Viabilidade de Construção de Software com MDD e MDA

André Sandri

Ciência da Computação – Centro Universitário La Salle (UNILASALLE)
Av. Victor Barreto, 2288– 92.010-000 – Canoas – RS – Brazil
andresandri@hotmail.com

Abstract. *This article presents a software development viability study with MDD - Model Driven Development approach using MDA - Model Driven Architecture standards.*

Resumo. *Este artigo apresenta um estudo da viabilidade da construção de software através da abordagem MDD – Model Driven Development utilizando padrões MDA – Model Driven Architecture.*

1. Introdução

MDD – *Model Driven Development*, é uma nova abordagem de desenvolvimento de software focada no modelo (que podem ser diagramas UML), e não na linguagem de programação. Ao invés de construir um software utilizando linguagens de programação, MDD permite "desenhar" as funcionalidades e recursos desejados utilizando um conjunto de modelos. Dessa maneira, a maior parte do esforço ficará em cima do modelo, e não mais nos códigos-fontes. O software será gerado a partir dos modelos, gerando código-fonte ou não. Note que através desta abordagem será possível futuramente eliminar as linguagens de programação e seus códigos-fonte deste processo, pois o software poderá ser executado diretamente a partir da leitura dos modelos (KWB, 2003).

MDA – *Model Driven Architecture*, é uma abordagem padronizada aberta e independente de fornecedor criada pela OMG - *Object Management Group*, para solucionar os desafios das mudanças de negócio e de tecnologia. Contempla padronizações de especificações de sistemas e de operação independente de tecnologia baseados na utilização de modelos formais. É uma padronização abrangente que deve ser utilizada com MDD (OMG, 2005).

Este artigo apresenta um estudo da viabilidade de construção de software utilizando estas duas abordagens, que são inter-relacionadas. Inicialmente são apresentados os principais conceitos utilizados nestas abordagens. Em seguida, são apresentados os recursos essenciais necessários para tornar a construção de software com MDD e MDA viável.

2. Principais Conceitos

Antes de apresentar o estudo da viabilidade, é imprescindível apresentar os principais conceitos envolvidos nestas abordagens, que serão utilizadas no decorrer deste artigo.

2.1. MDD – *Model Driven Development*

MDD está cada vez mais ganhando a atenção da indústria e de comunidades de pesquisa. MDD visa utilizar modelos na maior parte do tempo durante um processo de desenvolvimento de software, e prevê automação através da execução de modelos, transformações e técnicas de geração de código (KWB, 2003).

A abordagem MDD promete aumentar a produtividade daqueles que tem a tarefa de desenvolver e manter um sistema de software (SEARCHWEBSERVICES, 2004). Se um maior índice de produtividade for desejado durante a manutenção de sistemas, uma pesquisa da Compuware indica que MDD é a resposta. Resultados desta pesquisa indicam ganhos de 70% durante a fase de manutenção do software. A equipe que utilizou MDD completou cinco recursos 37% mais rápido do que a equipe tradicional, em 165 horas contra 260 horas. (COMPUWARE, 2005)

Um dos projetos do Eclipse, o projeto MDDi - *Model Driven Development Integration*, que está em sua fase inicial de desenvolvimento, é dedicado a fortalecer a sua plataforma de desenvolvimento em Java oferecendo tecnologias de integração necessárias para a aplicação da abordagem MDD. Este projeto irá produzir *frameworks*[1] extensíveis e ferramentas para suportar várias linguagens de modelagem (UML, *Domain-Specific Languages*) e metodologias. Na prática, é um esforço de criar uma alternativa totalmente gratuita com código aberto que suporte MDD com MDA (MDDI, 2005).

2.2. MDA – *Model Driven Architecture*

O OMG é uma associação de várias empresas que se uniram para criar padrões visando facilitar a integração de recursos de TI. Inicialmente, quando o OMG iniciou suas atividades em 1989, o foco era desenvolver padrões de *middleware*[2] orientados a objeto. O resultado disso foi a criação do CORBA, uma tecnologia que permitiu que as empresas conectassem toda e qualquer aplicação com qualquer outra, escrito em qualquer linguagem orientada a objetos, indiferente de plataforma ou sistema operacional. Com o passar dos anos o OMG trabalhou com seus membros para criar uma variedade de outros padrões, incluindo o UML - Unified Modeling Language, a notação padrão hoje utilizada por desenvolvedores de software no mundo todo (OMG, 2005).

---

[1] *Framework*: No desenvolvimento de software, um framework é uma estrutura de suporte definida em que um outro projeto do software pode ser organizado e desenvolvido. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes do seu projeto.

[2] Middleware: Software de interface que permite interação de diferentes aplicações de softwares, geralmente sobre diferentes plataformas de hardware e infra-estrutura, para troca de dados.

Em 2001, após uma revisão do estado da arte das tecnologias, o OMG concluiu que a heterogeneidade de tecnologias existentes no mercado era permanente. Conforme o tempo passa, cada grande empresa é forçada a suportar novas linguagens de programação, outros sistemas operacionais, além de uma variedade de protocolos de rede. Então, decidiram que o principal objetivo, a interoperabilidade, poderia somente ser conseguido com um sistema padronizado e público de modelos e interfaces que são independentes de linguagem, sistema ou protocolo.

MDA é este sistema. Essencialmente, o OMG está padronizando diversos modelos abstratos para que as empresas usem para representar aplicações. Alguns destes padrões resultam em modelos que são independentes dos detalhes da implementação. Outros padrões complementam as características existentes em linguagens de programação, de sistemas operacionais ou de redes. Todos os modelos que uma empresa define podem ser integrados por interfaces e mapeamentos que asseguram que uma empresa possa inicialmente desenvolver um modelo independente de plataforma de uma aplicação, e em seguida possa gerar um modelo específico para uma linguagem e um sistema operacional escolhido. Futuramente, caso a empresa decidir executar a aplicação em uma linguagem ou sistema operacional diferente, basta reutilizar o modelo independente de plataforma para gerar um modelo novo, específico para a aplicação nova. Assim, enquanto que as empresas evoluem e novas tecnologias são introduzidas, as aplicações, documentadas como modelos independentes da plataforma, serão preservadas e podem ser reutilizadas (OMG, 2005).

Os membros do OMG acreditam que esta solução possa resultar em uma arquitetura que sirva a uma empresa por uma década, independendo das mudanças nas linguagens de programação, dos sistemas operacionais e dos protocolos de rede.

A solução MDA engloba diversos padrões e conceitos, os quais serão apresentados a seguir.

2.2.1. UML – *Unified Modeling Language*

UML é a especificação mais utilizada da OMG, pois é a maneira mais utilizada para modelar um sistema com seu comportamento, sua arquitetura, seus processos de negócio e estruturas de dados.

Esta especificação auxilia o desenvolvedor a especificar, visualizar e documentar modelos de sistemas, incluindo sua estrutura e *design*[3], de maneira que comporte todas as exigências relatadas por requisitos. Pode também ser utilizado para modelar processos de negócio e sistemas que não são softwares, muitas vezes utilizando perfis (*profiles*) para estes casos.

Profile (perfil) é um mecanismo especializado definido como parte do UML. Um *profile* define uma maneira específica de utilização do UML. Por exemplo, "CORBA Profile" define uma maneira específica de utilizar UML para modelar interfaces CORBA, e "Java Profile" define uma maneira de modelar código-fonte Java em UML (OMG, 2005).

[3] *Design*: é um termo da língua inglesa que se refere a um determinado esforço criativo, seja bidimensional ou tridimensional, no qual se projetam objetos ou meios de comunicação diversos para o uso humano. Devido a este fato, ela pode ser traduzida como "desenho", mas não se refere ao ato de desenhar. Devido à dificuldade de tradução, costuma-se adotar a palavra original.

Profiles (perfis) foram introduzidos no UML 1.3 como uma maneira de estender esta linguagem de modelagem. Antes do UML 2.0, a idéia de estender UML estava limitada a estereótipos (*stereotypes*) e *profiles*. A idéia de estender mudou ligeiramente no UML 2.0. Existe agora um novo mecanismo, o meta-modelo. Um meta-modelo é um mecanismo subjacente que define uma linguagem para expressar um modelo. Em outras palavras, o meta-modelo do UML é um modelo que é utilizado para definir o UML, e é possível adicionar novas regras ao meta-modelo de forma a estender o UML (KWB, 2003).

Atualmente o UML está em sua segunda versão - UML 2.0, que teve como principal evolução a modelagem visual. Os novos recursos permitem que esta nova versão contemple novos elementos que existem atualmente em novas tecnologias de software, como MDA e SOA - *Service-Oriented Architecture* (OMG, 2005).

2.2.2. MOF – *Meta-object Facility*

MOF é um padrão da OMG que define a linguagem utilizada para definir modelos padronizados. É a linguagem em que as definições de UML e de CWM (que será apresentado a seguir), ou seja, os meta-modelos de UML e de CWM, são escritas. Ele não é usado somente para definir modelos padronizados, mas também para permitir a construção de ferramentas que auxiliam nesta atividade (OMG, 2005).

UML é uma das linguagens de modelagem definida com MOF. O meta-modelo de UML descreve exatamente como um modelo de UML é estruturado. A figura abaixo demonstra uma parte simplificada deste meta-modelo.

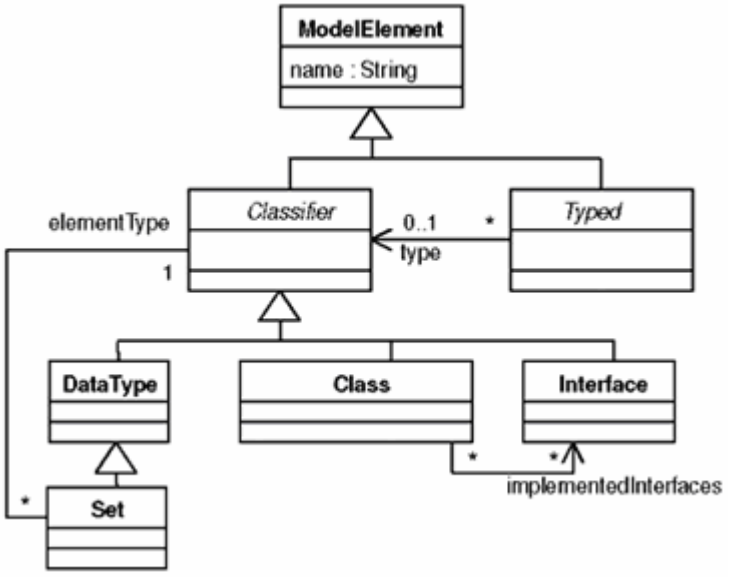

Figura 1. Meta-modelo simplificado do UML (KWB, 2003).

O meta-modelo de UML descreve exatamente como um modelo de UML é estruturado. Do meta-modelo apresentado na figura 1, pode-se deduzir o seguinte:

- No topo podemos ver que tudo em UML é um elemento do modelo (*ModelElement*) e possui um nome (*name*).
- A meta-classe abstrata Classificador (*Classifier*) é uma generalização de uma classe (*Class*), Interface, e tipo de dado (*DataType*). Eles possuem

muitas características comuns, mas algumas características particulares também.

- Uma classe (*Class*) pode implementar uma Interface, mas não pode implementar um tipo de dado (*DataType*) ou uma Interface.

Todo o modelo de UML deve cumprir estas regras; senão, não é uma instância correta do meta-modelo de UML. Por causa disto, pode-se utilizar modelos de UML sabendo exatamente como são representados e como são estruturados.

O papel principal do MOF dentro do MDA é que ele dá os conceitos e as ferramentas em relação aos modelos padronizados. Usando a definição de MOF sobre uma linguagem de modelagem pode-se definir transformações entre os modelos. E já que as transformações são definidas utilizando os meta-modelos da linguagem de modelagem envolvida, podem ser aplicadas a todo o modelo escrito em qualquer linguagem. Sem uma linguagem padrão para descrever os meta-modelos, as transformações não poderiam ser definidas corretamente e a utilização do MDA seria muito difícil de realizar. Enfim, o MOF é a tecnologia principal para o MDA (KWB, 2003).

Um dos padrões incorporados ao MOF é o OCL – *Object Construct Language*, que é a linguagem utilizada para descrever padrões MOF (WK, 2004).

Atualmente um padrão novo está sob desenvolvimento na OMG, chamado de QVT - *Query*, *Views*, *and Transformations*. Este padrão contemplará uma maneira padrão onde as transformações serão especificadas entre os modelos cujas linguagens são definidas usando o MOF. Será parte do MOF, e terá principalmente os seguintes componentes (QVT, 2005):

- Uma linguagem para a criação de visões sobre os modelos;
- Uma linguagem para efetuar pesquisas sobre os modelos;
- Uma linguagem declarativa para descrever transformações;
- Uma linguagem visual para descrever transformações.

Este novo padrão, que está em sua 3ª revisão, está atualmente na fase final de votação e já foi aprovado. Conforme a agenda do OMG, entre 5 a 9 de dezembro de 2005, na cidade de Burlingame, Califórnia, USA, ocorrerá a votação entre membros da OMG para a recomendação deste padrão (OMG, 2005). Apesar de ainda não ser um padrão publicado, já existem ferramentas que permitem utilizá-lo, de forma experimental. E existem também ferramentas que realizam transformações a partir de linguagens proprietárias.

2.2.3. XMI – *XML Metadata Interchange*

XMI é um formato padrão recomendado pela OMG desde 1999, que tem como objetivo o intercâmbio de dados possibilitando o compartilhamento de modelos entre ferramentas de modelagem diferentes de uma forma padronizada, trazendo assim consistência e compatibilidade em sistemas criados em ambientes diferentes. XMI possibilita a transferência de modelos UML e meta-modelos baseados em MOF através do padrão XML (OMG, 2005).

XMI é uma tecnologia da OMG, mas é baseado no padrão XML da W3C[4]. Por isso ele é conhecido por integrar três padrões: XML da W3C, UML e MOF.

2.2.4. CWM – *Common Warehouse Metamodel*

CWM é uma linguagem de modelagem que foi projetada especificamente para modelar aplicações de *data warehousing*[5]. Possui interfaces e relações padronizadas que podem ser usadas para permitir um fácil intercâmbio de meta-dados de sistemas de *Data Warehouse* e *Business Intelligence* entre ferramentas, plataformas e repositórios em ambientes heterogêneos distribuídos (OMG, 2005).

O meta-modelo tem muito em comum com o meta-modelo de UML, mas tem um número de meta-classes especiais, como por exemplo, para modelar bases de dados relacionais. Os autores do CWM removeram tudo do UML que não era necessário para suas necessidades, e adicionaram detalhes específicos para *data warehousing*.

Pelo fato de *data warehousing* ser uma tecnologia que combina informação de diferentes fontes, este meta-modelo inclui diversos meta-modelos simplificados para bancos de dados relacionais, registros e estruturas, OLAP, XML, transformações, visualização da informação, mineração de dados, banco de dados multidimensionais, processos e operações de *data warehousing*, entre outros.

Todos estes meta-modelos foram modelados em MOF. Assim, todos podem ser utilizados como fonte ou destino para as transformações dentro de um processo de MDA (KWB, 2003).

2.2.5. PIM – *Platform-independent Model*

O padrão MDA define três etapas para a construção de um software. A primeira etapa consiste na criação de um modelo independente de plataforma chamado de PIM – *Platform-independent Model* (OMG, 2005).

Um PIM descreve um sistema completo para uma determinada necessidade de negócio. No PIM, o sistema é modelado a partir do ponto de vista que melhor representa o sistema: o negócio, e apenas o negócio. Neste modelo não existem indícios de que o sistema será implementado em um mainframe com um banco de dados relacional, ou se será implementado em uma aplicação J2EE sobre um sistema Linux (KWB, 2003).

Características principais:

- O PIM deve ser escrito para ser compreendido e corrigido por outros profissionais. Para isso, é importante que a linguagem de modelagem utilizada pelo PIM seja muito bem elaborada, pois deve ser compreendida por seres humanos e por máquinas.
- O PIM deve ser independente de qualquer tecnologia de execução.

[4] W3C: *World Wide Web Consortium*, foi criado em 1994 para levar a *Web* para o seu potencial máximo, através do desenvolvimento de protocolos comuns e fóruns abertos que promovem sua evolução e asseguram a sua interoperabilidade. O W3C desenvolve tecnologias, denominadas *Web Standards* (ou padrões *Web*) para a criação e a interpretação dos conteúdos para *Web*.
[5] *Data warehousing*: O processo de projetar, construir e manter um sistema de *Data WareHouse*.

2.2.6. PSM – *Platform-specific Model*

A segunda etapa definida pelo padrão MDA para a construção de um software consiste na criação automática de um ou mais modelos dependentes de plataforma, chamados de PSM – *Platform-specific Model*. Isto é feito utilizando transformações automatizadas do modelo representado pelo PIM (OMG, 2005).

Um PSM, inversamente ao PIM, deve refletir de forma muito próxima os conceitos e as construções utilizados na tecnologia correspondente a ser utilizada para a implementação do sistema. Em um PSM destinado a bases de dados, para citar um exemplo, os conceitos de tabela, coluna, e chaves estrangeiras devem estar claramente visíveis (KWB, 2003).

A terceira etapa é a geração do código-fonte ou da criação de uma base de dados em um SGBD a partir do PSM gerado. Como os PSMs são automaticamente gerados, cada PSM necessita ser compreendido apenas por ferramentas automatizadas de transformação e por peritos da tecnologia para a qual o PSM foi gerado. Para facilitar isso, atualmente já existem *profiles* de UML para tecnologias específicas, como por exemplo, o *EJB Profile for UML* (KWB, 2003).

2.2.7. Processo de Construção de Software com MDA

Um processo MDA é essencialmente dividido em três etapas (KWB, 2003):

1. Construir um modelo com um alto nível de abstração, o qual deve ser independente de qualquer tecnologia de implementação. Este modelo é chamado de PIM (modelo independente de plataforma);

2. Transformar o PIM em um ou mais modelos para especificar o sistema com os aspectos envolvidos para sua execução em uma tecnologia específica, como por exemplo, um modelo de banco de dados ou um modelo EJB - *Enterprise Java Beans*. Este modelo é chamado de PSM (modelo específico da plataforma).

3. Transformar os PSMs em código.

A seguir tem-se uma descrição do processo envolvido em cada etapa.

Etapa 1 – Criação do PIM

Nesta etapa não deve existir a preocupação com a tecnologia que será empregada, o banco de dados que será utilizado ou mesmo a linguagem ou o servidor de aplicações que será adotado, pois a única preocupação que o analista, engenheiro ou desenvolvedor tem neste momento é em descrever da maneira mais fiel possível a necessidade do usuário. Desta maneira é criado um modelo de classes, um modelo de atividades e um modelo de seqüência, entre outros artefatos, que no seu conjunto servem para criar a aplicação desejada independente das tecnologias disponíveis. Este conjunto é chamado de PIM (KWB, 2003).

Etapa 2 - Transformação do PIM para PSM

Em um determinado momento, quando o modelo PIM já foi devidamente refinado e validado, o próximo passo previsto é a transformação deste modelo em outras formas de representação que aí sim definirá o banco de dados que será utilizado, a linguagem em que será implementada e a arquitetura em que será construída o sistema. O que diferencia o MDA de outras abordagens de desenvolvimento é como é feita esta

transformação, ou seja, com o MDA as transformações deverão ser feitas usando-se padrões (*Patterns*) que necessariamente são códigos previamente testados e adotados pelo mercado como as melhores práticas de programação. Este compromisso é esperado do fornecedor que se propõe a desenvolver uma ferramenta em conformidade com o MDA (KWB, 2003).

Com essa transformação, teremos um modelo mais aproximado do sistema final (modelo PSM), pois já existe a representação dos objetos específicos da linguagem adotada na implementação, e também já existe as definições de tabelas e conexões à base de dados que receberá o sistema. O desenvolvedor precisa apenas trabalhar no refinamento destas informações garantindo que no próximo processo de transformação esteja criado um sistema quase totalmente acabado.

Já que um PSM é um modelo muito aproximado de sua tecnologia, a última transformação é direta. Um relacionamento bem próximo entre o PSM e sua tecnologia assegura de que a transformação resultante ao código seja eficiente e eficaz. A etapa complexa deste processo é transformar um PIM a um ou mais PSMs (KWB, 2003).

Outro elemento chave deste processo é que as transformações são executadas com auxílio de ferramentas. Muitas ferramentas já transformam um modelo de uma plataforma específica para código, e isto não é nenhuma novidade. A novidade aqui é que a transformação de PIM para PSM é automatizada também, e é aqui que o benefício do MDA está. Qualquer um que já trabalhou exaustivamente em desenvolvimento do software sabe quanto tempo é gasto em tarefas rotineiras. Por exemplo, construindo um modelo da base de dados a partir de um projeto orientado a objetos, ou construindo um modelo COM - *Common Object Model*, ou EJB – *Enterprise Java Bean*, tudo a partir de outro modelo com um nível de abstração superior. O objetivo do MDA é automatizar a parte incômoda e repetitiva do desenvolvimento do software.

Todos os modelos, PSM e PIM, devem ser consistentes e precisos, e devem conter tanta informação quanta for possível sobre o sistema. E á aqui que o OCL pode ser útil, porque os diagramas de UML sozinhos por si só não fornecem informações suficientes (WK, 2004).

Etapa 3 - Transformação do PSM para Código

Como já ficou claro, o desenvolvedor aplica uma nova transformação no modelo PSM para criar código-fonte na linguagem desejada.

Algumas das definições de transformação serão criadas e definidas por usuários, isto é, escrito pelos colaboradores que trabalham dentro do processo de MDA. Preferivelmente, as definições de transformação poderão estar publicadas na Internet com acesso público, e talvez virem novos padrões ajustáveis às necessidades individuais dos seus usuários. Alguns fabricantes destes tipos de ferramentas desenvolveram suas próprias definições de transformação, que infelizmente não podem ser adaptadas por usuários porque seu uso não é transparente (KWB, 2003).

3. Estudo da Viabilidade

Para ser viável um processo de construção de software com MDD e MDA, tem que ter, pelo menos, profissionais capacitados e uma ferramenta que auxilie nas tarefas de modelagem e transformação.

Para a escolha dos profissionais que participarão deste processo, é necessário, primeiramente, conhecer os papéis desempenhados pelas pessoas.

3.1. Papéis em um Processo MDA

Quais são as conseqüências de aplicar o MDA no processo do desenvolvimento do software? Comparando o processo de MDA com o processo tradicional, muitas atividades continuarão iguais. Devemos continuar a levantar os requisitos do sistema, testar o sistema, implantar, e continuar a fazer manutenção. O que mudará são as atividades da análise, do projeto de baixo nível e da codificação.

Durante a análise um PIM deverá ser criado. Provavelmente existirá uma equipe especial treinada para esta tarefa, formada provavelmente por analistas de negócio. Eles estarão cientes da funcionalidade que necessita ser implementada, e serão guiados pelas necessidades do negócio e pelo modelo do negócio. Deverão ter conhecimento de UML e estarem acostumados a utilizar uma ferramenta de modelagem.

Provavelmente haverá também uma outra equipe responsável pela transformação do PIM em um ou mais modelos PSM, formada por engenheiros de software. Terão conhecimento de diversas plataformas, arquiteturas, tecnologias, e das definições da transformação que estão disponíveis em suas ferramentas de transformações MDA. Eles irão decidir quais transformações deverão ser utilizadas e quais os valores dos parâmetros destas transformações. Alguns exemplos das questões que o engenheiro de software terá que avaliar:

- Quais plataformas e arquiteturas serão utilizadas pelo sistema?
- O sistema será implementado em três camadas, ou se será implementado em uma arquitetura cliente-servidor?
- O sistema será gerado em J2EE, Delphi ou .NET?
- O sistema rodará em Linux, Windows ou Solaris?

Para encontrar a resposta correta para estas questões, o engenheiro de software deverá obter determinadas informações do analista de negócio (PIM). Ele deverá conhecer os requisitos de negócio não-funcionais. Por exemplo, quando um analista de negócio disser ao engenheiro de software que o sistema será usado intensivamente por cinco mil pessoas, o engenheiro de software escolherá uma arquitetura que suporte isso, ou escolherá outra arquitetura (e assim utilizaria outras definições de transformação) caso o sistema for utilizado por no máximo cinco pessoas.

Uma outra tarefa do engenheiro de software é responder às mudanças no PIM e nas definições da transformação. O PIM e as definições de transformação podem mudar a qualquer momento. Quando os requisitos do negócio mudar, somente o PIM será afetado, pois a transformação para PSM é automatizada. Quando a plataforma do software mudar (no caso em que uma nova versão é instalada, por exemplo) somente a definição da transformação terá que ser modificada. O engenheiro de software deve responder a estas mudanças porque toda a mudança deve ser refletida no PSM gerado, e consequentemente no software gerado.

As transformações serão codificadas pelo engenheiro de software ou por programadores que conheçam a linguagem utilizada pela ferramenta de transformação. Quanto mais detalhes existirem no modelo PIM, e desde que as transformações tratem

desses detalhes, maior será o código-fonte gerado. Inicialmente isso será uma tarefa incremental que e não gerará todo o código-fonte necessário para a execução completa do sistema gerado, e os códigos-fonte deverão ser completados por programadores.

A equipe que definirá as transformações será um fator crítico neste processo, pois caso falharem em alguns pontos, como introduzir erros em aspectos arquiteturais ou com a introdução de defeitos nos códigos gerados, estes defeitos se repercutirão em todos os sistemas gerados. Porém existe também um fator positivo: caso um defeito for detectado e corrigido, todos os sistemas gerados terão a possibilidade de serem corrigidos de forma automatizada e transparente. Esta equipe também deverá ter comunicação direta e transparente com os programadores para incrementalmente melhorar a geração dos códigos-fonte.

3.2. A Escolha da Ferramenta

A escolha da ferramenta é essencial para o processo de MDD com MDA. Como MDA ainda está em sua infância, muitas das ferramentas disponíveis atualmente ainda não oferecem recursos completos visando a produtividade que se espera neste processo. As ferramentas devem oferecer, pelo menos, funcionalidades suficientes, de forma a garantir impacto significativo no processo de desenvolvimento de software. Por isso, a ferramenta a ser adotada deve pelo menos possibilitar:

- Linguagens e ferramentas de transformação produtivas, e de preferência que permitam hoje ou em um futuro próximo a utilização da linguagem QVT (QVT, 2005), pois a tendência é que existirão transformações à venda ou de domínio-público disponíveis para utilização nesta linguagem;
- Geração de código para diferentes linguagens, oferecendo a possibilidade de geração dos sistemas em outra linguagem caso isso for necessário no futuro;
- Geração de código para diferentes mecanismos de persistência (JDO, Hibernate, JBoss 4 POJOs, entre outros), tecnologias e *frameworks*;
- Importar e exportar arquivos XMI de e para diferentes ferramentas de modelagem (Together, MagicDraw, Visio, entre outros);
- Possibilidade de modelar em UML, permitindo também a importação, a criação e a utilização de *Profiles*;
- Mecanismos de controle de concorrência para trabalho em grupo;
- Mecanismos que possibilitem a re-geração dos modelos e dos códigos-fonte sem destruir os artefatos introduzidos manualmente, como trechos de código introduzidos por programadores;

Este artigo apresenta algumas ferramentas que podem ser utilizadas em um processo MDA: JDeveloper da Oracle, o software-livre AndroMDA, alguns produtos da linha Rational da IBM, Together da Borland, ArcStyler da Interactive Objects e OptimalJ da Compuware.

3.2.1. Oracle JDeveloper 10g - Versão 13.1.3

JDeveloper é a ferramenta de desenvolvimento Java da Oracle. É o único produto desta empresa que suporta parcialmente recursos de MDA. Entre suas principais características podemos citar (ORACLE, 2005):

- Sincronização *two-way* entre código-fonte e meta-dados da implementação, ou seja, mudanças nos diagramas (modelos) são automaticamente propagadas para o código-fonte Java, e vice-versa;
- Suporta exportação XMI 1.1 com UML 1.3, 1.4 e 1.5;
- Possui transformações de modelos de classes UML para modelos de Java e modelos de componentes de negócio (*Business Component Model*). Estas transformações são feitas por tecnologia proprietária;
- Integração com a ferramenta AndroMDA;
- Futuramente pretendem introduzir UML 2.0 nesta ferramenta.

Os recursos atualmente existentes na ferramenta são limitados, e a maioria das transformações são focadas em transformações diretas para o Oracle ADF - *Application Development Framework*. Existe interesse da empresa em adicionar novos recursos de MDA na ferramenta em versões futuras (RAMACKERS, 2005). Para uso efetivo desta ferramenta em um processo de MDA, é preferível aguardar disponibilização de recursos complementares em futuras versões.

3.2.2. AndroMDA

AndroMDA, antigamente chamado de UML2EJB, é uma ferramenta de geração de código-fonte *open-source* que segue alinhada ao paradigma MDA. Esta ferramenta recebe como entrada modelos XMI (exportados por outras ferramentas) e gera código-fonte e outros componentes a partir de *templates* pré-configurados. Existem *templates* para diversas tecnologias, principalmente baseadas em Java, como EJB, Struts, Hibernate, etc. Suas principais características (ANDROMDA, 2005):

- Utiliza o conceito de cartuchos (*cartridges*), onde cada cartucho é uma transformação pré-definida para uma tecnologia específica a partir de arquivos de *template*;
- Utiliza um mecanismo de *script* para as transformações em cima dos arquivos *templates* para transformações simples e diretas, possibilitando estender este mecanismo com implementações Java para transformações mais complexas;
- Permite acessar todos os elementos do meta-modelo UML;
- A entrada é um modelo PIM. A ferramenta, antes de gerar o código-fonte, transforma o modelo solicitado em um modelo PSM de acordo com a tecnologia escolhida;
- Utiliza o conceito de plataforma destino, que é uma combinação da linguagem, mecanismo de persistência, mecanismo de comunicação cliente-servidor, entre outras propriedades.

Esta ferramenta só faz transformações e não possui interface para modelagem UML. O ponto positivo é que este projeto disponibiliza vários cartuchos (transformações) prontos para uso. Porém, a linguagem de transformação é proprietária, fato que elimina a portabilidade e a escalabilidade desta ferramenta.

3.2.3. IBM Rational

A IBM, com sua linha de softwares Rational, possui vários produtos que suportam MDD e MDA (IBM, 2005).

A ferramenta IBM Rational Software Architect, apesar de não ser uma ferramenta de modelagem criada para MDA, suporta os princípios e padrões do MDA. Principais recursos:

- Suporta modelagem em UML 2.0 com meta-dados (MOF) e OCL;
- Geração de código-fonte, com transformações a partir de modelos e *design patterns*;
- Suporta a criação de meta-modelos.

Como alternativa temos a ferramenta Rational Rose XDE Developer Plus, que utilizado juntamente com o Rational Rose, possibilita a integração com diversas ferramentas (inclusive de outros fabricantes), e gera código para diversas linguagens.

A IBM disponibilizou também um protótipo da linguagem QVT, chamado de MTF - *Model Transformation Framework*. É um *plug-in* que pode ser integrado ao Eclipse e a alguns dos seus produtos, possibilitando escrever, rodar, depurar e visualizar transformações.

3.2.4. Borland Together Architect 2006 for Eclipse

Esta ferramenta oferece todos os recursos necessários para um processo de software com MDA, pois suporta UML 2.0, a criação e a utilização de *profiles* UML, OCL 2.0, exportação e importação de modelos por XMI 2.0, recursos de produtividade para trabalho concorrente em equipe, um protótipo de QVT com possibilidade de depuração, entre outros recursos, disponibilizados de forma direta e produtiva (BORLAND, 2005).

Além desta versão, que é integrado ao Eclipse para a linguagem Java, existem também versões similares para outras ferramentas e linguagens.

3.2.5. ArcStyler 5.1

ArcStyler, da Interactive Objects, oferece uma solução completa para MDD com MDA para aplicações J2EE e .NET. Pode ser utilizada também para criar aplicações com a finalidade de modernizar sistemas legados (INTERACTIVE OBJECTS, 2005). Principais recursos:

- Possui todos os diagramas do UML 1.4;
- Possui XMI 1.1, MOF 1.4 e JMI 1.0;
- Possui recursos para colaboração em equipe, inclusive dispersas geograficamente;
- Inclui recursos para gerência de projeto;

- Inclui QVT;
- Utiliza o conceito de cartuchos para transformação PSM para código, e a ferramenta disponibiliza um bom número de transformações pré-definidas;
- Integra-se ao Borland CaliberRM e em ferramentas da IBM Rational.

3.2.6. OptimalJ 4.0

A ferramenta da Compuware, OptimalJ 4.0, oferece uma solução completa para J2EE. Principais recursos (COMPUWARE, 2005):

- Ferramenta completa especializada para J2EE, com possibilidade de geração de código para outras linguagens/tecnologias;
- Possui MOF, UML, CWM, XMI;
- As transformações utilizam a *Technology Patterns* (escritos em Java) para transformações do PIM para PSM e TPL - *Template Pattern Language*, para geração de código a partir do PSM;
- Abrange todos os modelos: PIM (*Domain Model*), PSM (*Application Model*) e código (*Code Model*);
- Possui *active synchronization* entre modelo e código, e tecnologia *Guarded/Free Blocks* que possibilita manter códigos “feitos à mão" integrados ao modelo para a re-geração da aplicação;
- Integra-se ao Borland JBuilder e Macromedia Dreamweaver (compartilha os códigos-fonte).

3.3. Comparação das Ferramentas

Provavelmente, a melhor forma de comparar as ferramentas apresentadas neste artigo é cruzar os recursos indispensáveis para um processo MDD com MDA. Para isso, foi elaborada uma tabela representando as seguintes variáveis:

- UML: Qual versão do UML a ferramenta suporta para modelagem?
- XMI: Qual a versão do padrão XMI que a ferramenta suporta para importação e exportação dos modelos?
- Transformações: Quais as linguagens são utilizadas para as transformações?
- Re-geração: A ferramenta oferece alguma tecnologia para possibilitar a re-geração do código sem destruir artefatos ou trechos de código introduzidos manualmente?
- Colaboração: A ferramenta oferece alguma forma de trabalho concorrente em equipe?

Note que esta tabela apresenta somente os recursos essenciais, indispensáveis em um processo MDD com MDA. Recursos como modelagem CWM, por exemplo, não são necessários para a construção de um software, apesar deste recurso ser bem-vindo

em uma ferramenta de modelagem, pois oferece artefatos específicos para a modelagem de um sistema de *Data Warehouse* ou de *Business Intelligence*.

Tabela 1: Comparação de Recursos das Ferramentas Pesquisadas

Ferramenta	UML	XMI	Transformações	Re-geração	Colaboração
JDeveloper	1.5	1.1	Java	Sim	Não
AndroMDA	-	1.1	Script, Java	Não	Não
IBM Rational	2.0	1.1	QVT, Java	Sim	Não
Together	2.0	2.0	QVT, Java	Não	Sim
ArcStyler	1.4	1.0	QVT, Java	Sim	Sim
OptimalJ	2.0	1.2	Java, TPL	Sim	Sim

Fonte: Elaborado pelo autor conforme a pesquisa elaborada.

Note que quanto maior a versão do UML e do XMI, maiores são os recursos disponibilizados pela ferramenta, permitindo assim criar e manter modelos mais sofisticados, que permitem contemplar melhores detalhes do sistema a ser gerado.

A escolha da ferramenta em uma empresa normalmente é influenciada por questões culturais e técnicas, como por exemplo, pela ferramenta de modelagem que a empresa utiliza, pelas tecnologias utilizadas em seus sistemas, e pela metodologia em que os profissionais estão acostumados a produzir software. Por este motivo, a tabela acima deve ser utilizada como ponto de partida para a escolha da ferramenta, e outros recursos deverão ser avaliados conforme as necessidades dos futuros projetos. Além disso, é recomendável avaliar outras ferramentas que não participaram desta pesquisa.

4. Conclusão

Apesar de o MDA ser um padrão recente, já existem ferramentas completas disponíveis para estabelecer um processo em cima desta abordagem. Basta conhecer esta arquitetura, conhecer os papéis executados pelos profissionais neste tipo de processo de construção de software, escolher a ferramenta, e capacitar os profissionais para que MDD com MDA aumente a produtividade durante o desenvolvimento e a manutenção de software.

5. Referências

OMG (2005) "**OMG Web Site**", Object Management Group, http://www.omg.org/, acesso em 13/11/2005.

SEARCHWEBSERVICES (2004) "**Research: Model-driven development increases productivity**", SearchWebServices.com, http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci953184,00.html, acesso em 13/11/2005.

MDDI (2005), "**Model Driven Development Integration. A Technology Project Proposal**", Eclipse, http://www.eclipse.org/proposals/eclipse-mddi/main.html, acesso em 13/11/2005.

KLEPPE, A., WARMER, J., BAST, W., (2003) "**MDA Explained: The Model Driven Architecture™: Practice and Promise**". Addison Wesley.

QVT Merge Group (2005), "**Revised submission for MOF 2.0 Query/View/Transformation RFP (ad/2002-04-10)**". QVT-Merge Group, version 2.1 ad/2005-07-01, http://www.omg.org/docs/ad/05-07-01.pdf, acesso em 13/11/2005.

WARMER, J., KLEPPE, A., (2004) "**Object Constraint Language, Getting Your Models Ready for MDA**", Second Edition, Addison Wesley.

SENDALL, S., KOZACZYNSKI, W., (2003) "**Model Transformation – the Heart and Soul of Model-Driven Software Development**". Swiss Federal Institute of Technology in Lausanne.

BORLAND (2005), "**Together**", Borland, http://www.borland.com/, acesso em 10/11/2005.

INTERACTIVE OBJECTS (2005) "**ArcStyler**". Interactive Objects, http://www.interactive-objects.com/, acesso em 11/11/2005.

COMPUWARE (2005), "**OptimalJ 4.0**", Compuware, http://www.compuware.com/, acesso em 11/11/2005.

RAMACKERS, Guus (2004). "**UML Modeling and MDA in Oracle JDeveloper 10g: Statement of Direction**". Oracle Corporation, http://www.oracle.com/technology/ products/jdev/collateral/papers/10g/uml_mda_sod.pdf, acesso em 13/11/2005.

ORACLE (2005), "**JDeveloper**", http://www.oracle.com/, acesso em 08/11/2005.

ANDROMDA (2005) "**AndroMDA**", http://www.andromda.org/, acesso em 08/11/2005.

IBM (2005) "**MDA Information Center**". IBM, http://www-306.ibm.com/software/ rational/mda/, acesso em 08/11/2005.