Capítulo 4

Desenvolvimento de Software Dirigido a Modelos

Almir Buarque[1]

O objetivo deste capítulo é apresentar o processo de desenvolvimento de software dirigido a modelos (MDD[2]), padronizado pela Arquitetura Dirigida a Modelos (MDA[3]) proposta pela OMG[4], sua relevância para a melhoria da qualidade do processo de engenharia de software e, consequentemente, do produto. Duas abordagens MDD serão descritas: OO-Method 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-Method por ser uma referência na literatura MDD, ter precisão e definição semântica baseada na linguagem formal orientada a objetos chamada OASIS, e por ser totalmente suportada pela ferramenta OlivaNova da Care Technologies [OlivaNova 2009].

4.1. Introdução

Dentro do contexto de que modelar é uma atividade essencial da engenharia de software, o MDD representa 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 MDA, o grupo OMG, na verdade, definiu apenas uma nova instância de processo de desenvolvimento de software dirigido a modelos que já existia há anos.

Os principais argumentos para a utilização de um processo de desenvolvimento de software dirigido a modelos são os seguintes: maior produtividade, portabilidade,

[1] almirbuarque@gmail.com
[2] Do inglês Model Driven Software Development
[3] Do inglês Model Driven Architecture
[4] Do inglês Object Management Group – site: www.omg.org

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 Orientada a Objetos (OO) tradicional. Isso ocorre principalmente pelas seguintes razões:

- A principal ideia em MDD é a transformação de modelos de maiores níveis de abstração (domínio do problema) em modelos mais concretos (domínio da 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, o 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[5] têm exagerado nesses benefícios, transmitindo a falsa ideia 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 ideia de que gerar código é o principal objetivo do MDD quando, de fato, o principal objetivo é transformar modelos Assim, muitos desenvolvedores têm sido iludidos com várias ferramentas/ambientes CASE que geram códigos fontes a partir de técnicas diversas e que, na verdade, não transformam modelos conforme o que prega 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.

4.2. Arquitetura Dirigida a Modelos

Nesta seção, será apresentada uma visão geral dos padrões OMG, 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, segundo a arquitetura MDA essa técnica facilita a automação do processo ao se utilizar tecnologias de

[5] Engenharia de Software Auxiliada por Computador. Do inglês Computer Aided Software Engineering

transformações modelo-modelo tais como ATL [Eclipse ATL documentation 2009] ou QVT [OMG: QVT specification 2009].

Excluído:

4.2.1. Conceitos Básicos

Para um melhor entendimento da arquitetura dirigida a modelos, o padrão MDA [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 (lógica formal) maior ou menor.

- **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 funcionalidades através de interfaces e padrões de uso especificados, que qualquer aplicação (sistema) suportada por essa plataforma pode usar, sem ter que conhecer os detalhes de como essa funcionalidade provida pela plataforma é implementada.

- **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])**

O modelo independente de computação ou CIM 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])**

O modelo independente de plataforma ou PIM 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 - Unified Modeling Language) 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])**

O modelo específico de plataforma ou PSM 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

[6] Do inglês Computation Independent Model
[7] Do inglês Platform Independent Model
[8] Do inglês Platform Specific Model

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) quanto no sentido 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).

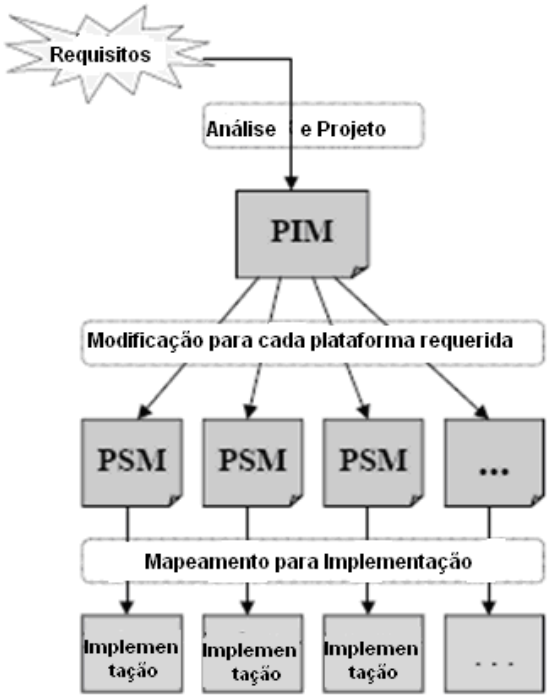

Figura 4.1. Transformações em MDA [Adaptada de Hitachi 2002]

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 automatizadas, passando pelas semi-automáticas. Por exemplo, transformações de PIM para PSM podem ser realizadas através do uso de UML *Profiles* (extensões de UML), uso de padrões (*patterns*), marcas (*markings*), metamodelos (*metamodels* - modelos que descrevem e especificam os modelos originais) e transformações automáticas (via algoritmos) [MDA Guide Version 2003]. A seguir, será descrito como as técnicas de metamodelagem e UML Profile são usadas. As outras técnicas mencionadas estão detalhadas na referência bibliográfica citada 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.

  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 (Meta Object Facility) ou CWM (Common Warehouse Metamodel). Depois, a partir das várias regras de transformações entre os elementos desses metamodelos, técnicas de mapeamento são utilizadas para implementar a transformação.

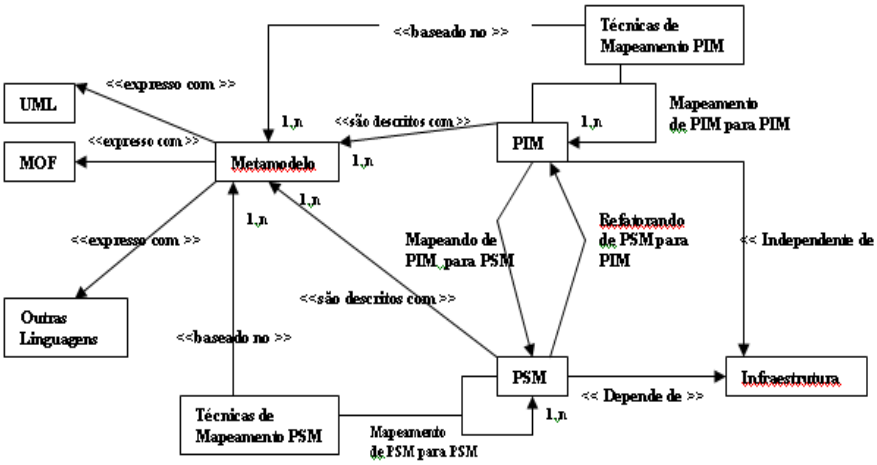

Figura 4.2 Metamodelo MDA [Adaptada de MDA 2001]

Para ilustrar o esquema geral de transformações através de metamodelos, considere a Figura 4.3, onde se tem como entrada um modelo 1, instância do metamodelo A, que é transformado num modelo 2, instância do metamodelo B. É importante destacar que para realizar essa transformação é necessário ter primeiramente os metamodelos especificados em algum padrão, por exemplo, MOF e implementação das regras de mapeamento entre esses metamodelos através de alguma linguagem de transformação modelo-modelo (ATL ou QVT).

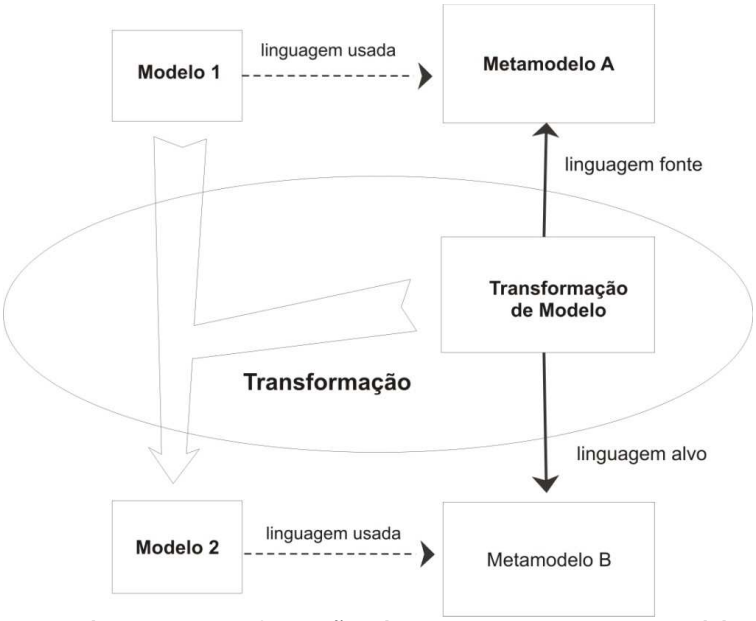

Figura 4.3. Transformações de mapeamentos por metamodelos [Adaptada de MDA Guide Version 2003]

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 à modelagem de classificadores reusáveis (metaclasses que representam elementos UML que podem ter suas propriedades ou métodos compartilhados por outros elementos UML ou generalizados) e também suporte à 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 & Ghosh 2006] cita que o 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 ao armazenamento e intercâmbio de informação de modelos UML.

No entanto, UML carece de precisão semântica, pois muitos dos seus elementos (primitivas) têm diferentes interpretações que variam conforme entendimento do projetista, causando muitas ambigüidades [France & Ghosh 2006]. Oscar Pastor [Pastor & 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, diferentes projetistas podem especificar que um relacionamento entre classes em UML seja uma associação, uma agregação ou uma herança e, para cada um desses tipos de relacionamento, pode-se ter implementações do sistema completamente diferentes. Isso ocorre porque os conceitos de relacionamentos de classes em UML não são precisos e claramente definidos, dando margem a múltiplas interpretações.

Essa imprecisão, aliada à 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 & 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. Como se observa a partir do projeto do sistema (modelo PIM), uma determinada ferramenta CASE MDD pode permitir que seja selecionado o UML Profile que estenda e transforme esse modelo de projeto num modelo de plataforma específica (PSM) desejado, por exemplo, EJB ou .NET . Obtendo-se esse modelo PSM específico, há finalmente uma nova transformação para gerar o código do sistema numa linguagem de programação suportada pela plataforma. Dessa forma, os modelos PIM (ou CIM se exitir) ficam independentes e isolados dos detalhes e mudanças que venham a ocorrer nos modelos de mais baixos níveis (PSM e Implementação). A portabilidade é alcançada, pois se pode gerar modelos PSM qualquer que seja a plataforma suportada pelo ambiente CASE MDD.

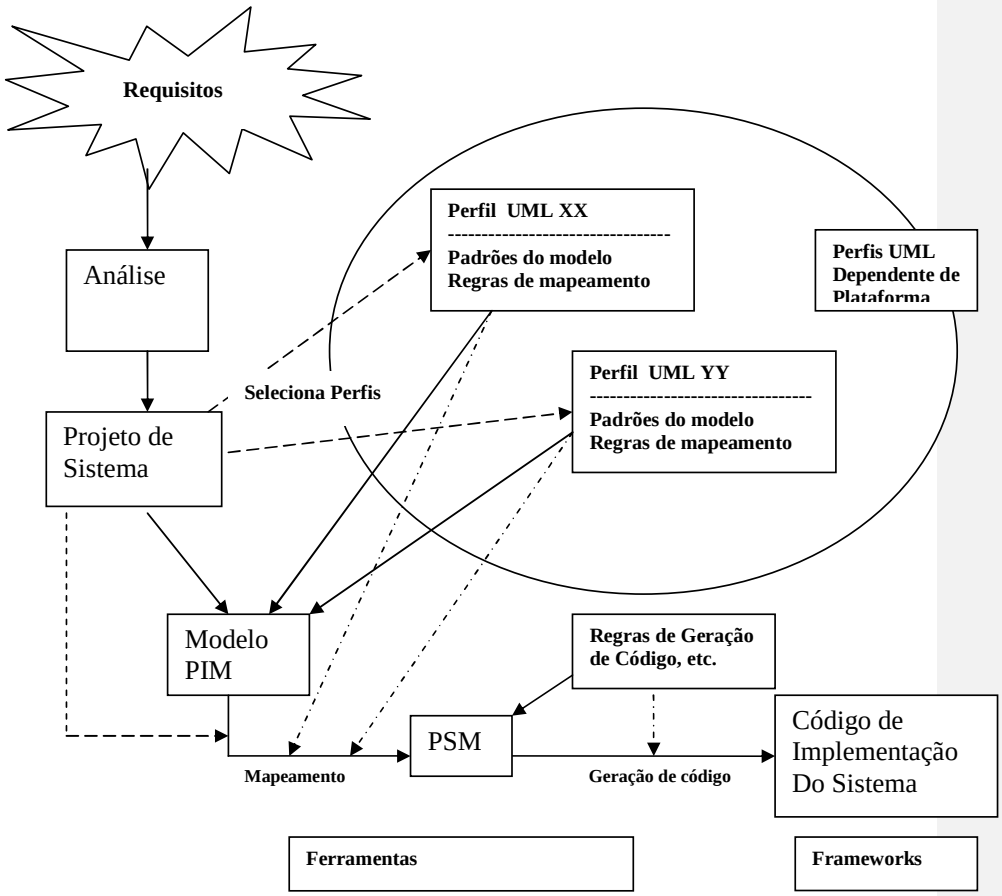

Figura 4.4. Transformações com UML Profile [Adaptada de Hitachi 2002]

Atualmente muitas extensões já estão padronizadas pela OMG, algumas estão em processo de padronização e outras ainda estão em discussão [MDA profile catalog 2010] como mostrado na Figura 4.5.

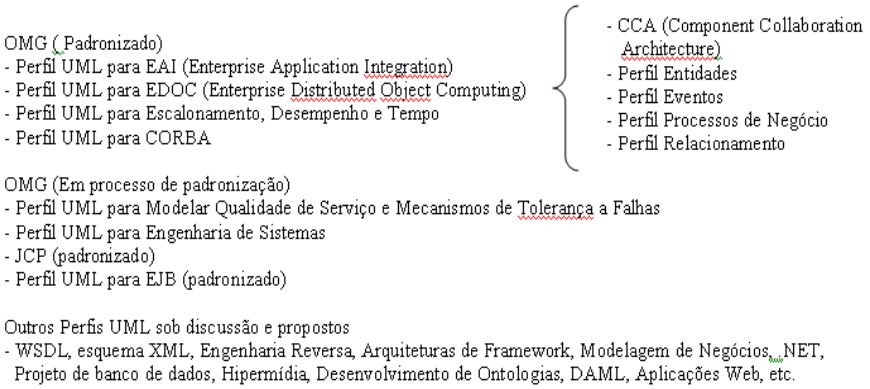

Figura 4.5. Perfis UML da OMG [Adaptada de Hitachi 2002]

Assim, 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 & 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 a modelos só o tempo dirá.

4.2.2. 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, bem como melhorar a portabilidade e a interoperabilidade dos sistemas.

Para atingir esses objetivos e separar os níveis de abstrações, MDA [OMG 2003] foi definida pela OMG em três camadas conforme Figura 4.6.

A primeira camada de especificação (núcleo da arquitetura) representa o modelo PIM e possui os seguintes padrões que ditam um conjunto de regras para estruturação da especificação expressa nos modelos, e que não abordam características de plataformas específicas:

- 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 de software complexos;
- Metamodelo Comum de Armazenamento (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 (MOF): padrão que define uma linguagem abstrata para definição de linguagens de modelagem (metamodelos). Esta linguagem é utilizada para descrever modelos da UML, CWM e o 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 inerentes a determinadas tecnologias e plataformas. Essas plataformas e tecnologias 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:

- Intercâmbio de Metadados em XML (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 a 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.

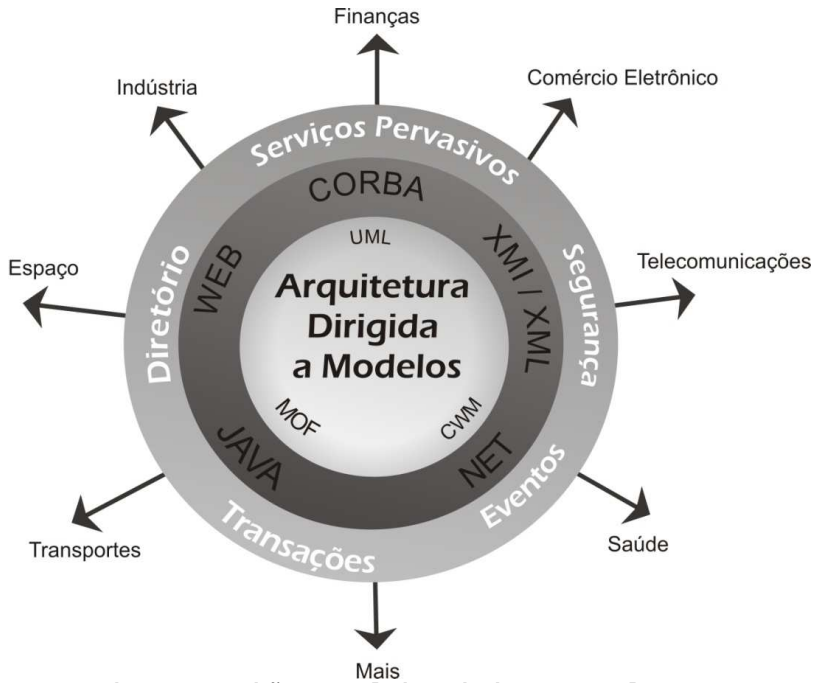

Figura 4.6. Padrões MDA [Adaptada de OMG 2003]

4.3 Abordagens MDD

Esta seção tem como principal objetivo descrever as seguintes abordagens MDD: OO-Method, que apresenta características de um real ambiente MDD através de uma

completa transformação de modelos; e AndroMDA que está se tornando bastante popular.

4.3.1. OO-Method

O OO-Method inova com o conceito de compilador de modelos que, de fato, é uma máquina virtual de transformação de modelos. Além disso, o modelo de alto nível, chamado modelo conceitual, tem todos seus elementos (primitivas) descritos numa notação visual (gráfica) e são especificados numa linguagem formal orientada a objeto (OASIS). Essas características fazem com que o OO-Method 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.

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

"Para desenvolver um sistema de informação é necessário e suficiente definir seu esquema conceitual"

Esta ideia também aparece em trabalhos e propostas de outros pesquisadores de prestígio. Toni Morgan defende a ideia de usar Programação não Extrema (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 de 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 modelo conceitual do OO-Método, passível de transformação através de um compilador de Esquema Conceitual.

O processo básico de transformação

OO-Method 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 inicia com uma 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 compõem 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 objetos 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.

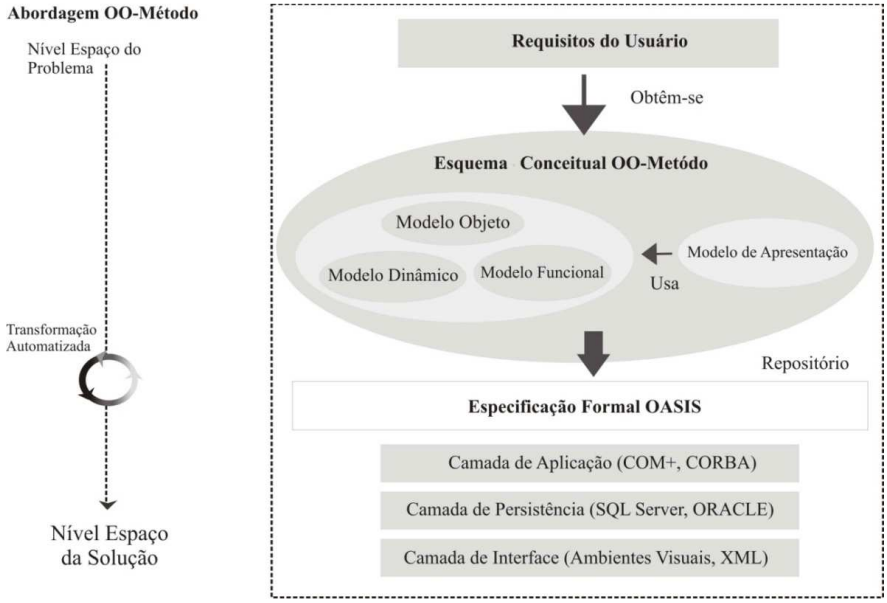

Figura 4.7. Abordagem OO-Method [Adaptada de Pastor & Molina 2007]

Comparação com MDA

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

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

MDA	OO-Method
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-Method 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, o qual é implementado através de uma máquina virtual de compilação de modelos onde se realiza uma verdadeira transformação desses modelos.
- Em termos de PIM, OO-Method 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-Method são também computacionalmente completos e contêm 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: modelo objeto, modelo dinâmico, modelo funcional e modelo de apresentação.

O Modelo Conceitual do OO-Method utiliza uma notação visual (gráfica) similar a UML, formada apenas por conceitos (diagramas) necessários e suficientes para representar um sistema de informação. Além disso, a grande diferença, quando comparado com UML, é que o modelo conceitual do OO-Method tem uma semântica e sintaxe bem precisas e baseadas na 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. Maiores detalhes podem ser encontrados em [Pastor & Molina 2007].

- **Modelo Objeto**

Este modelo 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

O OO-Method representa precisamente os relacionamentos dos seguintes tipos: 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. É estática quando os objetos associados não mudam durante o tempo. É dinâmica quando as instâncias associadas mudam com o tempo durante o ciclo de vida dos objetos que participam do relacionamento.

A agregação é um tipo de associação mais restrita onde uma das classes é o todo e a outra representa a parte do todo (objetos agregados ao todo), significando que uma das classes é constituída ou formada por um conjunto de outros objetos agregados. Como exemplo de relacionamento de agregação, pode-se definir as classes Caixa e Bombons.

A composição é um tipo mais forte de agregação onde a parte e o todo estão vitalmente associados, ou seja, para um objeto-todo existir os demais objetos-parte também devem existir. Como exemplo de composição pode-se definir as classes Corpo Humano e Orgão. Obviamente, a classificação de um relacionamento em associação, agregação ou composição depende de como o analista interpreta as classes de acordo com um determinado contexto.

Além da precisão que o OO-Method 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-Method suporta também os conceitos de Herança (generalização e especialização) e herança múltipla; e para gerenciar a complexidade do modelo, pode-se representar um subsistema através de uma notação visual igual à de um pacote (*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

E "**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**

Este modelo especifica o relacionamento entre os modelos objeto (que é estático) e dinâmico visto no tópico anterior. O modelo funcional consiste em:

- Definição da 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 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-Method, combinado com sua linguagem formal 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 (UML 2.0), criado para modelagem de aspectos funcionais, não possui uma semântica definida de forma clara e precisa. O modelo funcional do OO-Method e sua linguagem formal proveem uma solução adequada, permitindo:

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

- **Modelo de Apresentação**

Este modelo especifica os requisitos de Interface de Usuário, modelando uma interface abstrata que é independente de plataforma ou dispositivo. O modelo de apresentação em nível de análise (conceitual) é considerado uma inovação do OO-Method em relação a outras abordagens que, na maioria das vezes, descrevem interface-usuário apenas em nível de implementação. No OO-Method, 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 que descreve um cenário particular de interação entre o usuário e o sistema. Geralmente, a interface de usuário de um sistema é definida como uma coleção de unidades de interação relevantes e pelo modo como essas unidades estão estruturadas. O OO-Method provê quatro tipos de unidades de interação, descritas a seguir:

  - Unidade de Interação de Serviço: representa 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, 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**

  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, tendo um nó raiz e respectivas ramificações até chegar às folhas. Por exemplo, um sistema de aluguel de carros (nó raiz) pode ser organizado 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-Method dá suporte à 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 quatro modelos básicos (objeto, dinâmico, funcional e apresentação) do esquema conceitual do OO-Method, tem-se toda infraestrutura necessária e suficiente para representar um sistema de informação no contexto do espaço do problema.

O Compilador de Modelos

Para ser aderente ao padrão de desenvolvimento MDD, OO-Method precisa de alguma forma de 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-Method inova com essa ideia de compilador de modelos, tornando-o diferente de outras abordagens que apenas focam na 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-Method 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 transformações insuficientes 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-Method

A implementação do OO-Method é suportada através da ferramenta OlivaNova da Care Technologies [OlivaNova 2009]. O principal objetivo da OlivaNova é separar o que deve ser feito (espaço do problema), de como deve ser feito (espaço da solução). Ela é composta por 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-Method, gerando código fonte na plataforma destino.

4.3.2. AndroMDA

AndroMDA [AndroMDA 2009] é uma abordagem MDD que utiliza UML Profiles para transformação de modelos PIM em PSM. Essa abordagem MDD é implementada por uma ferramenta open source também denominada de AndroMDA.

AndroMDA utiliza como entrada modelos PIM desenhados por qualquer ferramenta que modele em UML (nível projeto) e que exporte esse modelo UML no formato OMG de intercâmbio (XMI). Depois disso, seleciona-se o Perfil UML específico para transformar esse modelo de projeto para a plataforma específica desejada (JAVA, .NET, etc).

Em AndroMDA é possível gerar componentes para as linguagens: Java, .Net, HTML, PHP. Caso seja necessário utilizar alguma tecnologia que ainda não esteja contemplada, é preciso somente desenvolver plugins, chamados pela comunidade de cartuchos, ou alterar algum plugin já existente. AndroMDA é mais comumente utilizado por programadores da tecnologia J2EE, inclusive podendo gerar um projeto J2EE e seu código partindo de um modelo de classe. Além disso, pode-se gerar automaticamente código para Hibernate (framework de persistência em banco de dados de objetos Java e .NET), Enterprise JavaBeans-EJB (principal componente da plataforma Java Enterprise Edition), Spring (framework open source para desenvolvimento em J2EE), WebServices (protocolo e padrão para integração de aplicações via web) e Apache

Structs (framework open source para desenvolvimento de aplicações web em Java compatível com a arquitetura MVC).

A abordagem de AndroMDA permite fazer com que o trabalho de arquitetura e desenvolvimento seja mais curto e de melhor 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.

A partir do PIM efetua-se 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 uma grande variedade de plugins (cartuchos) já desenvolvidos e que realizam a transformação PIM → PSM para vários tipos de linguagens, tecnologias e plataformas diferentes.

Além das vantagens citadas anteriormente, destacam-se os seguintes pontos positivos de AndroMDA:

- Não desenvolvimento de código redundante, o código reflete exatamente o que definem os modelos e existe a possibilidade de alterar de o respectivo plugin para gerar o mesmo sistema em outras linguagens.
- Para se desenvolver novos plugins para qualquer plataforma específica deve-se basicamente identificar as regras de transformação e criar um perfil UML.

Como mencionado, AndroMDA trabalha com modelos de entrada, no seu processo de transformação, no nível PIM (projeto). A ferramenta de melhor aceitação para modelar em UML e fazer exportação do metamodelo UML(formato XMI) para AndroMDA é a Magicdraw. Entretanto em [AndroMDA 2009] há uma lista extensa de ferramentas modeladoras UML que podem ser utilizadas para exportar modelos PIM para AndroMDA.

Apenas como uma breve comparação, AndroMDA não provê recursos para definição de interface do usuário abstrata tal como existe em OO-Method. A abordagem usa conceito de sincronização de modelos (PIM e PSM), diferindo da abordagem OO-Method que utiliza “compilação de modelos”. Além disso, trata questões de rastreabilidade e validação de modelos de forma bem mais limitada que o OO-Method.

A versão atual estável de AndroMDA é 3.3 que suporta UML 2.2. Já a versão 4.0 que está sendo desenvolvida visará aperfeiçoar o processo de transformação de modelos e, principalmente, receber como entrada "input", metamodelos de qualquer linguagem de domínio específico. Isso, tornará o ambiente MDD de AndroMDA mais extensível, flexível e eficiente, capaz de importar qualquer outro modelo de sistema, e não apenas, UML.

Excluído: ¶

4.4.Problemas e Desafios dos Processos MDD

4.4.1. Visão Geral

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

A abordagem MDD ainda está na sua infância. Nem as linguagens de modelagem e 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, 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 XP cujo foco predominante é a implementação de código.

Como trabalhos futuros e desafios para o processo desenvolvimento de software dirigido a modelos, destacam-se os seguintes:

- Integração com a etapa de requisitos para elevação do nível de abstração inicial a ser modelado (modelo CIM da MDA); [Martinez A 2008]
- Suporte a modelos orientados a metas (*goals*), agentes e aspectos; [Van Lamsweerde A 2008]
- Melhoria da precisão semântica dos modelos em relação às características estáticas, dinâmicas; [Alencar, F. 2003]
- Melhores mapeamentos entre os modelos; [Alencar, F. 2009]
- Melhor transformação automática de modelos (automação); [Nurcan S., 2010]
- 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 entre elementos dos modelo CIM->PIM->PSM; [Spanoudakis G, 2005]
- Melhor suporte à engenharia reversa;
- Suporte à computação autonômica, ou seja, geração de sistemas que têm as propriedades de autoconfiguração, auto-recuperação, auto-otimização e autoproteção; [STERRIT, R 2005]

4.4.2.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. Um resumo com os passos necessários para se desenvolver uma solução MDA adequada é apresentado a seguir:

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

4.4.3. O programa FastStart da OMG

Recentemente, o grupo OMG lançou o programa "FastStart" para ajudar as organizações a 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 a equipe executiva e técnica da empresa a:

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

4.5. Considerações finais

O processo de 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 ideia 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 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.

4.6.Tópicos de Pesquisa

O Processo de desenvolvimento de software dirigido a 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 eles 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 evoluir 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 o processo de desenvolvimento específico de software a ser utilizado. Processos existentes tais como RUP, OPEN ou XP podem ser adaptados para incorporar automação e transformação de modelos conforme a abordagem 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 que transformar modelos para várias plataformas (sistemas operacionais, arquitetura de hardware, linguagens de programação, *middlewares* diversos – JAVA/CORBA, COM+/.NET, Web Services), etc.
- MDA em Linhas de Produtos de Software (LPS): A necessidade de produzir aplicativos distintos a partir de pontos comuns e variáveis, utilizando tecnologia de LPS, tem MDD como um forte aliado na automatização do desenvolvimento de software.
- 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

Excluído: ¶

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.

- Análise da aderência do MDD a modelos de qualidade de software: Modelos de qualidade como CMM ou ISO atualmente são empregados para saber se um determinado processo/produto está aderente ou atende a um determinado nível de qualidade. Há uma tendência e necessidade de aplicação desses modelos de qualidade aos processos de engenharia de software MDD.

Excluído:

4.7.Sugestões de Leitura

Este capítulo propôs-se a dar uma visão geral sobre Desenvolvimento de Software Dirigido a modelos. Para complementar o material apresentado neste capítulo, sugere-se que o leitor realize leituras relacionadas aos seguintes temas:

- 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 4.2.1 e 4.3.1, UML carece de precisão semântica. Assim, dependendo da interpretação do projetista pode-se modelar um determinado relacionamento como agregação, composição, associação ou herança. Visando aperfeiçoar a precisão semântica e formalidade da linguagem UML, existe um grupo de trabalho desenvolvendo Precise UML (pUML). Detalhes podem ser encontrados 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, que roda no Eclipse IDE, sendo considerado também o estado da arte em desenvolvimento dirigido a modelos. Possui boa documentação adicional e a ferramenta está disponível para download em http://www.openmdx.org/.

- Livros específicos sobre o tema:

- **MDA Explained: The Model Driven Architecture: Pracice and Promise** by Anneke Kleppe, Jos Warmer, Wim Bast (Editora Addison Wesley 2003).
- **Model-Driven Software Development** by Sami Beydeda, Matthias Book , Volker Gruhn (Editora Springer 2005).
- **Real-Life MDA: Solving Business Problems with Model Driven Architecture (Interactive Technologies)** (Editora Morgan Kaufmann, 2006).

4.8. Exercícios

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

1. Quais os benefícios em se adotar um processo de desenvolvimento de software dirigido a modelos?
2. MDA usa modelos distintos para separar os sistemas/aplicações em os níveis de abstrações/visões distintos. Quais são estes modelos? Descreva o propósito de cada um.
3. Descreva a técnica de transformação de modelos através de metamodelos.
4. Quais os padrões que estão no núcleo da arquitetura MDA do grupo OMG?
5. 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)?
6. Caracterize o modelo conceitual do OO-Método.
7. O que é o Compilador de Modelos do OO-Método?
8. Compare as ferramentas CASE de desenvolvimento de software dirigido a modelos: OLIVANOVA e AndroMDA.
9. 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?
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, com recursos de persistência num banco de dados escolhido (Mysql, Postgresql, etc.) e de distribuição (*deployment*) num servidor JSP Tomcat Apache. Que comentários você pode dar sobre o processo de transformar seu modelo PIM em PSM automaticamente? O 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).

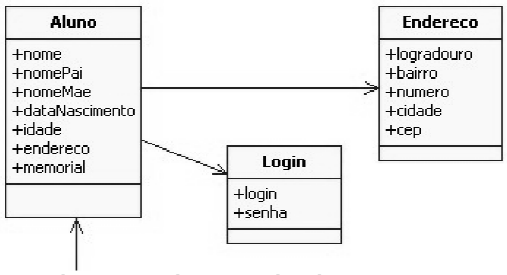

Figura 4.8 Diagrama de Classes

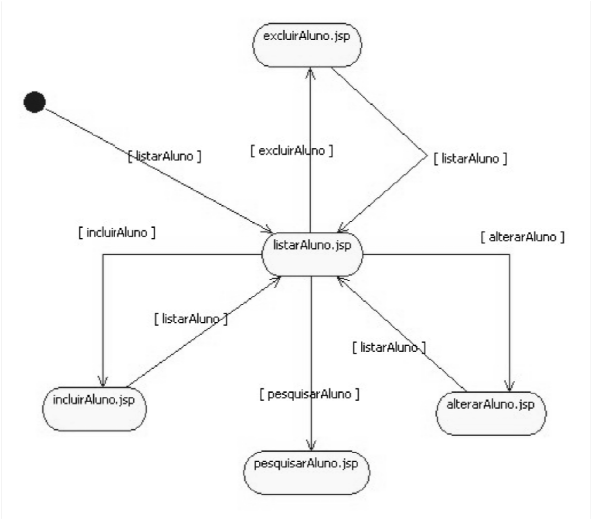

Figura 4.9 Diagrama de Atividades

4.9. 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.

OMG: QVT specification (2009).< http://www.omg.org/spec/QVT/ > Acesso em julho 2010.

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.

MDA profile catalog.<http://www.omg.org/technology/documents/profile_catalog.htm. Acesso em julho 2010.

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.

Eclipse ATL documentation 2009.<http://www.eclipse.org/m2m/atl/doc/>. Acessado em julho 2009.

Martinez A (2008) Conceptual schemas generation from organizational models in na automatic software production process. Phd Thesis

Van Lamsweerde A (2008) Systematic requirements engineering- from system goals to UML models to software specifications. Wiley Publisher 2008

Alencar, F., Marín, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.H.: From i* Requirements Models to Conceptual Models of a Model Driven Development Process. In: PoEM 2009. Springer LNIBP (2009)

Excluído: ¶

Alencar F.M.R., Pedroza F., Castro J., Amorim RCO (2003). New mechanisms integration of organizational requirements and object oriented modeling. In Proceedings of the VI workshop on requirements engineering (WER'03), Piracicaba - SP, Brasil, p.109-123.

Nurcan S., Salinesi C., Souveyet C., Ralyté J. Intentional Perspectives on Information Systems Engineering. pp. 257-275. Springer Publisher 2010

Spanoudakis G, Zisman A (2005) Software Traceability: A Roadmap. Handbook of Software Engineering and Knowledge Engineering, Vol. III: Recent Advancements. World Scientific Publishing Co., 395–428

STERRIT, R. Autonomic computing. Innovations in Systems and Software Engineering, 2005.