficiente) para se modelar um sistema, dificultando-se padronizações, interoperabilidade e produzindo-se ambientes MDD que não são integrados com modelos em nível de requisitos, os quais são essenciais para todo o processo de Engenharia de Software. Este capítulo abordará todos esses tópicos referentes ao processo de desenvolvimento de software dirigido a modelos, mas precisamente sobre a arquitetura MDA.

#  Arquitetura Dirigida a Modelos

Nesta seção, serão apresentados os seguintes tópicos: uma visão geral dos padrões OMG[[5]](#footnote-5), arquitetura MDA e seus conceitos básicos, com o objetivo de explorar a teoria básica sobre o processo de desenvolvimento de software orientado a modelos. Será dado um foco maior sobre a transformação de modelos a partir de metamodelos, pois essa técnica facilita a automação do processo.

##  Conceitos Básicos

Para um melhor entendimento da arquitetura dirigida a modelos, o padrão MDA do OMG [OMG 2003] define os seguintes conceitos:

* **Modelo**

Um modelo de um sistema é a sua representação (especificação) funcional, estrutural e comportamental. Uma especificação é dita como formal quando é baseada em uma linguagem que tem uma sintaxe e semântica bem definida e, possivelmente, regras de análise, inferência ou prova de seus elementos. Essa sintaxe pode ser gráfica (visual) ou textual. A semântica pode ter um formalismo maior ou menor (lógica formal).

* **Dirigido a Modelos**

MDA é uma abordagem de desenvolvimento de sistema que usa o poder dos modelos. É dirigida a modelos porque provê meios de usar modelos para direcionar o curso de entendimento, projeto, construção, distribuição, operação, manutenção e modificação.

* **Arquitetura**

Arquitetura de um sistema é a especificação de suas partes e conectores, além das regras de interação dessas partes usando os conectores.

* **Ponto de vista**

Um ponto de vista (*viewpoint*) de um sistema é uma técnica de abstração, usando um conjunto selecionado de conceitos arquiteturais e regras de estruturação que visa focar ou representar um aspecto (característica) dentro desse sistema. O termo abstração é usado para significar o processo de suprimir (esconder) um detalhe selecionado para estabelecer um modelo simplificado.

* **Plataforma**

Uma plataforma é um conjunto de subsistemas e tecnologias que provê um conjunto coerente de funcionalidade através de interfaces e padrões de uso especificados, que qualquer aplicação (sistema) suportada por essa plataforma pode usar, sem ter que saber os detalhes de como essa funcionalidade provida pela plataforma é implementada.

* **Pontos de Vistas (Modelos) MDA**

Visando separar os diferentes níveis de abstração de um sistema de software e promover as facilidades e benefícios preconizados pelo processo de desenvolvimento dirigido a modelos (produtividade, interoperabilidade, portabilidade, etc), a arquitetura MDA especifica basicamente os três modelos a seguir:

* + **Modelo Independente de Computação (CIM[[6]](#footnote-6))**

O modelo independente de computação ou CIM (Computation Independent Model) é uma visão do sistema a partir de um ponto de vista independente de computação. O CIM não mostra detalhes da estrutura dos sistemas, sendo usualmente chamado de modelo de domínio ou modelo de negócio e utiliza, em sua especificação, um vocabulário familiar aos usuários do domínio (problema) em questão. Os usuários do CIM geralmente não têm conhecimento sobre modelos ou artefatos usados para realizar as funcionalidades definidas através dos requisitos. Esse modelo foca no ambiente do sistema e nos seus requisitos, deixando os detalhes da estrutura e processamento (computação) do sistema escondidos aos usuários ou, até mesmo, indeterminados.

Dessa forma o CIM tem um papel importante de fazer a ponte (reduzir a lacuna) entre aqueles que são especialistas no domínio do problema e seus requisitos, e aqueles que são especialistas em projeto (arquitetura) e construção dos artefatos que juntos vão satisfazer aos requisitos do domínio, elicitados pelos usuários.

O CIM é obtido no processo de documentação e especificação dos requisitos, ou seja, ao se especificar um modelo de requisitos para o sistema. Outra forma também de definição do CIM é através do modelo de negócios do sistema.

* + **Modelo Independente de Plataforma (PIM)[[7]](#footnote-7)**

O modelo independente de plataforma ou PIM (Platform Independent Model) foca na operação do sistema (modelo computacional), escondendo os detalhes necessários para implantar esse modelo numa plataforma específica. O PIM é único para o sistema e não muda quando se varia de uma plataforma para outra, permitindo assim portabilidade. Esse ponto de vista independente de plataforma pode ser especificado usando-se uma linguagem de modelagem de propósito geral (UML) ou uma linguagem específica (OO-Method) como será visto na seção 4.3.1. O PIM é um modelo conceitual do sistema.

* + **Modelo Específico de Plataforma (PSM[[8]](#footnote-8))**

O modelo específico de plataforma ou PSM (Platform Specific Model) é uma visão do sistema que agrega características e elementos constituintes de uma plataforma específica, contendo informações da tecnologia utilizada na aplicação, bem como a linguagem de programação, os componentes de *middleware*, e a arquitetura de hardware e de software. Para que isso seja possível, é necessário o suporte de ferramentas que façam o mapeamento adequado de uma especificação abstrata (PIM) para uma determinada plataforma. O PSM, por sua vez, passa por processo(s) de refinamento(s) para obtenção do nível de especificação desejado. A obtenção desse nível torna possível a transformação do mesmo no código (implementação) da aplicação. O modelo PSM é o responsável por lidar com toda heterogeneidade e complexidade dos diversos tipos de plataformas existentes.

* **Transformações (Mapeamentos)**

A força motriz do padrão MDA é a transformação de modelos, que pode ser realizada entre modelos de um mesmo ponto de vista ou entre modelos de pontos de vistas diferentes, tanto num sentido direto (de maior nível de abstração para menor nível de abstração) quando inverso. Em qualquer caso, sempre um modelo é usado como parâmetro de entrada para ser transformado em outro modelo.

A Figura 4.1 mostra o ciclo (sentido) mais natural do MDA, partindo do CIM (modelo de requisitos) até o nível mais baixo de código (implementação).



Figura 4.1. Transformações em MDA [Adaptada de Hitachi 2002]

Entretanto, são possíveis também as seguintes transformações (mapeamentos):

* + PSM => PIM (Engenharia Reversa)
	+ PIM => PIM, PSM => PSM (Modelos de mesmo nível)
	+ Implementação => PSM (Engenharia Reversa)
	+ PIM => Implementação
* **Transformações e Mapeamentos em MDA**

Existe uma quantidade enorme de ferramentas para suportar transformação de modelos. Transformações podem utilizar diferentes técnicas que vão desde uma transformação manual até as semi-automáticas e as automatizadas. Por exemplo, transformações de PIM para PSM podem ser realizadas através do uso de UML *Profiles* (extensões UML), uso de padrões (*patterns*), marcas (markings), metamodelos (metamodels) e transformações automáticas (via algoritmos) [MDA Guide Version 2003]. Os elementos centrais dessas transformações são os mapeamentos dos modelos. Segundo a arquitetura MDA, um mapeamento é um conjunto de regras e técnicas utilizadas para modificar, refinar ou transformar um modelo e obter outro modelo. Esse mapeamento, usando-se metamodelos (modelos que descrevem e especificam os modelos originais), facilita a automação. A Figura 4.2 descreve o Metamodelo MDA [MDA 2001]. Basicamente os modelos PIM e PSM são descritos através de seus metamodelos expressos preferencialmente com as tecnologias núcleo do OMG: UML, MOF ou CWM. Depois, a partir das várias regras de transformações entre os elementos desses metamodelos, técnicas de mapeamento são utilizadas para implementar a transformação.



Figura 4.2 Metamodelo MDA [Adaptada de MDA 2001]

Nota-se ainda que o OMG não contempla, nesse metamodelo MDA, o Modelo Independente de Computação (CIM) o que, sob a ótica da Engenharia de Software é como se deixasse uma lacuna a ser preenchida ou, como se o próprio PIM (UML) se unificasse com o CIM (UML), transmitindo a idéia de ser um único modelo capaz de representar de modo completo e consistente todo um sistema.

🡪Para ilustrar o esquema geral de transformações através de metamodelos, considere a Figura 4.3 onde um modelo 1 é transformado num modelo 2, usando como entrada do processo o metamodelo A do modelo 1 e produzindo o modelo 2 expresso no metamodelo B. É importante destacar que para realizar essa transformação é necessário ter regras de mapeamento precisas entre esses metamodelos.



Figura 4.3. Transformações de mapeamentos por metamodelos

 [Adaptada de MDA Guide Version 2003]

De fato, a Linguagem de Modelagem Unificada (UML) é um marco na história de modelagem visual de software, pois antes dela havia várias notações até mesmo incompatíveis entre si. Desde a sua primeira versão (UML 1.0) lançada em 1997, a linguagem de modelagem unificada recebeu diversas críticas e propostas de extensão. Em 2001, o OMG publicou a UML 2.0.

Alguns dos novos aperfeiçoamentos da UML 2.0 foram:

* Melhor suporte de extensão para outros modelos(linguagens) através do uso de UML Profiles;
* Aperfeiçoamento da expressividade de modelar, incluindo modelagem de processos de negócios, suporte a modelagem de classificadores reusáveis e suporte para modelagem de arquiteturas distribuídas e sistemas heterogêneos;
* Integração com Semântica de Ações (Actions Semantics) que o desenvolvedor pode usar para definir a semântica de tempo de execução do modelo (aspecto funcional) e prover precisão semântica exigida para analisar modelos e transformá-los em implementações.

Robert B. France em “Desenvolvimento orientado a Modelos usando UML 2.0: Promessas e Armadilhas” (Model-Driven Development Using UML 2.0: Promises and Pitfalls) [France and Ghosh 2006] cita que padrão UML 2.0 contém um largo conjunto de conceitos de modelagem que são relacionados de um modo complexo em termos estruturais. Para cobrir essa complexidade seus projetistas organizaram o padrão UML 2.0 em quatro partes:

* Infraestrutura: elementos ou construtores básicos da linguagem.
* Super-estrutura: o próprio metamodelo UML.
* Linguagem de Restrição de Objeto (OCL): especificação de consultas, invariantes (de estado), pré-condições, pós-condições, restrições e operações em modelos UML.
* Intercâmbio de Diagramas: extensão do metamodelo (Super-estrutura) para dar suporte armazenamento e intercâmbio de informação de modelos UML.

Além de toda essa complexidade, UML carece de precisão semântica, pois muitos dos seus elementos (primitivas) têm diferentes interpretações que variam conforme entendimento do projetista, causando muitas ambigüidades [France and Ghosh 2006]. Oscar Pastor [Pastor and Molina 2007] afirma que a maioria das ferramentas CASE baseadas em UML não realizam o processo de transformação de modelos de modo completo e correto. Isso porque UML tem conceitos tão ambíguos como generalização, associação, agregação e composição; e dependentes da interpretação subjetiva do projetista que o resultado em termos do produto de software é imprevisível. Assim, por exemplo, dois projetistas podem especificar que um relacionamento entre classes em UML seja uma associação, agregação ou herança e, para cada um tipo de relacionamento desses, pode-se ter implementações do sistema completamente diferentes. A causa disso é porque os conceitos de relacionamentos de classes em UML não são precisos e claramente definidos, dando margem a múltiplas interpretações.

Essa imprecisão, aliada da ausência de formalismo do metamodelo UML, faz com que sua validação fique comprometida, e como consequência, erros e inconsistências sejam propagados, durante o refinamento desses elementos, para os níveis de menor abstração da UML [Pastor and Molina 2007]. Para dar um melhor suporte MDD, UML 2.0 lançou o conceito de UML Profile. Esse mecanismo de extensão auxilia a transformação de modelos PIM para PSM específicos, conforme esquema ilustrado na Figura 4.4:

**Geração de código**

**Mapeamento**

**Requisitos**

Análise

Projeto de Sistema

Modelo

PIM

PSM

Código de Implementação

Do Sistema

**Perfil UML XX**

**---------------------------------**

**Padrões do modelo**

**Regras de mapeamento**

**Perfil UML YY**

**---------------------------------**

**Padrões do modelo**

**Regras de mapeamento**

**Perfis UML Dependente de**

**Plataforma**

**Seleciona Perfis**

**Regras de Geração de Código, etc**

**Frameworks**

**Ferramentas**

Figura 4.4. Transformações com UML Profile [Adaptada de Hitachi 2002]

Atualmente muitas extensões já estão padronizadas pela OMG, algumas estão em processo de padronização e outras ainda em discussão como mostrado na Figura 4.5.



Figura 4.5. Perfis UML da OMG [Adaptada de Hitachi 2002]

Enfim, devido à sua imprecisão semântica e complexidade, UML 2.0 torna-se um problema não somente para desenvolvedores de ferramentas MDD, mas também para os próprios projetistas (grupos de trabalho) da OMG na evolução do padrão UML [France and Ghosh 2006]. Isso não significa subestimar o valor inegável da UML no contexto da Engenharia de Software, entretanto, afirmar que UML vai ser mesmo o futuro do desenvolvimento de software dirigido por modelos (MDD) só o tempo dirá.

##  Padrões OMG e a Arquitetura MDA

O surgimento da arquitetura MDA em 2001 foi resultado da necessidade cada vez mais emergente de realizar manutenções em aplicações, integrá-las com outros sistemas, mudar suas infraestruturas, alterar seus requisitos e lidar com a frequente evolução e criação de novas tecnologias. MDA visa também proporcionar os seguintes benefícios: aumentar a produtividade no desenvolvimento de sistemas, melhorar a portabilidade e interoperabilidade .

Para atingir esses objetivos e separar os níveis de abstrações, MDA [OMG 2003] foi definida pela OMG em três camadas conforme Figura 4.6, tendo, na primeira camada de especificação (núcleo da arquitetura), padrões que ditam um conjunto de regras para estruturação da especificação expressa nos modelos e que não abordam características de plataformas específicas. Esta camada representa o modelo PIM. Os padrões também constituem definições propostas pela OMG. São eles:

* Linguagem de Modelagem Unificada (UML): padrão que define uma linguagem de modelagem geral orientada a objetos para especificação, construção e documentação de artefatos de sistemas complexos de software;
* Metamodelo Comum de Armazenamento (Common Warehouse Metamodel - CWM): padrão para armazenamento de dados que permite fácil manipulação dos mesmos entre ferramentas e plataformas de armazenamento em ambientes heterogêneos distribuídos;
* Serviço/Facilidade de Meta-Objeto (Meta Object Facility - MOF): padrão que define uma linguagem abstrata para definição de linguagens de modelagem (metamodelos). Ela é utilizada para descrever modelos da UML, CWM e do próprio MOF, além de definir o formato de intercâmbio para modelos, base do padrão XMI (XML Metadata Interchange);

Na segunda camada, encontram-se os modelos PSM que possuem características próprias a determinadas tecnologias e plataformas. Entre elas, algumas seguem padronização da OMG, elevando a resolução dos problemas de integração através da definição de especificações voltadas para interoperabilidade, que sejam abertas e independentes de fornecedores ou fabricantes específicos. São elas:

* Intercâmbio de Metadados em XML(XML Metadata Interchange- XMI): padrão para o intercâmbio de modelos através do mapeamento da linguagem definida pelo padrão MOF para o padrão XML do World Wide Web Consortium (W3C);
* Arquitetura CORBA: arquitetura que padroniza e simplifica a troca de dados entre sistemas distribuídos.

Na camada PSM, pode-se ter também outros padrões de arquitetura como JAVA EJB/JEE, Microsoft.NET, etc.

Na camada mais externa, segundo Figura 4.6, são exibidos os serviços que a maioria dos domínios de aplicações necessita, para então, serem apresentados os múltiplos domínios que fazem uso desses serviços. Esses serviços podem ser de segurança, persistência, controle de transações, tratamentos de eventos, etc.



Figura 4.6. Padrões MDA [Adaptada de OMG 2003]

#  Abordagens MDD

Esta seção tem como principal objetivo descrever a abordagem MDD, chamada OO-Método, que apresenta características de um real ambiente MDD através de uma completa transformação de modelos. O OO-Método inova com o conceito de compilador de modelos (model compiler), que de fato, é uma máquina virtual de transformação de modelos. Além disso, o modelo de alto nível, chamado modelo conceitual, do OO-Método tem todos seus elementos (primitivas) descritos numa notação visual (gráfica) e que são especificados numa linguagem formal orientada a objeto (OASIS). Essas características fazem com que o OO-Método tenha precisão sintática e semântica suficientes para prover um ambiente capaz, inclusive, de fazer validação de modelos e consequentemente gerar um produto de software final de qualidade.

 Ainda nesta seção, será descrita uma outra abordagem MDD Software Livre chamada AndroMDA que está se tornando bastante popular.

##  OO-Método

 A primeira versão do OO-Método foi introduzida em 1992 através da tese de PhD de Oscar Pastor juntamente com a da linguagem formal de especificação de sistemas de informação – OASIS [Pastor and Molina 2007]. Deste então, o método incorporou vários componentes até chegar à versão apresentada neste trabalho. Segundo autor [Pastor and Molina 2007], o método cobre todas as fases do processo de desenvolvimento de software, das fases iniciais de obtenção de requisitos e representação, passando pelo desenvolvimento correspondente do esquema conceitual OO, mais a geração do produto final de software numa plataforma específica. O centro do desenvolvimento do software dirigido por modelos do OO-Método é o Esquema (Modelo) Conceitual fundamentado a partir da seguinte afirmação do Prof. Antoni Olivé [Pastor and Molina 2007]:

**“Para desenvolver um sistema de informação é necessário e suficiente definir seu esquema conceitual**”

Esta idéia também aparece em trabalhos e propostas de alguns pesquisadores de prestígio. Toni Morgan defende a idéia de usar Programação não Estrema (Extreme Non-Programing) [Morgan 2002] em oposição à metodologia ágil XP, argumentando que a principal atividade no desenvolvimento de software é modelagem, e não programação, pois modelagem está no espaço do problema enquanto programar está no espaço da solução. O objetivo final é tornar verdadeira a sentença “O Modelo é o Código“, em vez de “O Código é o Modelo“. Tudo isso é possível se obter, quando se tem um Modelo Conceitual Executável que abstrai de modo completo e consistente todos os aspectos estáticos, dinâmicos e de interação (interface usuário) de um sistema, tal como o do OO-Método, passível de transformação através de um compilador de Esquema Conceitual.

**O processo básico de transformação**

OO-Método estabelece uma distinção clara entre o espaço do problema, onde está definido o esquema conceitual; e o espaço da solução, onde é obtido o produto de software representado pelo esquema conceitual. Na figura 4.7, o processo se inicia com uma a entrada que representa os requisitos do sistema, não importando por quais processos de engenharia de requisitos eles foram obtidos, nem o modelo de requisitos utilizado. De forma que esses requisitos são insumos para se criar (projetar) o esquema conceitual. Especificados os quatro modelos que compõe o esquema conceitual: modelo objeto, funcional, dinâmico e de apresentação, é gerado um repositório para conter todos os elementos (primitivas) especificados nos modelos desse esquema conceitual, utilizando-se a linguagem formal orientada objeto OASIS, conforme figura 4.7. Regras de mapeamentos das primitivas desse esquema conceitual para um modelo de aplicação específico de cada plataforma são definidas e, por fim, é realizada uma transformação automática desse modelo de aplicação para o código da aplicação correspondente à plataforma (modelo de aplicação) escolhida. O processo garante que há uma equivalência funcional entre toda primitiva definida no esquema conceitual com sua(s) respectiva(s) primitiva(s) no modelo de aplicação. No exemplo da figura 4.7, foi escolhido um modelo de aplicação (plataforma) constituído por uma arquitetura de três camadas: lógica da aplicação, persistência e interface com usuário.



Figura 4.7. Abordagem OO-Método [Adaptada de Pastor and Molina 2007]

**Comparação com MDA**

Os modelos do OO-Método, seus mapeamentos e transformações podem ser comparados com o padrão MDA, como ilustrado através da tabela 4.1.

Tabela 4.1. Comparação do OO-Método com MDA [Adaptada de Pastor and Molina 2007]

|  |  |
| --- | --- |
| **MDA** | **OO-Método** |
| Modelo Independente de Plataforma (PIM) | Modelo Conceitual |
| Modelo Específico de Plataforma (PSM) | Modelo de Aplicação |
| Modelo de Implementação (IM) | Código de Aplicação |
| Transformação PIM para PSM | Mapeamentos |
| Transformação PSM para IM | Transformações |

Entretanto, algumas propriedades do OO-Método estão ausentes no padrão MDA (OMG), tais como:

* O processo de compilação de modelos que, de fato, compila um modelo fonte (entrada) em outro modelo destino (saída) a partir de regras de mapeamentos precisas e implementado através de uma máquina virtual de compilação de modelos onde se realiza uma verdadeira transformação desses modelos.
* Em termos de PIM, OO-Método provê uma solução semanticamente precisa, porque está especificado usando uma linguagem formal OASIS que é computacionalmente completa. Em outras palavras, os modelos do esquema conceitual do OO-Método são também computacionalmente completos e contém toda a informação que é necessária para gerar automaticamente código-fonte.

**O Modelo Conceitual**

O modelo ou esquema conceitual é composto por quatro visões ou modelos que representam os requisitos funcionais de uma aplicação. Esses modelos são: modelo objeto, modelo dinâmico, modelo funcional e modelo de apresentação.

O Modelo Conceitual do OO-Método tem uma notação visual (gráfica) parecida com UML, mas que só usa apenas parte dos seus conceitos (diagramas) que julga necessário e suficiente para representar um sistema de informação. Além disso, a grande diferença, quando comparado com UML, é que o modelo conceitual do OO-Método tem uma semântica e sintaxe bem precisa e, como base, a linguagem formal OASIS. Isso propicia a validação automática do modelo conceitual a fim de não deixar passar falhas (erros) para os modelos posteriores de mais baixos níveis.

A seguir serão apresentadas, de modo geral, as principais características desses modelos. Para detalhes mais específicos, consultar referência bibliográfica [Pastor and Molina 2007].

* **Modelo Objeto**

O modelo objeto especifica as propriedades estáticas do sistema, definido pelo diagrama de configuração de Classe que é composto por:

* Classes
	+ Atributos
	+ Precondições e Serviços
	+ Restrições de Integridade
* Relacionamento entre as Classes
	+ Associação, Agregação e Composição
	+ Herança
* Agentes

OO-Método também representa precisamente os tipos relacionamentos como associação, agregação e composição. A associação considera aspectos de cardinalidade, papéis e também de temporalidade. A temporalidade de uma associação se define como estática ou dinâmica, conforme a associação seja constante (estática), desde o momento em que é criada até o fim do seu tempo de vida.

Além da precisão que o OO-Método trata os conceitos de associação, agregação e composição, ele também considera quais são os impactos que eventos de inserção, deleção e mudança de objetos têm sobre esses relacionamentos entre as classes.

O OO-Método suporta também os conceitos de Herança (generalização e especialização) e herança múltipla. O modelo conceitual lida com gerenciamento de complexidade do modelo através da definição de subsistema que utiliza uma notação visual igual a de uma package em UML.

* **Modelo Dinâmico**

Este modelo representa o comportamento do sistema, especificando suas propriedades dinâmicas através de dois diagramas:

* Diagrama de Transição de Estado: especifica o ciclo de vida válido dos objetos de uma classe e seus serviços disponíveis em cada estado.
* Diagrama de Interação de Objeto: especifica as interações válidas entre os objetos através das transações, operações e gatilhos.

Um gatilho é uma condição sobre um estado do objeto que, tornando-se verdadeira, faz com que este objeto dispare eventos ou transações sobre si mesmo (self) ou sobre outros objetos (object) do sistema. A sintaxe de gatilhos na linguagem formal OASIS é: **<destination>::<condition>:<service>**

Onde, **<destination> := self | object | class | for al**l

Sendo que “self” significa para a si mesmo, “object” para uma instância de outra classe, “class” para todas as instâncias da classe e “for all’ para um subconjunto de objetos de uma classe.

* **Modelo Funcional**

O modelo funcional especifica o relacionamento estático e dinâmico através de:

* Definição semântica relacionada às transições de estado
* Descrição de como a execução dos eventos muda o valor dos atributos das classes

O modelo funcional trata também questões de eventos de criação e destruição de objetos, além de transações e operações que afetam os estados (valores dos atributos) dos objetos. O modelo funcional do OO-Método, combinado com sua linguagem de fórmula, provê de modo completo e preciso uma solução para especificar os aspectos funcionais de um sistema via modelo conceitual.

O recurso de Semântica de Ações (Action Semantics) da UML 2.0, criado para suprir essa necessidade de modelagem de aspectos funcionais, não possui uma semântica definida claramente e precisamente. O modelo funcional do OO-Método e sua linguagem de fórmula, proveem solução adequada, permitindo:

* Acesso a dados de acordo com o Modelo Objeto
* Definição de lógica seqüencial
* Manipulação de classes e objetos
* Manipulação de relacionamentos
* Uso de operadores lógicos, aritméticos e relacionais
* **Modelo de Apresentação**

O modelo da apresentação especifica os requisitos de Interface de Usuário, modelando uma interface abstrata que é independente de plataforma ou dispositivo. Esse modelo de apresentação em nível de análise (conceitual) é considerado uma inovação do OO-Método, em relação outras abordagens que, na maioria, descrevem interface-usuário apenas em nível de implementação. No OO-Método, o modelo de apresentação é organizado em três níveis:

* **Nível 3**: **Elementos Básicos**

Nível mais baixo, constituído pelos elementos básicos de entrada de dados, seleções, grupos de dados, filtros, critérios de classificação, conjunto de visualização, ações e navegação.

* **Nível 2**: **Unidades de Interação**

Nível intermediário com conceito fundamental do modelo de apresentação que descreve um particular cenário de interação entre o usuário e o sistema. Geralmente, a interface de usuário de um sistema é definida como uma coleção relevantes unidades de interação e pelo modo como essas unidades estão estruturadas. OO-Método provê quatro tipos de unidades de interação, descritas a seguir:

* Unidade de Interação de Serviço: define um cenário no qual o usuário interage com o sistema para executar um serviço, definindo propriedades como: argumentos, grupos de argumentos, valores padrão, ajuda e retorno (feedback) em caso de erro de execução do serviço. Por exemplo, Para a classe Veículo , pode ser definida uma unidade de interação de serviço chamada “Comprar\_Veículo’, contendo todos os argumentos necessários, valores padrão, informações de ajuda, exceções em caso de erro, etc.
* Unidade de Interação de Instância: define um cenário no qual apenas informações sobre um único objeto são mostradas, incluindo lista de serviços, bem como cenários de informações relacionadas a esse objeto que o usuário pode navegar. Um exemplo típico é quando o usuário quer gerenciar um veículo específico. Isso inclui exibição dos atributos do veículo, possibilidade de navegar por informações relacionadas como relatórios de manutenção e os serviços executados sobre o veículo (aluguel, enviar para manutenção, etc).
* Unidade de Interação de População: descreve um cenário de interação onde o usuário pode manipular uma coleção de objetos de uma mesma classe, podendo definir filtros, critérios de ordem, conjunto de exibição (display set), ações e navegação.
* Unidade de Interação Mestre/Detalhe: descreve um cenário para interação com múltiplas coleções de objetos pertencentes a classes diferentes, mas relacionadas. O mestre representa o cenário de interação principal e o detalhe, o secundário.
* **Nível 1**: **Árvore de Hierarquia de Ação**

Nível mais alto. Uma vez definidos os cenários de interação do nível 2 do modelo de apresentação, faz-se necessário determinar como essas unidades de interação serão estruturadas e apresentadas ao usuário. Essa estrutura caracteriza o nível mais alto da interface com o usuário, o que poderia ser descrito como o “Menu” principal da aplicação. A árvore de hierarquia de ação serve para esse propósito. Ela é estruturada hierarquicamente por uma árvore, tendo um nó raiz e respectivas ramificações até chegar às folhas. Por exemplo, um sistema de aluguel de carros (nó raiz) é organizado principalmente como tendo as seguintes ramificações: Veículos, Clientes, Aluguéis e Usuários.

A construção do modelo de apresentação pode ser realizada de modo “top-down”, ou seja, partindo-se do nível 1 até chegar aos elementos do nível 3; ou “bottom-up” , partindo-se do nível 3 até a definição do nível 1.

Além dessas quatro visões do modelo conceitual, o OO-Método dá suporte a interoperabilidade e interface do sistema modelado com outros sistemas externos existentes, chamados de sistemas legados, através do conceito de visão de legado.

Enfim, ao se definir os quadro modelos básicos (objeto, dinâmico, funcional e apresentação) do esquema conceitual do OO-Método, tem-se toda infraestrutura necessária e suficiente para representar um sistema de informação no contexto do espaço do problema.

**O Compilador de Modelos**

Como OO-Método é segue o processo ideal MDD de transformação de modelos, ele precisa de alguma forma transformar o modelo conceitual, que contém todas as propriedades estáticas, dinâmicas e de interação com usuário (apresentação), em um modelo de implementação (código). Essa transformação é automatizada através do Compilador de Esquema Conceitual que pode ser visto como uma máquina virtual de programação. OO-Método inova com essa idéia de compilador de modelos, tornando-o diferente de outras abordagens que apenas focam a geração de código a partir de modelos utilizando técnicas diversas como integração, sincronização de código, refatoração etc.

Em comparação com outras abordagens existentes, o OO-Método se destaca por tratar de modo completo e preciso todos os aspectos da compilação de modelo. Um exemplo típico de problemas com outras abordagens são as insuficientes transformações para se obter o produto final de software. Mais grave ainda, a maioria das ferramentas que geram código a partir de modelos, considera única e exclusivamente como seu modelo inicial de entrada o diagrama de classes em UML, negligenciando todos os demais diagramas que capturam os demais aspectos dinâmicos e de interação de uma aplicação.

**OLIVANOVA: Uma Ferramenta de apoio ao OO-Método**

O OO-Método é implementado através do produto OlivaNova da Care Technologies [OlivaNova 2009] . O principal objetivo de OlivaNova é separar o que deve ser feito (espaço do problema), de como deve ser feito (espaço da solução). Ela é composta de duas principais ferramentas: o modelador e a máquina de transformação. O modelador permite:

* Modelar objetos e negócios;
* Modelar dados;
* Modelar integração;
* Modelar sistemas legados;
* Modelar regras e limitações;
* Definir conceitualmente interfaces do usuário;
* Suporte a UML;
* Suporte a XML.

A máquina de transformação é que implementa todo o processo de compilação de modelos do OO-Método, gerando código fonte na plataforma de destino.

No sítio da OlivaNova [OlivaNova 2009] existe um quadro que a compara com várias outras ferramentas MDD comerciais, tais como Borland Together , Compuware OptimalJ, IBM Rational Software Architect, etc.

##  AndroMDA

AndroMDA [AndroMDA 2009] é uma poderosa ferramenta MDA Open Source. Possui arquiteturas como Spring, EJB, .Net, Hibernate, Struts e está desenvolvida sobre o Eclipse. Pode ser utilizada pelos servidores de aplicação Jboss e TomCat e suporta a UML2.0. Agora está em sua versão 4.0, disponível somente para “preview”e permite a criação e utilização de metamodelos no padrão EMF (Eclipse Model Framework) [Projetos Eclipse 2009] e, além disto, possibilita a definição de transformação de modelos PIM a modelos PSM para depois atingir a geração de código fonte, fazendo uso de transformações Modelo para Texto (Model to Text).

Como framework, gerencia qualquer tipo de modelo (geralmente modelos UML guardados no formato XMI) produzido por outras ferramentas case, que combinados aos plugins que possui, permite a geração de modelos e código fonte.

Em AndroMDA é possível gerar componentes para todas as linguagens: Java, .Net, HTML, PHP. Se desejarmos utilizar alguma tecnologia que ainda não esteja contemplada, somente temos que desenvolver plugins para isto (ou mudar algum que já exista). É mais comumente utilizado por programadores da tecnologia J2EE, inclusive

podendo gerar um projeto J2EE e seu código partindo de um modelo de classe. É possível definir gerar código para Hibernate, EJB, Spring, WebServices e Structs e o código gerado é automaticamente adicionado ao projeto e ao processo de compilação.

Sua realidade MDA permite fazer com que o trabalho de arquitetura e desenvolvimento seja mais curto e de mais qualidade, trabalhando com modelos independentes de plataforma que posteriormente serão refletidos em modelos UML (PSM). Isto permite, entre outras vantagens, ter foco no modelo e necessidades organizacionais (Modelo PIM) e a possibilidade de reutilizar o modelo PIM em outros projetos.

Como etapa seguinte se pode efetuar a transformação até o código fonte da aplicação, tendo como etapas intermediárias a geração de um ou mais modelos PSM. Neste ponto, é onde AndroMDA mais se destaca, por possuir muitos plugins (programas que adicionam funcionalidades específicas) já desenvolvidos e que realizam a transformação PIM > PSM em muitos tipos de linguagens, tecnologias e plataformas diferentes. Estes plugins são chamados cartuchos (cartridges) e utilizá-los são bastante fáceis.

Além das vantagens citadas anteriormente, destacamos como pontos positivos de AndroMDA: não desenvolvimento de código redundante, o código reflete exatamente o que definem os modelos e a possibilidade de alterar de "cartridge" para que gere o mesmo sistema em outras linguagens. Para se desenvolver novos cartuchos para qualquer plataforma específica deve-se basicamente identificar as regras de transformação e criar um perfil (profile) UML

 Entretanto, o nível mais alto de abstração de AndroMDA depende da ferramenta de modelagem que gera UML (PIM). Assim, o nível mais alto é o conceito de caso de uso. A ferramenta de melhor aceitação para modelar em UML e, fazer exportação do metamodelo UML em XMI é a Magicdraw.

 AndroMDA não provê recursos de definição de interface usuário abstrata tal como existe em OO-Método. AndroMDA usa conceito de sincronização de modelos (PIM E PSM ) e trata questões de rastreabilidade e validação de modelos de forma limitada.

A versão AndroMDA 4.0 que está sendo desenvolvida visa aperfeiçoar o processo de transformação de modelos e, principalmente, receber como entrada "input", no modelo inicial de mais alto nível, metamodelos de qualquer linguagem de domínio específico. Isso, tornará o ambiente MDD de AndroMDA capaz de importar qualquer outro modelo de sistema, e não apenas, UML.

#  Problemas e Desafios dos Processos MDD

##  Visão Geral

Pode-se considerar que desenvolvimento de software dirigido a modelos ainda não está num nível de maturidade suficiente para realizar o sonho acalentado de todo desenvolvedor: ter um produto final de software com qualidade e 100% gerado automaticamente a partir dos seus requisitos. Nos melhores ambientes, o desenvolvedor ainda consegue modelar em nível de análise e projeto, mas não a nível de requisitos. Com isso, o desenvolvedor, dessa primeira década do terceiro milênio, ainda realiza esforço manual considerável em nível de projeto e implementação. E isso, tem impactos negativos nos custos, prazos e qualidade dos softwares produzidos. O paradigma dirigido a modelos traz uma promessa de tornar realidade esse sonho. Entretanto, muitos desafios haverão de ser superados.

Sabe-se que processo MDD ainda está na sua infância. Nem as linguagens (modelos) nem as ferramentas se desenvolveram o suficiente para concretizar suas promessas feitas. O processo MDA, padronizado pela OMG, é apenas uma referência e pode ser usado por qualquer outro processo específico de desenvolvimento de software existente (RUP, XP, OPEN, Agentes, Aspectos, Formais, etc.) desde que se adapte e seja dado um foco especial em modelos e suas transformações. Decerto que processos como RUP e OPEN, por suas próprias características, são mais fáceis de serem adaptáveis a um processo dirigido a modelos do que o XP cujo foco predominante é implementação.

Este capítulo do livro procurou relatar alguns aspectos dessa tecnologia MDD. Como trabalhos futuros e desafios para o processo desenvolvimento de software dirigido por modelos, destacam-se os seguintes:

* Integração com a fase requisitos (Engenharia de Requisitos) para elevação do nível de abstração inicial a ser modelado (modelo CIM da MDA);
* Suporte a modelos orientados a metas (goals), agentes e aspectos.
* Melhor precisão semântica dos modelos em relação às características estáticas, dinâmicas e de apresentação (interação-usuário);
* Melhores mapeamentos entre os modelos;
* Melhor transformação automática de modelos (automação);
* Melhor suporte à Validação de Modelos;
* Melhor integração com as plataformas específicas (PSM);
* Melhor e maior percentagem de código fonte gerado;
* Maior suporte à rastreabilidade;
* Melhor suporte à engenharia reversa;
* Suporte à computação autonômica;
* Suporte a ontologias.

##  Lições Aprendidas na adoção de soluções MDA

Muitas organizações que, nos últimos anos, vêm utilizando com sucesso soluções MDA, perceberam que um conjunto de práticas e passos consistentes devem ser considerados ao se adotar um processo MDD automatizado. A seguir, é mostrado um resumo com os passos necessários para se desenvolver uma solução MDA adequada:

* Examinar os modelos atualmente usados na empresa no seu processo de desenvolvimento e a conexão/correlação semântica entre os elementos desses modelos.
* Identificar as transformações (fases do processo ou modelos) candidatos para automação.
* Especificar (documentar) os requisitos (regras, restrições) dessas transformações
* Criar os UML Profiles necessários
* Desenvolver o código da(s)s transformação(coes)
* Esboçar documentos de uso, empacotar e distribuir

##  O programa FastStart da OMG

Recentemente, o grupo OMG lançou o programa “FastStart” para ajudar as organizações aprenderem sobre MDA e aplicar MDA nas arquiteturas de seus sistemas, na integração dos sistemas e nos seus processos de desenvolvimento de software. Durante programa FastStart , a organização recebe consultores do grupo OMG para realizarem as seguintes atividades:

* Análise inicial MDA
* Revisão da Arquitetura Empresarial MDA
* Plano de Transição MDA
* Seminários Executivos MDA
* Prática MDA

Essas atividades duram em média cinco semanas e ajudam à equipe executiva e técnica da empresa a:

* Analisar claramente e planejar como MDA pode melhor ser introduzido e aplicado para melhor beneficiar a organização e seus processos de negócios chaves
* Capacitar a empresa em MDA para que ela própria, sem ajuda de provedores externos, desenvolva seu processo MDA/MDD.

#  Considerações finais

O processo desenvolvimento de software dirigido a modelos (MDD) tem recebido grande atenção e importância no meio acadêmico e empresarial, sendo considerado por muitos pesquisadores, como uma tendência irreversível no futuro da engenharia de software. MDD, conforme arquitetura MDA, foi concebido para alcançar os seguintes benefícios: produtividade, qualidade, interoperabilidade, portabilidade e redução de custos.

Apesar de outros processos de desenvolvimento de software também terem essas metas, nenhum deles consegue alcançá-las com mais efetividade do que o processo dirigido a modelos. Isso, porque em MDD está explícita a idéia de automatização, ou seja, de se ter uma sistemática de transformação de modelos até se obter o produto de software final. Essa automatização e foco em modelos geram impactos profundos no modo de como as organizações produzem, gerenciam e fazem manutenções em softwares.

Assim, para se obter os benefícios de um processo MDD, a empresa ou desenvolvedor deve, primeiramente, entender bem o processo e, depois, promover mudanças de atitude, cultura e tecnologia que, de fato, são o maior desafio para adoção de um processo MDD.

Este capítulo do livro apresentou o processo de Desenvolvimento de Software Dirigido por Modelos, focando seus aspectos conceituais e tecnológicos. Foi apresentada a arquitetura dirigida a modelos (MDA) do grupo OMG, padrões e o processo básico de transformação de modelos.

Foi reservada uma atenção especial ao OO-Método, uma vez que ele é considerado uma referência entre as abordagens que realmente seguem os padrões de desenvolvimento dirigido a modelos (MDD). Fez-se uma comparação do OO-Método com o padrão MDA. Foram apresentadas suas quatro visões básicas: modelo objeto, dinâmico, funcional e de apresentação. Mostrou-se OLIVANOVA, o produto que implementa o OO-Método.

Finalmente, foi analisada a ferramenta livre de desenvolvimento dirigido a modelos chamada AndroMDA.

# Tópicos de Pesquisa

O Processo de desenvolvimento de software dirigido por modelos (MDD) ainda é um tema recente. Os benefícios do padrão MDA ainda não foram bem entendidos pelas organizações. Existem várias linhas e tópicos de pesquisa relacionados com MDD/MDA, entre estes se destacam:

* Desenvolvimento de modelos precisos semanticamente e completos para facilitar transformações e validações: Linguagens para modelar sistemas (Modelos) necessitam de mais precisão semântica, mais formalidade e descrever, de modo mais completo possível, as propriedades estáticas e dinâmicas de uma aplicação. Não apenas UML, mais outras linguagens de modelagens, necessitam evoluírem para cobrir essas necessidades.
* Desenvolvimento ou aperfeiçoamento de ferramentas de apoio ao processo MDD: Um processo completo de desenvolvimento dirigido a modelos tem que dar suporte aos modelos que vão dos níveis de abstrações mais elevados (CIM, PIM) até níveis de plataforma (PSM) e implementação. Raras são as ferramentas que dão suporte com qualidade a todo o espectro de desenvolvimento de software. Sendo assim, muita pesquisa está sendo realizada no sentido de preencher essas lacunas.
* Adaptações dos processos específicos de desenvolvimento de software ao padrão dirigido a modelos: MDD não determina qual processo de desenvolvimento específico de software seja utilizado. Processos existentes tais como RUP, OPEN ou XP podem ser adaptados para incorporar automação e transformação de modelos conforme arquitetura MDA.
* Extensões MDA para várias plataformas: Um grande desafio atual é que MDA resolva problemas de interoperabilidade entre plataformas. A grande heterogeneidade de plataformas existentes aumenta bastante a complexidade de soluções MDA que têm de transformar modelos para praticamente várias plataformas (sistemas operacionais, arquitetura de hardware, linguagens de programação, *middlewares* diversos (JAVA/CORBA, COM+/.NET, Web Serviçes), etc.
* MDA em Linhas de Produtos de Software (LPS): A necessidade de produzir aplicativos distintos a partir de pontos comuns e variáveis, utilizando tecnologia LPS, tem MDD como um forte aliado na automatização do desenvolvimento de software considerando essas características chamadas de *Features.*
* Rastreabilidade em processos MDD: Saber a correlação entre os diversos elementos de um sistema, desde requisitos até implementação, é uma poderosa forma de gestão de mudanças que permite, entre outros benefícios, análise de impactos em manutenções de software. Como MDD preconiza mapeamento e transformações entre esses diversos elementos de um sistema, a rastreabilidade é extremamente facilitada, dispensando inclusive o uso de ferramentas específicas para esse fim. Entretanto, poucas ferramentas MDD fazem uso eficaz de rastreabilidade.
* Medição/Estimativas de projetos de software em ambientes MDD: Medir tamanho, estimar esforço, prazo e custo de software são temas que estão sendo reavaliados com a atual tendência de desenvolvimento dirigido a modelos. Isso porque um ambiente MDD automatizado reduz, tempo, esforço e custo de desenvolvimento dos projetos.
* MDD para sistemas embutidos (embedded systems development): Desenvolvimento de sistemas embarcados pela natureza e propósito específicos, necessitam otimizar e agilizar o processo de desenvolvimento de software. MDD está sendo utilizado como tecnologia para obter esses objetivos.

#  Sugestões de Leitura

Este capítulo propôs dar uma visão sobre Desenvolvimento de Software Dirigido por Modelos. Entretanto, devido ao enorme escopo e dinamismo dessa área, realizou-se um esforço considerável para contemplar, bem como resumir de modo claro e objetivo, um maior número de tópicos possíveis sobre o tema, ou seja, tentou-se elaborar um trabalho que fosse o mais científico, atualizado e completo possível.

Para complementar o material apresentado neste capítulo, o autor sugere que o leitor realize as seguintes leituras:

* UML executável (OMG): Usando apenas os diagramas de classes, de estado e a extensão UML “Action Semantics” torna um modelo UML capaz de ser executável e o transforma, através de compiladores de modelos específicos, em plataformas como C++, Java, etc.

Para mais detalhes sobre esse tópico, ver livro: **Executable UML, A Foundation for Model Driven Architecture**, Mellor-Balcer, Addison-Wesley, 2002 e o site http://en.wikipedia.org/wiki/Executable\_UML.

* Grupo de pesquisa “Precise UML- PUML”: Como visto nas seções deste capítulo 4.2.1[France and Ghosh 2006] e 4.3.1 sobre o OO-Método, UML carece de precisão semântica, por exemplo, dependendo da interpretação do projetista pode-se modelar uma um determinado objeto como agregação, composição, associação ou herança. Visando aperfeiçoar a precisão semântica e formalidade da linguagem de modelagem de propósito geral UML, há um grupo de trabalho desenvolvendo Precise UML-PUML em http://www.cs.york.ac.uk/puml/maindetails.html.
* Ambiente MDD “Moskitt” da Universidade Politécnica de Valencia: O Kit de Modelagem de Software (Modeling Software KIT-MOSKitt) é um ambiente case MDD de código aberto (livre) construído sobre o Eclipse IDE, desenvolvido pelo Ministério Regional de infra-estrutura e transportes de Valencia (Espanha). Para maiores informações ver http://www.moskitt.org/eng/moskitt0/.
* Ambiente MDD Open source “OPENMDX”: Ferramenta concorrente do AndroMDA, roda no Eclipse IDE, sendo considerado também o estado da arte em desenvolvimento dirigido a modelos. Boa documentação adicional, inclusive ferramenta disponível para download em http://www.openmdx.org/.
* Livros específicos sobre o tema:
	+ - **MDA Explained: The Model Drive Architecture:Pracice e Promise** by Anneke Kleppe, Jos Warmer, Wim Bast (Editora Addison Wesley 2003 ): Obs. Muito bom para uma introdução sobre MDA.
		- **Model-Driven Software Development** by Sami Beydeda, Matthias Book , Volker Gruhn (Editora Springer 2005 ): Obs. Excelente pelo nível teórico básico , avançado e técnico.
		- **Real-Life MDA: Solving Business Problems with Model Driven Architecture (Interactive Technologies)** (Editora Morgan Kaufmann, 2006 ): Obs. Totalmente prático, focado em soluções de processos de negócios, utilizando-se MDA/MDD.

#  Exercícios

Para sedimentar melhor os conhecimentos teóricos abordados nesse capítulo, os seguintes exercícios foram elaborados:

1. MDA usa modelos distintos para separar os sistemas/aplicações em os níveis de abstrações/visões distintos. Quais são modelos definidos no padrão MDA e descreva seus propósitos.
2. Descreva a técnica de transformação de modelos através de metamodelos.
3. Quais os padrões que estão no núcleo da arquitetura MDA do grupo OMG.
4. Discuta: MDA determina o uso ou recomenda algum processo específico de desenvolvimento de software, tal como RUP, XP ou outro qualquer? Como se poderia adaptar o RUP ou XP para utilizar um processo dirigido a Modelos (MDD/MDA)?
5. Caracterize o modelo conceitual do OO-Método.
6. O que é o Compilador de Modelos do OO-Método?
7. Compare as ferramentas CASE de desenvolvimento de software dirigido a modelos: OLIVANOVA e AndroMDA.
8. Suponha uma ferramenta MDD que dá suporte completo aos modelos CIM, PIM e PSM. Durante as manutenções dos aplicativos desenvolvidos com essa ferramenta qual é o único desses modelos que, segundo o padrão MDA, deve ser alterado?
9. Quais os benefícios em se adotar um processo de desenvolvimento de software dirigido a modelos?
10. Usando uma ferramenta de modelagem UML (MagicDraw, ArgoUML,etc) modele a seguinte aplicação simplificada para administração de alunos, utilizando o diagrama de classe da figura 4.8. As funcionalidades (operações) básicas da classe Aluno são inserir, excluir, listar e alterar, conforme diagrama de atividades da figura 4.9. Depois, instale uma das ferramentas free MDD (AndroMDA ou OPENMDX), configure-as adequadamente e importe/utilize o modelo UML. Por fim, tente usar ferramenta MDD para gerar seu aplicativo na plataforma JAVA JEE, inclusive com recursos de persistência num banco escolhido (Mysql, Postgresql, etc) e de distribuição (deployment) num servidor JSP Tomcat Apache. Talvez, você tenha tido dificuldade em instalar e configurar esses ambientes de desenvolvimento, mas que achou do processo de transformar seu modelo PIM em PSM automaticamente? Que achou de obter o código do aplicativo automaticamente? Tente fazer uma alteração (evolução) do aplicativo para adicionar disciplinas, professores e seus respectivos relacionamentos com alunos, lembrando-se de que a alteração deverá ser realizada apenas no modelo PIM (mais alto nível de abstração).



Figura 4.8 Diagrama de Classes



Figura 4.9 Diagrama de Atividades

#  Referências

MDA productivivity case study. The Middleware Company (2003). <http://www.omg.org/mda/presentations>. Acesso em julho 2009.

OMG: padrão MDA (2003).< http://www.omg.org/mda >. Acesso em julho 2009.

Pastor O.; Molina J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer Publisher 2007.

OlivaNova Care Technologies, Denia, Spain (<http://www.care-t.com> Acesso em julho 2009.

Kontonya, G. e Sommerville, I. (1998) Requirement Engineering: Processes and Techniques, John Wiley & Sons.

MDA Guide Version 1.0.1, Document Number: omg/2003-06-01 Date: 12th June 2003. <http://www.omg.org/mda/>. Acesso em julho 2009.

Hitachi M. O.; MDA and System Design. Presentation at MDA Information Day, OMG Technical Meeting, April 2002.

Model Driven Architecture (MDA). Document number ormsc/2001-07-01, Architecture Board ORMSC1, July 9, 2001. <http://www.omg.org/mda/presentations>. Acesso em julho 2009.

France R. B.; Ghosh S.; Dinh-Trong T. : Model-Driven Development Using UML 2.0: Promises and Pitfalls. In IEEE *Computer*, vol. 39, no. 2, pp. 59-66, Feb. 2006.

Morgan T (2002) Business rules and information systems – aligning IT with business goals.Addison-Wesley, Reading,MS.

Pastor, O.: Model-Driven Development: The OO-Method Aproach. Presentation at UFPE, Recife, Brasil August 2008.

AndroMDA. <http://www.andromda.org>. Acesso em julho 2009.

Projetos Eclipse <http://www.eclipse.org/projects/> Acesso em julho 2009.

1. almirbuarque@gmail.com [↑](#footnote-ref-1)
2. Do inglês Model Driven Software Development [↑](#footnote-ref-2)
3. Do inglês Model Driven Architecture [↑](#footnote-ref-3)
4. Engenharia de Software Auxiliada por Computador. Do inglês Computer Aided Software Engineering [↑](#footnote-ref-4)
5. Do inglês Object Management Group – site: www.omg.org [↑](#footnote-ref-5)
6. Do inglês Computation Independent Model [↑](#footnote-ref-6)
7. Do inglês Platform Independent Model [↑](#footnote-ref-7)
8. Do inglês Platform Specific Model [↑](#footnote-ref-8)