UMA ARQUITETURA BASEADA EM MODELOS - MDA

Hélder Pereira Borges[1,2], José Neuman de Souza[2], Bruno Schulze[3], and Antonio Roberto Mury[3]

[1]Federal Institute of Education, Science and Technology of Maranhão, São Luís, Brasil

[2]Department of Computing, Federal University of Ceará, Fortaleza, Brasil

[3]Department of Computing, National Laboratory for Scientific Computing, Petrópolis, Brasil

helderpb@hotmail.com, {bruno.schulze, neuman.souza, a.roberto.m}@gmail.com

RESUMO

O desenvolvimento de software tem encontrado grandes desafios que tem tornado este processo cada vez mais complexo, menos produtivo e mais dispendioso financeiramente. Por isto, a busca por abordagens que agreguem vantagens neste contexto é uma constante.

A necessidade de criação de softwares de forma rápida sem comprometer a qualidade a um custo menor intencionando-se a boa utilização dos recursos computacionais e novas tecnologias é uma grande preocupação da indústria de software. Sendo assim, a portabilidade, interoperabilidade, preparação para manutenções e documentação se apresentam como fatores cruciais na qualificação de um sistema.

A utilização de modelos, bem como a sua transformação em código ou outros modelos mais refinados são o cerne para propostas relacionadas ao desenvolvimento dirigido a modelos, incluindo MDA.

Alguns conceitos relacionados a utilização e transformação de modelos, dentro da proposta da MDA, são apresentados neste trabalho, dentre eles são vistos os conceito de modelos, meta-modelos, PIM, PSM, transformações e mapeamentos, além de uma descrição prática do funcionamento desta abordagem.

Palavras Chaves: MDA, Modelo Independente de Plataforma, Ferramentas MDA.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Este trabalho apresenta um estudo sobre desenvolvimento de software orientado a modelos, abordando diversos aspectos inerentes ao processo e objetivando o entendimento do funcionamento da abordagem MDA.

Neste capítulo serão apresentadas a justificativa e a motivação para o desenvolvimento deste trabalho, assim como os objetivos e as contribuições que se pretende alcançar sendo que ao final do capítulo, será descrito como está organizada o restante desta documento.

1.1 MOTIVAÇÃO

Um software para ser considerado bom e conseguir grande longevidade deve apresentar uma série de qualidades, dentre elas uma de suma importância está relacionada com a premissa sempre presente no mundo do desenvolvimento de software, as mudanças, e especificamente relaciona-se a adaptabilidade tanto em relação aos requisitos quanto ao ambiente em que a aplicação está inserida, principalmente quando considerado o contexto de produtividade.

Modelos elevam o nível de abstração do desenvolvimento de sistemas, ajudando no planejamento e entendimento dos mesmos, sendo que a importância do uso de modelos no desenvolvimento de software é um fato comprovado [16] [17].

Uma abordagem que cresceu bastante a partir dos anos 2000, é a da geração automática de código fonte a partir do modelo do sistema, e o desenvolvimento dirigido a modelos segue esta proposta, promovendo o modelo a artefato principal do desenvolvimento de software em detrimento do código.

A Arquitetura Dirigida a Modelos (Model Driven Architecture – MDA), especificada pelo OMG (Object Management Group) [3], é uma das iniciativas para esta abordagem que além de pregar o modelo como artefato principal possibilita a geração automática de código a partir dos modelos e ainda introduz o conceito de separação entre modelo e plataforma de suporte, agregando com isto independência da solução computacional em relação a tecnologia de implementação, o que

seguramente melhora a portabilidade, interoperabilidade e reusabilidade [12].

A proposta da MDA é promover o desenvolvimento de modelos que sejam independentes dos detalhes de implementação, criando-se, portanto, sistemas mais flexíveis e de fácil portabilidade, agregando ganhos imponentes em relação a qualidade do produto devido o foco dos desenvolvedores estar exclusivamente aplicado as regras de negócio, e em relação a produtividade, dado a automatização intrínseca a proposta da abordagem.

1.2 OBJETIVO E CONTRIBUIÇÃO

Este trabalho apresenta um estudo sobre desenvolvimento de software utilizando uma abordagem dirigida a modelos, onde o objetivo principal é compreender como funciona o processo proposto pela especificação MDA, identificando e relacionando os conceitos envolvidos na execução do mesmo e ainda comparando com o processo de desenvolvimento tradicional.

A principal contribuição deste trabalho é o levantamento do estado da arte em relação a Arquitetura Dirigida a Modelos e, além disto, a provisão de um catálogo de ferramentas atualmente disponíveis que se propõem a implementar este paradigma, bem como a demonstração de utilização de uma delas.

1.3 ESTRUTURA DA TRABALHO

Os próximos capítulos desta trabalho estão estruturados da seguinte forma:

- Capítulo 2: introduz uma contextualização do cenário atual em relação ao desenvolvimento de software, destacando a problemática dos seus desafios e apontando uma solução.
- Capítulo 3: apresenta um contexto histórico da solução apontada no capítulo 2, descrevendo sua origem e estruturas funcionais.
- Capítulo 4: aborda a MDA, descrevendo seus conceitos, princípios e funcionamento.
- Capítulo 5: descreve uma série de ferramentas que se propõem a

implementar MDA.

- Capítulo 6: apresenta um tutorial com o propósito de descrever, utilizando-se uma ferramenta de código aberto, os passos necessários para criar uma aplicação da MDA.
- Capítulo 7: apresenta as conclusões e possíveis trabalhos futuros.

2. A CONTEXTUALIZAÇÃO

Este capítulo relata os desafios relativos ao processo de desenvolvimento de software advindos de uma exigência cada vez mais exacerbada dos usuários dada a disponibilidade de maiores recursos computacionais e de diversas tecnologias emergentes.

2.1 O PROBLEMA E SEUS DESAFIOS

A Engenharia de Software, conforme [15], é proveniente da engenharia de sistemas e de hardware, sendo que ela abrange um conjunto de três elementos fundamentais, os métodos que proporcionam os detalhes de como fazer para construir um software, as ferramentas que proporcionam apoio automatizado ou semi-automatizado aos métodos e por fim, os procedimentos, que constituem os elos que mantém junto os métodos e as ferramentas possibilitando o desenvolvimento racional do software do computador.

Desta forma, possibilita-se ao gerente o controle do processo de desenvolvimento de software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente. E ainda segundo [1], que é considerado como a primeira proposta de definição, a Engenharia de software é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais.

Com o desenvolvimento da engenharia de software e a partir dos fundamentos erigidos por Pressman, [15], em relação ao desenvolvimento de sistemas, surgiram múltiplos métodos que agregam características de modelos distintos intentando a obtenção de melhores resultados, bem como interação entre modelos para proporcionar soluções mais complexas.

Assim sendo, genericamente, a proposta básica da área de informática é produzir sistemas de forma eficaz e eficiente utilizando-se das tecnologias disponíveis para agregar qualidade ao processo e, secundariamente, diminuir o

custo dos sistemas e aperfeiçoar as soluções.

O fato é que com o passar dos tempos os softwares evoluem de forma crescente em complexidade e os usuários se tornam cada vez mais exigentes em relação a muitos requisitos, e estas realidades nos levam a um cenário onde o desenvolvimento de software geralmente possui custo elevado e apresenta produtividade que normalmente pode ser definida como baixa, porém sendo justificada por vários fatores, como o fato do código possuir um nível muito mais baixo que o das abstrações utilizadas, além de normalmente ser dependente da plataforma de desenvolvimento, e ainda, o uso da reutilização de objetos com baixa granularidade, bem como as eventuais perdas dos mapeamentos de informações.

Com o passar do tempo, cada vez mais camadas de abstrações foram formalizas e novas ferramentas tiveram que ser construídas para suportar estes conceitos, em suma, os detalhes estavam cada vez mais sendo escondidos entre cada camada e as linguagens de alto nível ganhavam o potencial de serem mapeadas em linguagens de níveis inferiores, como pode-se observar na figura 2.1.

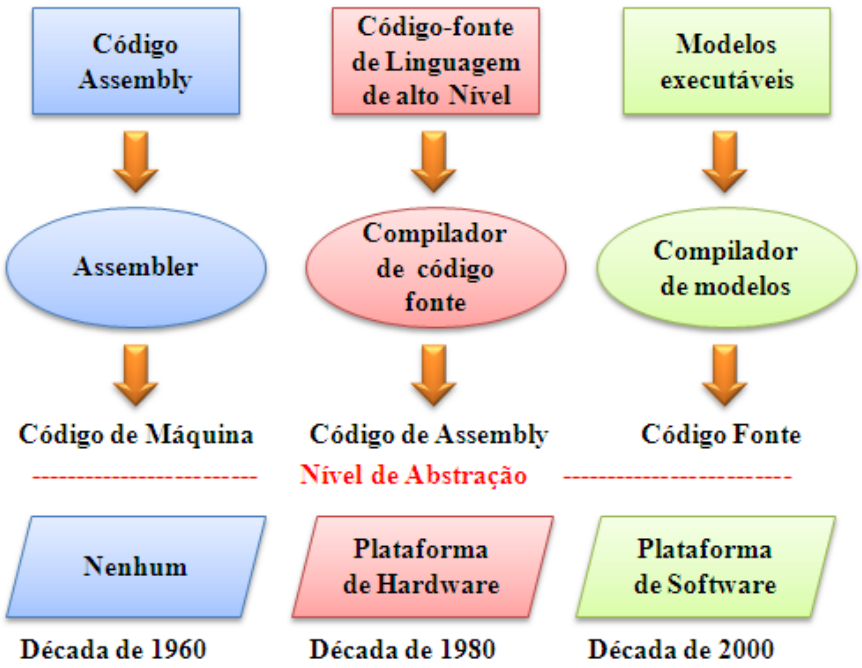

Figura 2.1 – Nível de abstração

Pode-se perceber a partir da figura 2.1que ocorre um aumento substancial do

nível de abstração ao longo dos anos e por sua vez da complexidade dos sistemas, sendo que nos primórdios da década de 60 não havia abstração, os sistemas eram desenvolvidos diretamente em linguagem de máquina. Na década de 80 já era possível ocultar características do hardware físico, sendo disponibilizadas plataformas eficientes para execução de aplicações sem haver uma preocupação em como utilizar de forma específica cada hardware disponível. A partir dos anos 2000, vivenciou-se a possibilidade de compatibilização entre diferentes plataformas de forma simplificada.

O nível de reuso também foi alterado de forma significativa no decorrer dos anos como se observa na figura 2.2.

Diferentes implementações de um mesmo algoritmo exigiam um controle mais complexo de correções, pois era necessário identificar se os erros eram no algoritmo ou estavam relacionados com uma implementação específica, além do árduo trabalho de atualização, quando necessária, ser feita em todas as implementações.

Um avanço nos anos 70 foi a utilização de funções, que poderiam ser reaproveitadas sem a necessidade de duplicação do código. Nos anos 80, foram desenvolvidas as linguagens orientadas a objetos trazendo um novo potencial de agrupamento e organização de funcionalidades, tornado mais fácil o reuso de artefatos já produzidos.

A utilização de frameworks, que disponibilizam um conjunto de funcionalidades e ainda o uso de componentes, que são artefatos auto-contidos, marcou a década de 90. Já a partir dos anos 2000, era possível construir aplicações reutilizando características comuns entre domínios utilizando-se conceitos e funcionalidades compartilhadas.

Pode-se perceber que o processo de reutilização foi amplamente desenvolvido, aumentando-se os ganhos inerentes ao seu uso como o aumento de produtividade e utilização de artefatos já validados em outros domínios ou sistemas.

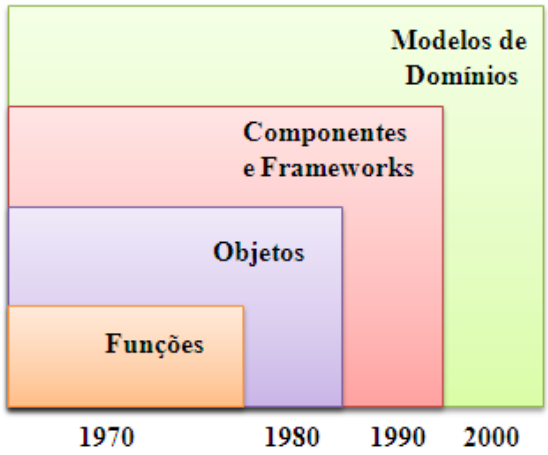

Figura 2.2 – Nível de reuso

Desta forma avanços portentosos podem ser observados, porém dentro deste cenário, podem ser destacados alguns grandes desafios apresentados para o desenvolvimento de software atualmente:

- A crescente complexidade das regras de negócio;
- Desenvolvimento rápido;
- Manter a viabilidade relacionada a qualidade, longevidade e custo pré-estabelecido, visto que os projetos de software se tornaram muito dispendiosos;
- Compatibilidade entre diferentes plataformas;
- Adaptação a novas tecnologias emergentes;
- Comunicação e consistência entre sistemas, legados ou não.
- A mudança freqüente de requisitos, indispensável para satisfação e completo atendimento das necessidades dos usuários;
- Retorno satisfatório do investimento em desenvolvimento.

2.2 UMA SOLUÇÃO PARA O PROBLEMA

Em resumo, pôde-se perceber a existência de um grande desafio relacionado a complexidade no desenvolvimento, manutenção e evolução de software, sendo que

os paradigmas de orientação à objetos e componentes já não conseguem gerenciar eficientemente esta complexidade e uma abordagem centrada em middlewares também não é suficiente, tornando-se necessária uma mudança de paradigma.

A Arquitetura Orientada a Modelos – MDA desponta como uma solução emergente para os problemas inerentes ao desenvolvimento de software, isto porque possui uma abordagem centrada no conceito de modelos, no relacionamento entre eles e nas transformações entre estes modelos. O seu principal objetivo é isolar a lógica de negócio da aplicação, da evolução e manutenção da tecnologia, e apresenta como proposta a construção de sistemas de forma rápida, consistente e independente de plataforma.

O desenvolvimento será rápido porque a maior parte do código será gerado a partir das especificações de correspondências e transformações entre modelos, sendo que novos modelos compatíveis com uma plataforma específica são obtidos sem a necessidade de um novo processo de desenvolvimento. Automaticamente também são geradas as conexões para comunicação entre os diversos modelos envolvidos, sendo então o código gerado, que também por ser automático, está menos propenso a erros humanos, garantindo-se desta forma uma maior consistência.

Exemplificando de forma concreta, um documento hipermídia feito para um middleware específico não poderá ser interpretado por um middleware de outros padrões, sendo necessária uma tradução manual entre os diferentes padrões, o que obviamente aumentará o esforço em sua autoria, desperdiçando, portanto, tempo e recursos. Para resolver este problema, [18], utilizou MDA para automatizar o processo de tradução de um documento hipermídia em diferentes padrões, sendo que a transformação pode ser feita em um nível mais abstrato denominado modelo.

2.3 CONCLUSÃO

Este capítulo fez uma descrição do problema no âmbito do desenvolvimento de software e apontou a MDA como uma solução adequada.

3. A ORIGEM DA MDA

Este capítulo tem o propósito de descrever o cenário de criação da MDA, bem como as tecnologias que dão suporte ao seu funcionamento, retratando o cenário no qual esta abordagem foi concebida.

3.1 INTRODUÇÃO

O desenvolvimento dirigido a modelos não era uma abordagem nova para o desenvolvimento de software, os especialista em software já se utilizavam de modelos para capturar a arquitetura e definições de sistemas, porém o conceito de MDA definido pelo OMG (Object Management Group) foi uma novidade no mundo da orientação a modelos.

O OMG é um consórcio de empresas, dentre elas (Adobe Systems Inc, AT&T, Boeing, Borland Software Corporation, Ericsson, GNOME Foundation, Hewlett-Packard, Massachusetts Institute of Technology (MIT), Motorola, NASA, Nokia, Oracle, Sun Microsystems, W3 Consortium), organizações e pessoas, ou seja, é amplamente aceito, difundido e organizado, que tem como objetivo a definição de padrões de processos relacionados ao desenvolvimento de software, com a intenção de solucionar problemas de integração de aplicações distribuídas através do fornecimento de especificações de interoperabilidade abertas e independentes de fornecedor, facilitando o reuso de componentes e garantindo a portabilidade.

A situação muito comum, a necessidade de correção em aplicações que estão em uso, bem como o imperativo de integração com outros sistemas, além de alterações na infra-estrutura e nos requisitos, bem como a evolução e criação de novas tecnologias, levou a OMG a criar um padrão de especificação onde os detalhes da implementação e da plataforma foram abstraídos, sendo o foco direcionado apenas para a modelagem das regras de negócio, então em 2001 foi criada a MDA.

Segundo o OMG [2], MDA define uma abordagem para especificação em sistemas computacionais que separa a especificação das funcionalidades do

sistema da implementação destas funcionalidades em uma plataforma tecnológica específica, sendo então definido uma arquitetura baseada em modelos que fornece um conjunto de diretrizes para estruturar especificações expressas como modelos permitindo que um mesmo modelo com especificações de funcionalidades seja utilizado em múltiplas plataformas através de mapeamentos auxiliares ou com pontos de mapeamentos para plataformas específicas, permitindo com isto que aplicações distintas sejam integradas baseado na relação explícita de seus modelos. Em [3], a portabilidade, a interoperabilidade e o reuso a partir da separação arquitetural de interesses, são apontados como as três metas primárias da MDA.

3.2 AS TECNOLOGIAS ENVOLVIDAS

O OMG criou um modelo de objeto padrão para definir o comportamento dos objetos em um ambiente distribuído, o Object Management Architecture (OMA), que utiliza o Common Object Request Broker (CORBA), [5], como componente de comunicação, sendo esta a arquitetura padrão do OMG para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos, onde os objetos ou componentes de software se comunicam de forma transparente ao usuário, apesar da necessidade de interoperar com outras aplicações desenvolvidas com ferramentas distintas e ainda em outros sistemas operacionais.

O objetivo da interoperabilidade é alcançado pelo CORBA através da definição de um ORB (Object Request Broker), que em computação distribuída é uma peça do middleware que permite que os desenvolvedores façam chamadas de um computador a outro através de uma rede, sendo que o mesmo é o um componente chave do OMA sendo a base para construção de aplicações que utilizam objetos distribuídos interoperáveis, habilitando-os a enviar e receber requisições, bem como receber as respostas de suas requisições, ou seja, trafegar os dados e requisições dos objetos do ambiente local para o remoto e vice versa.

A proposta básica do padrão CORBA é eliminar os problemas típicos em sistemas heterogêneos como a dependência de aspectos incompatíveis entre plataformas, provendo a independência de linguagem, de implementação, de arquitetura, de sistema operacional e também de protocolo / transporte dado que o

ORB decide dinamicamente que protocolo e transporte devem ser usados.

O OMA [4] incorpora a visão do OMG para o ambiente de componentes de software fornecendo instruções de como padronizar componentes de interfaces baseado na tecnologia de objetos e define a arquitetura sob a qual os objetos locais e remotos devem se comunicar.

Embora aplicações possam ter regras de negócio totalmente diferentes é comum elas compartilharem várias funcionalidades, como um objeto precisar notificar outro quando um evento ocorrer ou a referência a objetos novos ou destruídos serem propagados e, além disto, aplicações dentro de um mesmo domínio de negócios normalmente compartilham uma série de funcionalidades.

O OMA abstrai as funcionalidades comuns de aplicações CORBA em um conjunto de objetos padrão com funções claramente definidas que podem ser acessadas através de interfaces padronizadas pelo OMG que especificam o que o objeto faz, podendo então serem criadas implementações para serviços específicos. Com a utilização do OMA espera-se alcançar uma codificação e implantação mais rápida, uma melhor arquitetura devido o projeto ser desenvolvido em torno de serviços discretos e considerar a divisão em grupos de objetos baseado na funcionalidade e ainda robustez e escalabilidade.

A estrutura de um OMA é composta basicamente por quatro elementos, sendo que a maioria dos **Objetos de Serviços (CORBA Services)** oferecem funcionalidades básicas para aplicações com objetos distribuídos onde é executado um serviço que chama bibliotecas de um determinado sistema operacional, embora existam serviços mais complexos, como a segurança, que devido a estreita ligação com a infra-estrutura ORB exigem sua colocação nesta categoria.

Exemplos de CORBA Services são o serviço de nomes, o serviço Object Trader, o serviço de persistência de estado, o serviço de transações, os serviços para gerenciamento do ciclo de vida de objetos remotos e segurança.

Os **objetos de Aplicação** normalmente são customizados para cada aplicação individual e não necessitam de padronização, são os objetos que podem ser considerados visíveis ao nível de aplicação.

Por fim, as Facilidades Comuns que são dedicadas a usuários finais de aplicações e podem ser divididas em **Facilidades CORBA Horizontais**, são potencialmente utilizadas em vários domínios de negócios, como serviços de e-mail, impressão, interface de usuário, etc.; e **Facilidades CORBA Verticais** que são específicas para um domínio de aplicação onde empresas de um mesmo segmento podem compartilhar objetos.

Após a criação do padrão de interoperabilidade CORBA, a OMG o utilizou quase que exclusivamente na criação de padrões para domínios específicos, porém a partir de 1997 foram apresentadas especificações que não eram baseadas em CORBA.

Dentre elas a Unified Modeling Language (UML), [6], que surgiu para resolver o problema da falta de padronização entre as notações usadas no desenvolvimento de software, sendo portanto uma especificação que define uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema de forma que os relacionamentos entre os componentes sejam melhor compreendidos e visualizados e tendo como objetivo principal descrever qualquer tipo de sistema em termos de diagrama

Outra especificação, o Meta Object Facility (MOF) foi concebido como uma iniciativa da OMG para a padronização da representação e manipulação de meta-modelos, que é um modelo de uma linguagem de modelagem [7][14], através de uma linguagem abstrata para especificação, construção e gestão dos meta-modelos independente da tecnologia de implementação.

Posteriormente, o XML Metadata Interchange (XMI) [8] é definido como padrão para a troca de informações de metadados via Extensible Markup Language (XML) com o propósito de facilitar o intercâmbio entre ferramentas de modelagem UML e repositórios baseados em MOF em ambientes heterogêneos e distribuídos fornecendo um mapeamento entre MOF e UML.

Ainda foi definido o Common Warehouse Metamodel (CWM) [9], que é um padrão de metadados cujo objetivo é permitir a integração de sistemas de data warehouse, e-business e sistemas de negócios inteligentes em ambientes heterogêneos e distribuídos, através de uma representação e de um formato de

troca de metadados.

3.3 A EVOLUÇÃO

As especificações descritas na seção anterior foram largamente implementadas pelos fornecedores ao longo dos anos seguintes, agregando aos sistemas desenvolvidos com componentes que suportavam as padronizações já desenvolvidas pelo OMG, grandes facilidades em tarefas como a criação de soluções multi-fornecedor e na integração entre aplicativos, gerando com isto boas expectativas em relação a outras especificações.

Até o momento em que o OMG produziu apenas padrões baseados em CORBA a forma de ligação entre os padrões era entendida e mapeada pelo OMA sem grandes problemas, porém com o surgimento acelerado e de certo modo desordenado de novos padrões, foi necessário expandir a visão da arquitetura OMG para dar maior suporte as tecnologias emergentes.

Considerando-se que a vida de um sistema computacional normalmente é longa e nem sempre é possível ou simplório fazer a modificação de sistemas mais antigos com a intenção de que eles venham suportar um ou vários padrões e ainda a crescente necessidade de incorporação de front-ends baseados na web desenvolvidos com interfaces proprietárias, forçam os desenvolvedores / integradores a retomarem atividades pouco produtivas, porém essenciais, para manter os diversos componentes funcionando junto, e ainda é importante destacar a enorme quantidade de trabalho necessária para realizar a modificação e integração destes sistemas visando a utilização de tecnologias emergentes, portanto, existe um limite de interoperabilidade que pode ser alcançado com a criação de um conjunto de interfaces padrão para o desenvolvimento de sistemas.

A partir desta constatação, o OMG expandiu sua visão concernente aos requisitos para suportar a interoperabilidade com especificações para integração durante todo o ciclo de vida dos sistemas, da modelagem do negócio ao projeto do sistema, da construção do componente até a montagem, integração, desenvolvimento, gerenciamento e evolução [2], sendo esta nova visão incorporada ao desenvolvimento da MDA. A mesma descreve as relações entre os padrões OMG

e como usá-los coordenadamente e ainda como esta abordagem auxilia na criação, manutenção e evolução dos padrões.

Desta forma, MDA foi uma proposta para expandir o OMA e não para substituí-lo, incorporando o trabalho feito até então e apontando o caminho para os futuros padrões de integração, oferecendo mais suporte para uma criação rápida e eficaz de novas especificações que integram múltiplos padrões dentro e fora da organização.

A figura 3.1 ilustra a arquitetura da MDA, revelando como os padrões OMG trabalham com MDA e fornecendo uma visão geral onde vários padrões podem ser identificados, além de se perceber três camadas de especificações, onde o núcleo é formado por especificações que seguem os padrões definidos pelo OMG (UML, MOF e CWM) e não levam em consideração características específicas de uma plataforma, sendo totalmente independentes.

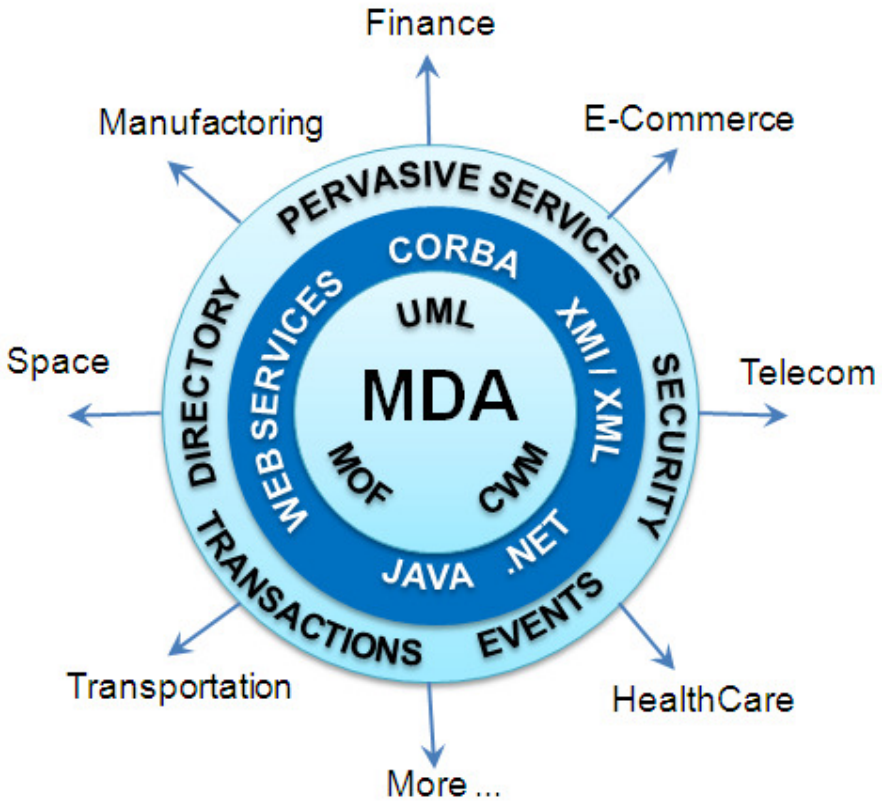

Figura 3.1 - Arquitetura MDA

Na segunda camada encontram-se modelos que já possuem detalhes de uma

determinada tecnologia, como Java, *.NET*, CORBA, *Web* etc. Na camada mais externa, são exibidos os serviços que a maioria dos domínios de aplicações necessitam, tais como segurança, persistência, controle de transações, tratamentos de eventos etc. Por fim, observa-se a aplicação da MDA em diferentes domínios.

3.4 CONCLUSÃO

Este capítulo descreveu a origem da MDA, destacando o grupo responsável pela especificação, os fundamentos relacionados com a proposta do processo e o cerne tecnológico que envolve a abordagem, apresentando suas características e finalidades.

4. ARQUITETURA DIRIGIDA A MODELOS - MDA

Este capítulo descreve a arquitetura dirigida a modelos, retratando seus conceitos fundamentais, seu ciclo de vida além de uma gama de ferramentas. Também é discutido como ficam os papéis de cada profissional envolvido no processo de desenvolvimento de sistemas.

4.1 INTRODUÇÃO

A MDA consiste de uma abordagem para especificação de sistemas que separa a especificação das funcionalidades ou lógica de negócio da implementação em uma plataforma específica.

Para alcançar este propósito, foi definida uma arquitetura para modelos que fornece um conjunto de diretrizes para estruturação das especificações, onde a abordagem e os padrões que lhe dão suporte permitem que um mesmo modelo de especificação seja utilizado para múltiplas plataformas através de um mapeamento padrão ou de pontos de mapeamento para plataformas específicas, permitindo então que aplicações diferentes sejam integradas por meio de uma relação explícita entre seus modelos, habilitando desta forma a interoperabilidade e a evolução dos sistemas de suporte conforme as tecnologias atuais.

O fato de ser orientada a modelos significa que os artefatos principais do processo de desenvolvimento são os modelos, e eles devem orientar o entendimento, o projeto, a construção, a operação, a manutenção e as modificações do sistema [10]. Os modelos podem ser utilizados para gerar os sistemas, scripts de bancos de dados, documentação para o usuário, configurações e quaisquer outros elementos que possam fazer parte do processo de desenvolvimento.

O OMG, conforme [3], define formalmente a MDA como uma abordagem que é alavancada pela idéia de separar a especificação das operações de um sistema dos detalhes de como este sistema usa as potencialidades de sua plataforma, possibilitando que ferramentas ofereçam a especificação de um sistema independente de qualquer plataforma, além da especificação de plataformas bem como a transformação da especificação para uma plataforma particular, tendo, portanto como objetivo principal a portabilidade, a interoperabilidade e a

reusabilidade através da separação arquitetural dos interesses.

4.2 CONCEITOS BÁSICOS

Existem muitas expressões que flutuam no ambiente da MDA e a proposta desta seção é fornecer o entendimento contextualizado do significado de alguns termos que estão diretamente envolvidos no cotidiano da MDA.

4.2.1 Modelo

É um conjunto de elementos que descreve um sistema [11] com grau de abstração maior que o do sistema [12]. Também pode ser definido como uma especificação formal de uma função, estrutura e/ou comportamento de um sistema [2].

Modelos são representações fidedignas, porém simplificadas e contextualizadas, de algo real, por isto, bons modelos servem como meio de comunicação.

Os modelos freqüentemente apresentam uma combinação de desenhos e texto sendo normalmente representados por linguagens de modelagem, como a UML, que é a linguagem recomendada pelo OMG para trabalhar com MDA.

4.2.2 Meta-Modelo e Meta-Meta-Modelo

Um meta-modelo é um modelo de uma linguagem de modelagem que define a estrutura, semântica e restrições para um conjunto de modelos que compartilham sintaxe e semântica comuns [11]. Por exemplo, um modelo que utiliza diagramas UML está sujeito a um meta-modelo UML, que descreve como modelos UML podem ser estruturados e que elementos contêm.

O meta-modelo da UML é uma instância da infra-estrutura do MOF [13], sendo este um framework para gerenciar meta-dados e serviços de meta-dados intencionando o desenvolvimento e interoperabilidade de sistemas baseados em modelo, ou seja, o MOF objetiva a criação de meta-modelo, sendo chamado de Meta-Meta-Modelo.

4.2.3 Plataforma

É um conjunto de subsistemas e tecnologias que provêem um conjugado de funções coerentes que podem ser utilizadas a partir de interfaces e padrões específicos onde

qualquer aplicação suportada pela plataforma pode utilizar sem se preocupar em como a funcionalidade foi implementada.

Pode-se entender plataforma também como a especificação de um ambiente de execução para um conjunto de modelos [11].

A independência de plataforma é uma qualidade que um modelo pode apresentar e representa que o modelo não depende de qualquer característica específica da plataforma.

4.2.4 Mapeamento

Dado que modelos podem possuir relacionamentos semânticos com outros modelos, os mapeamentos servem para conectar elementos de um modelo fonte a elementos de um modelo alvo que possuem mesma estrutura e semântica.

Desta forma, o relacionamento um para um descreve um elemento do modelo alvo possui a mesma semântica de um elemento do modelo fonte; já no caso de um para muitos, representa que um conjunto de elementos do modelo alvo possui a mesma semântica de um elemento do modelo fonte; e por fim, a relação de muitos para um significa que um elemento do modelo alvo possui a mesma semântica de um conjunto de elementos do modelo fonte.

Em síntese, um mapeamento provê especificações para que um modelo seja transformado em outro.

4.3 VISÃO GERAL DO PROCESSO DE DESENVOLVIMENTO MDA

A arquitetura MDA define modelos que são básicos para que a sua estrutura funcione corretamente, sendo então possível alcançar os objetivos propostos.

Uma sequência de atividades foi definida para o desenvolvimento de softwares, todavia estas atividades são divididas em etapas, iniciando-se com o levantamento dos requisitos do sistema para então, baseado nestes requisitos, criar-se um Modelo Independente de Computação.

4.3.1 Modelo Independente de Computação - CIM

Este modelo, o Modelo Independente de Computação (Computation Independent Model – CIM), descreve o domínio da aplicação e não possui nenhum detalhe sobre

a estrutura e processamento do sistema, sendo um modelo conceitual ou de análise, que identifica as entidades e que pode listar suas propriedades e seus relacionamentos, sem, contudo especificar a interdependência entre estas propriedades, sua execução interna e suas interações.

O CIM exibe o sistema no ambiente que ele vai operar e, portanto ajuda a comunicar exatamente o que o sistema deve fazer sendo que os requisitos representados por ele devem ser mapeados para os modelos seguintes da arquitetura.

4.3.2 Modelo Independente de Plataforma - PIM

O segundo modelo da arquitetura MDA, segundo o OMG [2], é o Modelo Independente de Plataforma (Plataform Independent Model – PIM) que deve ser gerado a partir do CIM, embora algumas publicações considerem o PIM como o início de um processo MDA, talvez porque normalmente o processo de geração do PIM a partir do CIM seja manual.

O foco deste modelo está em expressar as funcionalidades de negócio e comportamento, escondendo detalhes de plataforma, para exibir apenas a especificação do que não varia com a alteração da plataforma. O propósito desta independência é possibilitar o uso do mesmo PIM em múltiplas plataformas.

Os serviços pervasivos da arquitetura MDA podem ser incorporados ao PIM, e o mesmo pode ser refinado quantas vezes forem necessárias para se chegar ao nível de precisão adequado.

Uma transformação deve ser aplicada ao PIM para que o próximo modelo da arquitetura seja gerado. Esta transformação é o processo de convergir de um modelo para outro seguindo um determinado mapeamento que descreve uma correspondência entre os elementos dos modelos, ou seja, a partir de cada elemento de um modelo fonte a transformação gera os elementos correspondentes no modelo alvo, sendo definida por regras de transformação dentro de ferramentas de transformação.

4.3.3 Modelo Específico de Plataforma - PSM

O modelo advindo do PIM é o Modelo Especifico de Plataforma (Plataform Specific Model – PSM) que apresenta uma visão focada em uma plataforma, combinando as

especificações presentes no PIM com os detalhes relativos ao funcionamento do sistema em uma plataforma específica.

Obviamente se faz necessário a definição de qual ou quais plataformas serão utilizadas na transformação, para então serem construídos os mapeamentos necessários para cada plataforma, sendo para tanto necessário o uso de um modelo de plataforma que provê um conjunto de conceitos que representam as diferentes partes da plataforma, os serviços providos por ela e os diferentes tipos de elementos que podem ser utilizados.

Um exemplo de transformação de PIM para PSM pode ser visto na figura 4.1.

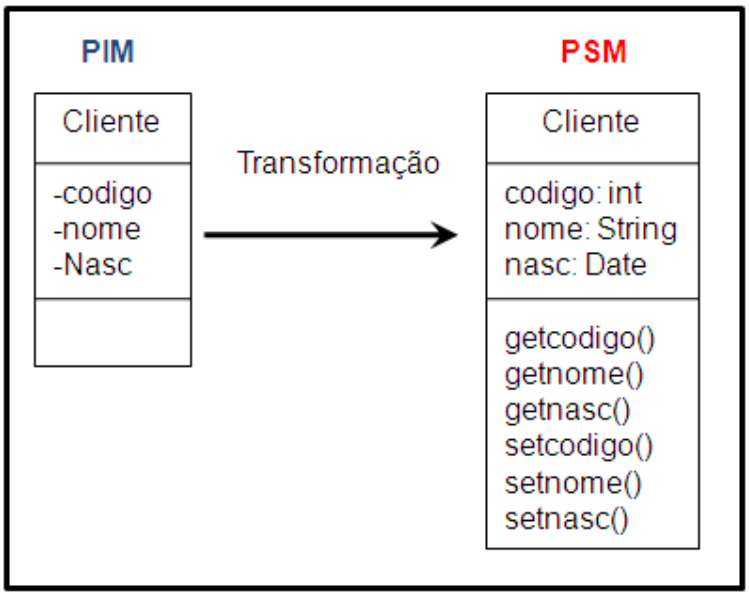

Figura 4.1 – Transformação PIM > PSM

A MDA possibilita que as ferramentas de transformação alterem o PIM em um PSM ou que seja gerado diretamente o código da aplicação além de que o PSM pode ser refinado em outro PSM ou usado para gerar o código do sistema.

Como resultado da transformação do PIM espera-se além do PSM propriamente dito ou do código, o registro de transformação, que armazena o mapeamento dos elementos do PIM aos elementos correspondentes no PSM descrevendo que parte do mapeamento foi usado por qual parte da transformação, possibilitando desta forma o rastreamento da transformação e possibilitando que a mesma seja bidirecional.

A figura 4.2 retrata o processo completo de transformação que ocorre durante

o uso da MDA.

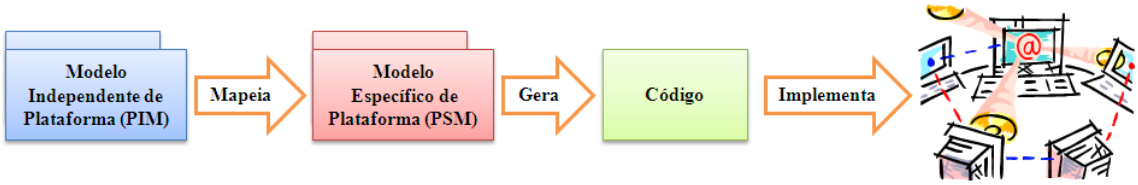

Figura 4.2 – Processo de Transformação MDA

4.4 O CICLO DE VIDA DA MDA

As mesmas fases de desenvolvimento podem ser identificadas tanto nos modelos de desenvolvimento de sistemas tradicionais quanto no modelo da MDA, sendo que as diferenças essenciais se encontram na natureza dos artefatos que são criados durante cada processo de desenvolvimento, sendo estes os modelos formais compreendidos pelo computador.

As imagens 4.3 e 4.4 respectivamente exibem o ciclo de vida tradicional e o ciclo de vida do desenvolvimento com MDA.

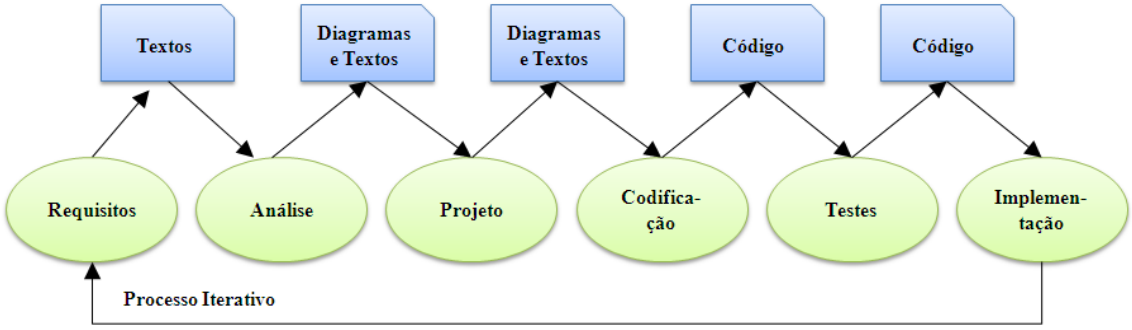

Figura 4.3 – Ciclo de Vida Tradicional

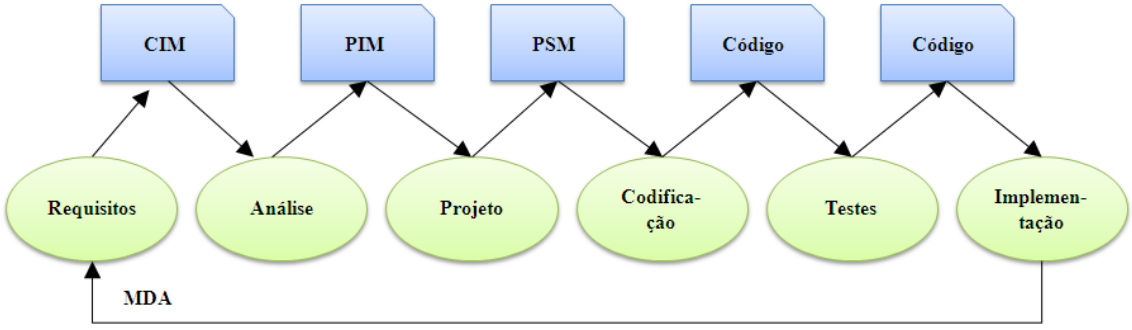

Figura 4.4 – Ciclo de Vida MDA

Os diagramas e textos da figura 4.3 são especificados de acordo com uma arquitetura, logo inovações tecnológicas ou mesmo alterações nas tecnologias podem introduzir incompatibilidades no processo. No caso da figura 4.4, como os artefatos iniciais são independentes de plataforma, a portabilidade é extremamente facilitada garantindo-se um aumento de produtividade dado a utilização da transformação entre os modelos.

4.4.1 Comparação de atividades

Considerando o ciclo de vida da MDA, podemos destacar e comparar o que cada profissional envolvido no processo de desenvolvimento de software faz no modelo tradicional e no modelo MDA.

O **Analista de Requisitos** é o responsável por capturar os resultados da análise de requisitos em modelos UML para futura verificação e testes dos modelos gerados no processo de desenvolvimento e ainda reduzir as possibilidades de má interpretação dos resultados da análise de requisitos.

Este papel no processo MDA não muda em relação ao tradicional, é exatamente igual. Apesar dos modelos UML com seus diagramas ajudarem na comunicação e visualização dos requisitos, como esta atividade é baseada em pessoas a sua criação não pode ser automatizada.

O **Analista** / **Projetista** possui a função de expressar formalmente os requisitos, em termos de diagramas de classe, de estado e métodos é a responsabilidade deste profissional.

Como estes modelos serão utilizados para transformação em outros modelos ou código a precisão dos mesmos é fundamental de maneira a reduzir a necessidade de modificações para ajuste.

Se forem utilizadas ferramentas com suporte a UML executável, pode-se facilmente demonstrar o comportamento dos modelos aos clientes e consequentemente se obter um retorno mais rápido, agregando uma melhor precisão ao modelo.

É responsabilidade do **Arquiteto** tomar decisões sobre a estrutura geral do sistema, em um processo MDA, continuam a fazê-lo, e esta tarefa envolve selecionar os modelos e mapeamentos entre eles.

O mercado de desenvolvimento de software já disponibiliza uma série de modelos e mapeamentos para reutilização, sendo necessário que os arquitetos apliquem sua experiência e conhecimento no processo de seleção dos modelos e sintonização dos mapeamentos para melhorar o desempenho do sistema.

Os **Analistas** / **Programadores** continuam a desempenhar seu papel praticamente da mesma maneira, porém existem duas grandes diferenças, a primeira está relacionada com a linguagem usada para expressar as abstrações, ao invés de Java, C++, etc deverá ser definida pelo QVT (Query, Views and Transforms) que é a especificação padrão do OMG para frameworks de regras de transformação formal.

A segunda diferença considera que as abstrações criadas pelos desenvolvedores devem exercer grande influencia no sistema, logo, ao invés de criar uma lista para um conjunto de contas, eles devem criar regras para construção de uma lista para qualquer classe que possua o mesmo padrão de acesso que uma conta.

Desta forma, o trabalho fica relacionado com a criação e ajuste dos padrões de geração de código utilizado por ferramentas MDA, gerando-se então, devido a perícia do programador, melhores resultados do que simplesmente desenvolver código manualmente.

De forma geral, as ferramentas MDA menos avançadas podem gerar pelo menos 50% de todo o sistema, embora ferramentas mais sofisticadas possam chegar até 100% do sistema modelado em determinadas circunstâncias, o fato é que trabalhar com MDA dá mais tempo aos programadores para trabalhar sobre as partes mais interessantes da lógica que o código gerador não conseguiu lidar.

A utilização de ferramentas para geração de scripts de testes a partir dos modelos torna os **responsáveis pelos testes** muito mais produtivos, além de não apresentar nenhuma incompatibilidade com a escrita manual de scripts de testes quando isto se fizer necessário.

A idéia durante os testes é simular todos os caminhos de execução da aplicação, com este propósito, um script de teste precisa combinar pequenos e diferentes conjuntos de parâmetros de entrada em um grande número de combinações, para na sequência poder comparar a saída da aplicação com os

resultados esperados.

Com uma ferramenta para geração automática dos scripts de teste, de forma rápida, o testador pode produzir um grande número de testes individuais que são necessários para o teste exaustivo de uma aplicação, isto sem o ônus de tempo e tédio de escrever e reescrever fragmentos de código ligeiramente diferente necessário no processo manual.

Os custos de manutenção representam mais da metade do custo total no desenvolvimento de qualquer aplicação de longa duração, isto normalmente se dá devido o serviço de detetive que os **responsáveis pela manutenção** devem fazer para entender o comportamento da aplicação quando não existe documentação ou ela está desatualizada ou atrasada.

Com o advento da MDA que utiliza um projeto preciso para construir uma aplicação, esta atividade se torna muito mais produtiva, porque quando se faz necessário uma alteração, este procedimento é realizado nos modelos e regras do projeto e não diretamente no código e no caso de modificações no comportamento da aplicação, é o PIM que será modificado.

Se a plataforma de destino for modificada, os mapeamentos são alterados ou se for o caso, totalmente substituídos por outros mapeamentos para uma plataforma distinta. O resultado disto é uma vida muito mais longa para a aplicação e um trabalho muito menos oneroso para os responsáveis pela manutenção.

4.5 FERRAMENTAS PARA MDA

O objetivo desta seção é a identificação de iniciativas disponíveis atualmente com o propósito de dar suporte ao desenvolvimento dirigido a modelos, sendo as descrições das ferramentas aqui relatadas adquiridas junto aos fornecedores das mesmas, pretendendo-se então, de forma sucinta, prover um breve relato de suas propostas e funcionalidades.

A perspectiva gerada com o surgimento da MDA levou ao surgimento de várias iniciativas propondo sua implementação, porém a geração automática de código a partir de modelos não é suficiente para qualificar uma ferramenta como sendo uma ferramenta MDA, porque se faz necessário a corretude em relação aos conceitos propostos pela abordagem e para tanto somente uma análise mais

profunda identificará se existe correspondência entre os preceitos da MDA e as implementações presentes nas ferramentas, o que não foi alvo deste trabalho.

4.5.1 AndroMDA

Segundo [19], é um framework de código aberto para geração de código fonte para plataforma Java a partir de modelos, proporcionando com isto que a atenção dos desenvolvedores esteja voltada apenas para as regras de negócio do sistema. Ela é dividida entre núcleo e plugins que são denominados de cartuchos e que contêm o mapeamento para as plataformas, e são adicionados ao núcleo para realizar as transformações.

Utiliza UML como linguagem de modelagem e utiliza apenas um modelo que é usado como entrada para as transformações e geração direta do código. Este modelo é o PIM do sistema e não pode conter tipos de dados específicos de qualquer plataforma. Para garantir a correção dos tipos de dados a ferramenta desenvolveu um perfil UML com a descrição de tipos de dados genéricos e estereótipos que deve ser adicionado ao modelo para ser utilizado com todos os seus cartuchos.

Além da utilização de tipos de dados corretos, os elementos dos modelos devem ser marcados com estereótipos para que sejam transformados da maneira adequada.

4.5.2 Blu Age

Conforme [20], é um gerador de aplicação integrado ao Eclipse que oferece um ambiente integrado para execução de modelos UML e transformação automática para aplicações de negócios para os ambientes Java e .NET.

Os diagramas UML são instantaneamente transformados para representar os processos de negócio, especificações funcionais e as regras do negócio. A proposta é para desenvolvimento centrado em modelos onde é possível capturar a concepção e cartografa do processo de negócio armazenado durante a análise de definição da aplicação para ser independente de plataformas, desta forma garantindo a durabilidade dos modelos conceituais e a descrição do sistema global.

A ferramenta BlueAge, oferece três módulos: o primeiro é focado no desenvolvimento dos modelos UML e se denomina Build; o modulo Deliver foi

desenvolvido para configurar os ambientes para geração automática de código-fonte de acordo com os frameworks padrões da empresa.

Estes dois módulos são disponibilizados como plugins para integração com a plataforma Eclipse, agregando uma resposta eficiente no desenvolvimento de aplicações desde a concepção até a implementação com uma arquitetura bastante flexível que permite a geração automática de vários componentes de software como entidades de negócio, serviços web, interfaces gráficas, serviços de integração e outros; além destes dois módulos de transformação é oferecido a possibilidade de utilização de engenharia reversa através do módulo Reverse Modeling, que permite a criação automática de modelos UML a partir de código-fonte existente, de aplicações java ou não, com a intenção de facilitar uma re-engenharia utilizando-se os modulos Build e Deliver.

Várias ferramentas de modelagem podem ser utilizadas para criar os modelos UML e então podem ser exportadas para dentro do BluAge através da transformação utilizando XML Metadata Interchange (XMI) ou Eclipse Modeling Framework (EMF).

4.5.3 OpenArchitectureWare

A partir de [21], pode-se afirmar que é um framework para geração de aplicações, implementado em Java e que foi incorporado pela Eclipse.org, estando disponível como parte do Eclipse Modeling Project (http://www.eclipse.org/modeling/). O openArchitectureWare (OAW) representa um conjunto de melhores práticas e processos para usar a tecnologia de modelagem do Eclipse no desenvolvimento de software dirigido por modelos.

O objetivo do grupo openArchitectureWare trabalhar com a Eclipse.org é promover uma comunidade de pessoas interessadas no desenvolvimento de software orientado a modelo (MDSD) com base em OAW e Eclipse, sendo um dos resultados desta junção uma lista de componentes prontos para uso designados ao Eclipse que constroem o conjunto de ferramentas necessárias aos processos anteriores.

4.5.4 PathMate

Segundo [22], o PathMate transforma modelos independentes de plataforma (PIM)

em códigos eficientes com alto desempenho, trazendo o poder e flexibilidade da arquitetura orientada a modelos para o desenvolvimento de sistemas embarcados complexos de tempo real com alta performance e ainda com recursos limitados.

O PathMate é utilizado no Eclipse e baseado nos padrões do OMG para transformação de modelos e execução flexível em ambiente distintos sendo completamente código aberto e pronto para implantação em código C, C++ e Java.

Esta ferramenta proporciona benefícios estratégicos para os negócios dos clientes, como o desenvolvimento pelo menos duas vezes mais rápido, redução de de até 90% dos defeitos, redução drástica nos esforços de manutenção e implementação de novas características e reutilização de componentes sendo que além da proposta de um excelente desempenho, o PathMate também gera, de forma customizável, a documentação do projeto e relatórios.

4.5.5 Rhapsody

Conforme [23], o Rational Rhapsody 7.5.3 da IBM, é uma solução para o desenvolvimento dirigido a modelos que auxilia os engenheiros de sistemas e desenvolvedores de sistemas de tempo-real e software embarcado a melhorar a qualidade do produto e a produtividade além de realizar a análise de requisitos, validação do projeto e testes para a entrega de um produto com maior excelência.

Dentre os recursos desta ferramenta pode-se destacar o simulador de diagrama de atividades baseada em tokens, o AUTOSAR 4.0, para criação, importação e exportação em sistemas automotivos. São propostas melhorias relacionadas ao desempenho e usabilidade, em resumo, o propósito é providenciar mecanismos de desenvolvimento que construam produtos inovadores e confiáveis em menos tempo.

4.5.6 iQgen

Considerando a grande quantidade de esforço requerido para implementar tarefas como a persistência de objetos, relacionamentos, gerenciamento do ciclo de vida, etc, não apenas corretamente mas também em conformidade com a arquitetura, o iQgen, [24], propõe uma automatização da maioria destas tarefas, possibilitando que o desenvolvedor fique focado no essencial, a implementação do modelo de domínio do negócio e dos requisitos funcionais. Regras arquiteturais são

especificadas como templates que são aplicados aos modelos transformando-os em artefatos de implementação preservando as informações existentes.

A proposta do iQgen não é ser exclusivamente um instrumento do gestor ou analista, dado que os modelos são especificados usando a sintaxe Java Server Pages (JSP), eles podem ser editados com Java e outros códigos de implementação, permitindo que desenvolvedores utilizem meta-programação para as tarefas repetitivas.

O iQgen apóia o desenvolvimento em conformidade com a especificação da MDA, ou seja, pode-se criar um PIM, utilizando-se ferramentas CASE padrão e utilizar-se o iQgen para transformá-lo em PSM. Ao fatorar as regras de arquitetura em modelos, alterações em componentes de tecnologia ou de infra-estrutura podem ser facilmente incorporados, e na maioria dos casos, isso pode ser feito sem alterar os modelos existentes ou código de implementação.

4.5.7 IUML

De acordo com [25], esta ferramenta oferece um ambiente único para construções e teste de modelos de sistemas utilizando UML executável além de incluir gerenciamento de requisitos e de configurações e características multi-usuário. O centro desta ferramenta é o modelador iUML que permite a entrada gráfica de modelos UML e das descrições associadas, além de manter a consistência entre diferentes visões do mesmo elemento eliminado portanto a necessidade de introduzir a mesma informação duas ou mais vezes, mesmo quando vários usuários estão editando o mesmo modelo.

4.5.7 OpenMDX

Segundo [26], o openMDX é um framework de aplicação dirigido a modelos que ajuda na escrita orientada a serviços de aplicativos baseados em MDA. Padrões como JDO (Java Data Objects, MDA, REST (Representational State Transfer) e JCA (J2EE Connector Architecture) constituem as pedras angulares do openMDX. Estes padrões são implementados, combinados e integrados pelo openMDX, resultando em um framework leve e totalmente funcional.

O coração do openMDX é o gerenciador de persistência JDO openMDX que não armazena objetos no banco de dados, em vez disso, ele atua como um proxy,

que é capaz de delegar a um ou mais (local ou remoto) gestores de persistência.

Uma característica importante do gerenciador de persistência openMDX é o fato de ele ser plugável, e os plugins que implementam a lógica do negócio podem ser registrados como gerenciadores de persistência podendo então o gerenciador de persistência openMDX interceptar uma chamada e enviá-la para o plug-in adequado.

Outra característica relevante é que gerenciador de persistência openMDX é distribuível, podendo delegar para gestores de persistência locais ou remotos, sendo o protocolo utilizado para a comunicação local o REST/JCA e para a comunicação remota REST/HTTP, sendo que o openMDX fornece a infra-estrutura REST necessária.

4.5.8 Jamda

Conforme [27], Jamda é um framework de código aberto para a construção de geradores de aplicações que cria o código Java a partir de um modelo do domínio do negócio, onde, ao contrário de um gerador que produz para uma arquitetura fixa, Jamda oferece uma estrutura e blocos de construção para que seja possível construir um gerador de aplicação que faz exatamente o que um determinado projeto necessita. Ele inclui um gerador de exemplo para aplicações J2EE que pode ser adaptado às necessidades de um projeto J2EE, ou usado como base de um gerador para uma arquitetura completamente diferente.

A partir de um modelo UML do domínio da aplicação, um gerador criado com Jamda poderia criar o código para todas as funções padrão de localizar, exibir e atualizar os objetos de negócio na aplicação. O desenvolvedor iria concentrar-se na lógica de negócios específicos do aplicativo, que será incorporado ao aplicativo gerado. Em uma aplicação típica, o desenvolvedor só precisa escrever cerca de 20% do código total do sistema.

Um gerador de aplicativos criado usando Jamda desempenha o papel de um compilador de modelos na especificação MDA, sendo necessário um modelo de domínio UML como entrada para então adicionar-se novas classes ao modelo de apoio à execução, para então gerar o código executável.

A maioria dos compiladores de modelo existentes usam uma abordagem de templates para iniciar a partir do modelo UML original e então gerar o código final, no

caso do Jamda, primeiro adiciona-se as definições das classes geradas para o modelo UML para então gerar o código, utilizando, portanto, passos mais curtos e mais simples no processo de geração ao contrário de grandes saltos; também é realizado menos trabalho na especificação de cada classe gerada além dos geradores serem codificados em Java simples usando-se a API Jamda, ao invés de um modelo de linguagem menos flexivel e talvez desconhecida.

O modelo UML novo, incluindo as classes geradas podem ser visualizadas na mesma ferramenta usada para criar o modelo inicial, além de mais oportunidades para o reuso do gerador de módulos para diferentes aplicações

4.5.9 smartGENERATOR

A proposta do smartGENERATOR [28], é gerar software baseado em UML independente de plataformas, onde a ferramenta de UML é conectada através de XMI e a aplicação é gerada com apenas um clique em um botão.

O conjunto completo do código fonte pode ser obtido observando-se regras pré-estabelecidas sendo que a aplicação ao ser compilada gera o executável com todas as interfaces necessárias e as funções são simplesmente programadas como de forma usual.

4.5.10 Altova Umodel

A Altova Umodel 2011, [29], é uma ferramenta visual para projetos de aplicações com modelos UML e geração de código em Java, C# ou Visual Basic .NET e ainda a documentação do projeto.

Esta ferramenta também se presta a fazer engenharia reversa em diagramas UML 2.0 para refinar um projeto. A UModel é uma ferramenta UML visual simples e prática para projeto de software com bom custo benefício para desenhar em UML.

Umodel 2011 combina uma rica interface visual com características de fácil utilização além de suportar todos os 14 diagramas da UML 2.3 e ainda um diagrama especial para esquemas XML, o que garante liberdade para aspectos individuais de cada equipe de desenvolvimento. A edição Enterprise ainda oferece suporte a modelagem UML para bases de dados SQL.

4.5.11 Borland Together Architect

De acordo com [30], Together é uma plataforma de modelagem que permite a visualização e manutenção continuada de arquiteturas de TI. Se a tarefa envolve mudar os processos de negócio, criar novas aplicações ou extrair informações de projeto de sistemas existentes, Together mantém os analistas de negócios, analistas de sistemas, arquitetos, desenvolvedores e modeladores de dados sincronizados, com uma compreensão visual comum da arquitetura de aplicações.

O Together acelera o projeto, análise e desenvolvimento de aplicações corporativas, dado suas características, como o núcleo UML, modelagem de dados e capacidades da MDA que aumentam a agilidade e diminuem o custo de manutenção.

As suas capacidades DSL (Domain Specific Language) de modelos centrados no negócio geram uma maior compreensão e podem representar com maior precisão a complexidade do negócio.

Os Padrões de projeto melhoram a produtividade da equipe de desenvolvimento ao longo do ciclo de desenvolvimento e a modelagem independente da plataforma garante flexibilidade para suportar diferentes tipos de aplicações.

O suporte a padrões da indústria como UML, XMI, QVT, OCL de acordo com as especificações MDA garante manutenção e portabilidade e ainda a Tecnologia LiveSource, que oferece tecnologia de ida e volta, mantendo os modelos e código sincronizado em qualquer tempo.

A ferramenta conta com suporte para uma ampla gama de linguagens de programação (Java, J2EE, C + +, C #), o que garante uma maior flexibilidade através da criação de projetos de plataforma neutra que visam múltiplas plataformas.

4.5.12 Aonix Ameos

No início de 2007, a Aonix decidiu liberar o código-fonte Ameos ao público, logo, o OpenAmeos é a versão código aberto deste produto de modelagem UML. Segundo [31], Ameos combina UML 2.0, MDA e o uso de cores para garantir um maior nível de abstração dos modelos e modelagem independente do alvo.

Por suportar o padrão UML, Ameos, pode ser usado para descrever

processos de negócios, para projetar arquiteturas de sistemas de sofwtare e modelar os aspectos dinâmicos em máquinas de estado com restrições críticas de tempo.

Dentro do Ameos, uma coloração pode ser atribuída aos perfis UML e aos modelo de elementos, onde cada vez que tal um elemento é referenciado, ele aparece na cor atribuída sendo que esse uso da cor em um nível semântico conduz a modelos UML que são muito mais fáceis de ler.

Como membro da OMG, a Aonix promoveu a idéia de transformação dos modelos UML para o ambiente de destino usando Ameos / ACD, um poderoso motor de transformação, baseado em MDA que objetiva separar os aspectos técnicos dos aspectos de domínio na UML.

4.5.13 Code Futures Firestore

O FireStorm / DAO é um gerador de código que aumenta a produtividade dos desenvolvedores de software por gerar automaticamente o código DAO (Data Access Object) para acessar bancos de dados relacionais, reduzindo com isto os custos através de maior produtividade e diminuição da manutenção e implicando também em uma maior qualidade do software, [32].

Podem ser importados para o FireStorm / DAO esquemas de banco de dados já existentes, a partir de um script SQL ou de uma conexão JDBC, e então gerado uma camada de persistência de dados completa com base nas tecnologias de persistência como Java DAO, HIbernate DAO, Spring DAO ou Ruby.

Uma ferramenta de acesso de banco de dados para reduzir a complexidade FireStorm / DAO adota uma abordagem pragmática de geração de código Java DAO para persistência de dados que é um mapeamento direto do banco de dados relacional. Também é possível definir consultas complexas com várias tabelas aproveitando a lógica contida em store procedures.

O FireStorm / DAO pode gerar código DAO para Java autônomo, bem como para os principais servidores de aplicação J2EE,como JBoss, BEA WebLogic, IBM WebSphere, Tomcat e Apache. O código DAO gerado é bem escrito, consistente e contém documentação técnica.

4.5.14 MDRAD

De acordo com [33] o construtor MDRAD é uma ferramenta integrada a IDE Visual Studio que permite a criação de um conjunto de classes que seguem de perto o esquema do banco de dados e fornece uma API dando-lhe a funcionalidade CRUD (Create, Read, Update, Delete) completa em tempo de execução, sem a necessidade de escrever quaisquer instruções SQL .

É utilizado um modelo para descrever as entidades de negócios criados a partir do zero, ou através da engenharia reversa de um esquema de banco de dados ou esquema XML, podendo ser construído uma biblioteca de modelos. Usa-se partes deles para criar novos modelos. Após a definição do modelo, usa-se o kit de ferramentas para transformá-la em um componente .NET que é chamado de Modelo RunTime, que pode ser programado e implantado, juntamente com os outros componentes para uma aplicação.

Os objetos e seus relacionamentos, são descritos eficazmente pelo modelo, reduzindo-se a necessidade de documentação em separado. O modelo também permite a definição e geração do código e dos métodos que permitem estender a funcionalidade em tempo de execução de seus objetos.

O modelo pode aplicar as alterações na estrutura definida para o seu banco de dados para acelerar a implantação, sendo este modelo baseado em um projeto ratificado pela especificação MDA.

4.5.15 Objecteering MDA Modeler

O Objecteering MDA Modeler, segundo [34], fornece uma ferramenta de modelagem gráfica dedicada para ajudar a desenvolver as ferramentas necessárias na implementação de uma abordagem MDA em projetos de desenvolvimento de softwares.

Com o Objecteering MDA Modeler, pode-se adaptar, automatizar, auxiliar e controlar o desenvolvimento de aplicações, através da utilização de boas práticas metodológicas e as melhores opções tecnológicas para um domínio específico.

É possível facilmente construir uma ferramenta UML com o propósito de integrar todos os conhecimentos de um especialista, aumentando consideravelmente a produtividade e melhorando a qualidade no desenvolvimento

de aplicações.

Pode-se utilizar ferramentas durante o ciclo de desenvolvimento inteiro através da associação e combinação de componentes MDA para a aplicação sucessiva em seus modelos.

Através da utilização de componentes MDA, características como aparência, ergonomia, menus e comandos do Objecteering Modeler se adaptam automaticamente às opções escolhidas.

Os modelos são construídos de acordo com regras metodológicas, os padrões são automatizados, os documentos são gerados usando-se templates de documentos, e a geração de código específico é sistematizada.

4.6 CONCLUSÃO

Este capítulo discutiu conceitos fundamentais da MDA, apresentando conceitos básicos, uma visão geral do processo necessário a sua implementação, bem como seu ciclo de vida, além de descrever diversas ferramentas disponíveis atualmente para trabalhar com MDA

5. USANDO MDA

O propósito deste capítulo é descrever o uso de uma ferramenta disponível no mercado, com a intenção de avaliar os desafios inerentes a esta tarefa através de um tutorial sobre a ferramenta AndroMDA.

5.1 INTRODUÇÃO

Como visto no capítulo anterior, existem disponíveis muitas ferramentas que se propõe a trabalhar com o paradigma orientado a modelos, não necessariamente cada solução incorpora todos os conceitos descritos na especificação da MDA, logo é importante conhecer mais profundamente as características da ferramenta para identificar como ele trata cada conceito descrito na especificação.

Este tutorial foi resultado da compilação de exemplos disponíveis na literatura, e tem a intenção de descrever uma primeira experiência prática no âmbito do desenvolvimento orientado a modelos com a utilização de uma das ferramentas disponíveis atualmente, sem conduto, neste momento, um uso mais profícuo da ferramenta.

A AndroMDA foi escolhida devido sua popularidade na literatura, porém sem ter havido uma avaliação técnica de suas proposta, porém, como a intenção é visualizar o processo em funcionamento, isto foi considerado suficiente.

Os demais programas necessários para o ambiente de desenvolvimento também não foram tecnicamente avaliados, porém são indicados pela documentação da ferramenta escolhida e possuem algum destaque na literatura embora, a priori, possam ser substituídas por equivalentes sem prejuízo ao processo, por fim, o sistema operacional de suporte das ferramentas do tutorial é o Microsoft Windows XP.

5.2 A CRIAÇÃO DO AMBIENTE

O primeiro passo é construir o ambiente necessário para o desenvolvimento orientado a modelos, e neste caso inicia-se pelo download e instalação das aplicações necessárias ao processo, isto considerando que o ambiente já possui

Java instalado, sendo recomendado versões superiores a 1.6.0_10.

O Apache – Maven, [35], tem por função facilitar o processo de construção, configuração e padronização no desenvolvimento de softwares, sendo neste tutorial responsável por criar as estruturas do projeto, download de arquivos .jar relativos as dependências, bem como a própria transformação dos modelos UML nos artefatos de software. Por possuir um escopo de funcionamento similar ao do projeto ANT, o Maven é considerado por alguns uma evolução do mesmo.

Para edição UML será utilizado o Argo UML, [36], um editor de UML livre e baseado em Java que para este tutorial suporta o padrão de UML 1.4. A partir do Argo UML é que serão gerados os modelos UML para posterior transformação no produto final, neste caso, o código fonte.

O Hypersonic HSQLDB, [37], é um projeto de banco de dados livre, escrito inteiramente em Java, que permite a manipulação de banco de dados em uma arquitetura cliente-servidor ou standalone que apresenta como vantagens o fato de poder ser anexado a aplicações de forma transparente, ser multi-plataforma e ainda ocupar um espaço minúsculo nas distribuições.

O JBoss Aplication Server, [38], é a implementação de um servidor J2EE livre, certificado para desenvolvimento e implantação de aplicações JAVA/J2EE que possui uma excelente aceitação no mercado e devido ao fato do mesmo ser 100% baseado em Java, ele pode ser usado em qualquer sistema operacional que suporte Java.

O plugin do AndroMDA, [19], é uma extensão para a utilização das funcionalidades do AndroMDA no Maven que tem por propósito criar as estruturas de projeto, bem como ser utilizado como o mecanismo de transformação.

Por questões didáticas e de boas práticas usou-se uma estrutura de arquivos padrão, onde cada pasta possui um objetivo específico, neste caso, como pode-se observar na figura a seguir, criou-se na raiz do sistema uma pasta denominada MDA_HOME que funcionará como root, a pasta APPS conterá os aplicativos, a DEV servirá para armazenar o projeto, a PLUGIN armazenará tanto o arquivo.zip quanto o plugin descompactado do AndroMDA e a pasta PROFILES guardará os perfis disponibilizados pelo AndroMDA. Dado esta configuração, os aplicativos do ambiente devem ser descompactados na pasta APPS com exceção do plugin do

AndroMDA que deverá ir para pasta PLUGIN. A figura 5.1 exibe o resultado da estrutura de pastas.

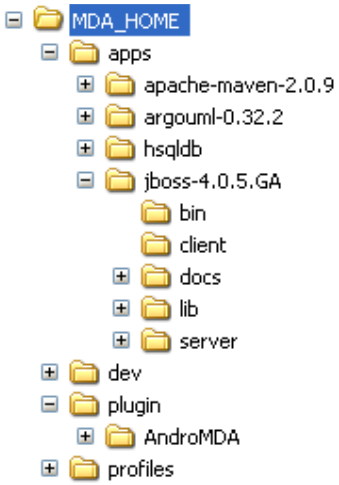

Figura 5.1 – Estrutura de arquivos do ambiente

5.3 CONFIGURANDO O AMBIENTE

Para o correto funcionamento dos mecanismos de transformação do AndroMDA é necessário a configuração de variáveis de ambiente do sistema operacional e adição das mesmas ao PATH do sistema, logo, devem ser criadas as seguinte variáveis de ambiente :

1. **MDA_HOME** = C:\MDA_HOME
2. **JAVA_HOME = o local de instalação de Java no sistema**
3. **JBOSS_HOME** = %MDA_HOME%\apps\jboss-4.0.5.GA
4. **M2_HOME** = %MDA_HOME%\apps\apache-maven-2.0.9
5. **M2_REPO** = C:\Documents and Settings**hpb**\.m2\repository , onde hpb representa o usuário do sistema.

A busca por dependências pode gerar um grande transtorno e tomar muito tempo, por isto a documentação do AndroMDA disponibiliza um projeto com a finalidade de realizar o download das bibliotecas e dependências que deverão ser utilizadas pela ferramenta, podendo o mesmo ser adequado para trazer perfis de desenvolvimento que adicionam funcionalidade aos modelos UML e são utilizados na forma de Estereótipos ou Tags.

Ao utilizar-se, por exemplo, um service profile, indica-se que determinadas entidades do modelo representam serviços que devem ser traduzidos no processo de transformação, indicando que a classe marcada possui estas funcionalidades e devem ser implementadas automaticamente.

Inicialmente, deve-se rodar o plugin do AndroMDA, figura 5.2, para que ele gere toda a estrutura inicial da aplicação, bem como baixar as dependências necessárias a execução desta primeira aplicação, ou seja, acontecerá um série de downloads que eventualmente, dependendo da conexão com a internet, pode demorar um pouco.

Além disto, o plugin faz uma série de questionamentos ao usuário com o objetivo de poder customizar a configuração do projeto, como pode ser visto na figura 5.3, inclusive contendo as respostas adequadas para este tutorial, que serão utilizadas posteriormente.

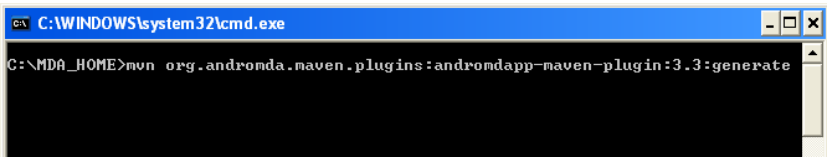

Figura 5.2 – Comando de execução do Maven

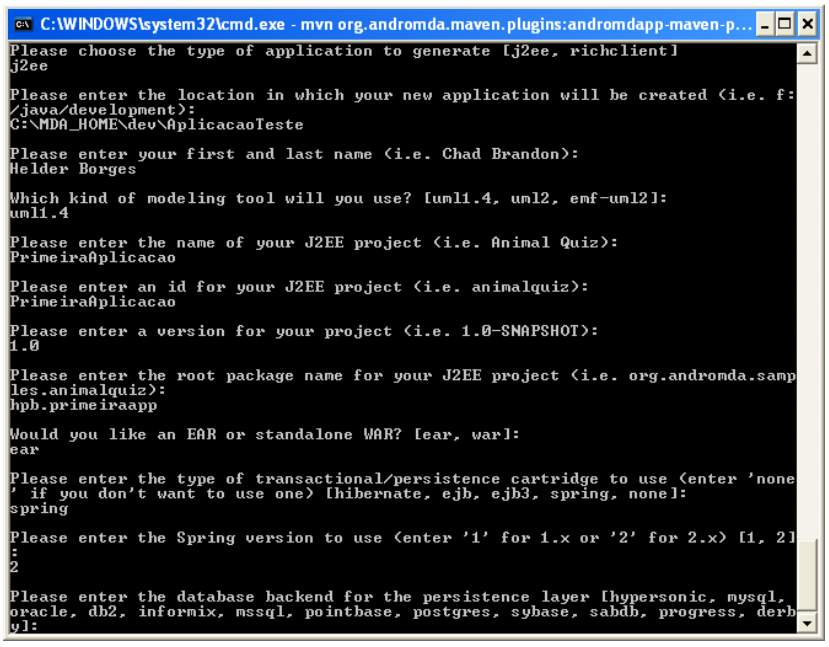

Figura 5.3 – Questões do plugin Maven

Como resultado deste processamento, após a inclusão de todas as respostas deverá ser exibido no prompt a mensagem “BUILD SUCCESSFUL”, indicando que o esqueleto de aplicação foi construído com sucesso, como pode ser observado na figura 5.4, sendo ainda exibido a nova estrutura de diretórios que foi criada na figura 5.5.

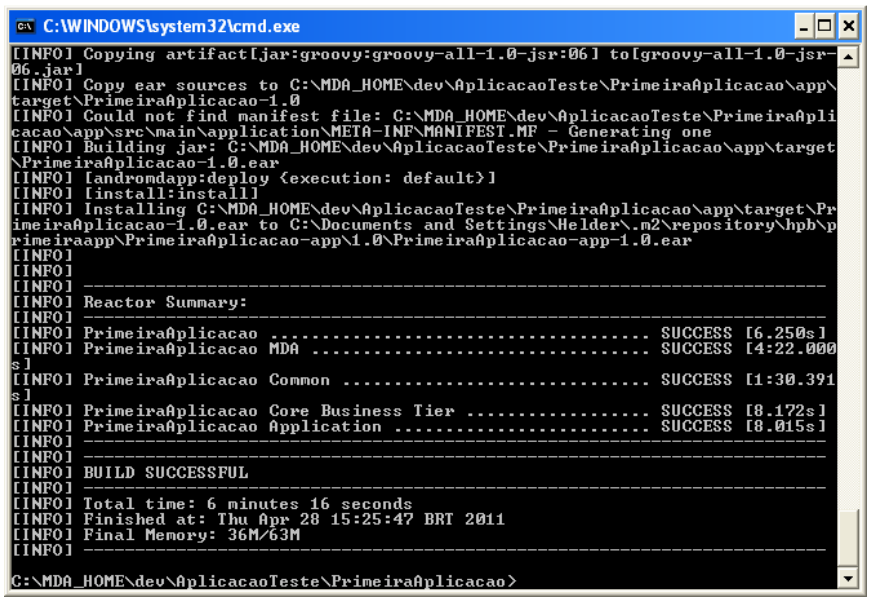

Figura 5.4 – Resultado do comando do Maven

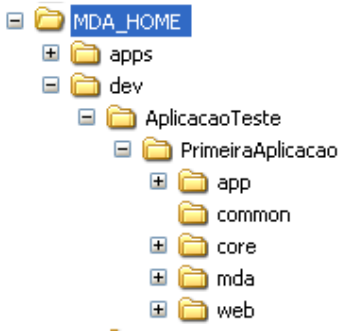

Figura 5.5 – Estrutura de arquivos criada pelo plugin Maven

A estrutura de pastas criada obedece a seguinte padronização, a pasta APP contêm os recursos utilizados na criação do .ear; a pasta CORE contêm os arquivos de mapeamento objeto relacional, lógica de acesso a dados, serviços e outras funcionalidades consideradas como núcleo da aplicação; na pasta COMMON estão os recursos e classes compartilhadas entre os sub-projetos; na pasta MDA estão tanto os descritores de transformação, quanto o próprio modelo de classes; e na

pasta WEB constarão todos os recursos utilizados para uma aplicação web, tais como imagens, arquivos .jsp, servlets, etc.

Após a criação do esqueleto da aplicação, deverão ser realizadas algumas configurações adicionais, portanto, dentro do diretório root (\PrimeiraAplicacao) deve ser editado o arquivo pom.xml com o objetivo de configurar o banco de dados. Primeiro, altera-se a tag **<jdbc.driver.jar>** para "***${env.MDA_HOME}/apps/hsqldb/lib/hsqldb.jar***", mudando com isto a localização do arquivo hsqldb.jar e com isto o projeto utiliza a variável de ambiente MDA_HOME para acessar o arquivo .jar.

A próxima alteração acontecerá na URL de conexão ao banco de dados modificando o valor da tag **<jdbc.url>** para ***jdbc:hsqldb:hsql://localhost/bancodeteste***. Usuário e senha do banco serão os default fornecidos pelo Hypersonic e portanto as tags <jdbc.username> e <jdbc.password> não precisam ser alteradas.

Dado a utilização do Argo UML o arquivo ..\PrimeiraAplicação\mda\pom.xml deve ser alterado para indicar o novo modelo que será utilizado, logo a tag **<model.uri>** deve ser modificada para ***jar:file:${project.basedir}\src\main\uml\PrimeiraAplicacao.zargo!\ PrimeiraAplicacao.xmi***.

Os perfis do plugin AndroMDA relacionados a UML 1.4 que se encontram na pasta %MDA_HOME%\plugin\AndroMDA\andromda\org\andromda\profiles\uml14 devem ser adicionados ao ArgoUML através do menu Editar >> Configurações, botão Adicionar para selecionar a pasta dos perfis e em seguida botão Atualizar Lista. A partir de então, todos os profiles serão carregados e tem-se a opção de adicionar ou não estes profiles em "Default Profiles", de forma que os mesmos irão constar em todos os projetos do ArgoUML.

5.4 CONSTRUINDO O MODELO

O propósito é construir um modelo UML simples que servirá de base para geração dos artefatos de software através da transformação realizada pela engine do Andromda.

No Argo UML cria-se um projeto novo através do menu Arquivo >> Novo, então deve-se adicionar os profiles que se deseja utilizar no projeto (menu Arquivo >> Propriedades >> Perfis), para este tutorial devem ser adicionados os seguinte

profiles:

- andromda-profile-datatype
- andromda-profile-persistence
- andromda-profile-meta
- andromda-profile-service
- andromda-profile

As respostas dadas aos questionamentos do plugin do AndroMDA estão refletidas nos próximos passos, quanto ao nome da aplicação e definição dos pacotes, portanto o nome da aplicação deve ser alterado para PrimeiraAplicacao. Com a aplicação selecionada, usando o botão direito do mouse >> Criar Elemento do Modelo >> Novo Pacote deve-se construir a estrutura de pacotes conforme a figura 5.6.

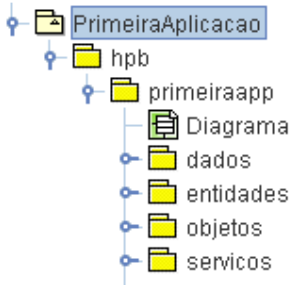

Figura 5.6 – Estrutura de pacotes do modelo

O modelo deve ser salvo com o nome de "**PrimeiraAplicacao.zargo**" na pasta %MDA_HOME%\dev\AplicacaoTeste\PrimeiraAplicacao\mda\src\main\uml, isto porque foi configurado a propriedade "**model.uri**" com este valor, embora somente agora criou-se o arquivo que deverá ser utilizado nesta propriedade.

Para este projeto serão modeladas quatro classes, uma Cliente, no pacote objetos, que tem como responsabilidade o encapsulamento dos dados de um cliente provenientes do banco de dados. A classe de acesso aos dados que faz a ligação entre uma base de dados e um requisitante será denominada ClienteDAO.

Minimamente, se espera de uma classe DAO, o Create, Update, Read, Delete, comumente denominado de CRUD, por isto, neste tutorial serão criados os seguintes métodos:

+	buscaporChava(id: Integer) : Cliente
+	update(cliente: Cliente):Cliente
+	delete(cliente: Cliente):void
+	insert(cliente: Cliente):Cliente

Cria-se uma classe no ArgoUML selecionando-se o pacote desejado através do botão direito do mouse >> Criar Elemento do Modelo >> Nova Classe, e se adiciona atributos e métodos através da aba Propriedades.

Uma vez criado o “Value Object” correspondente a Entidade Cliente, nada mais intuitivo do que também criar a Entidade correspondente, assim pode-se manter todos os detalhes da DDL (Data Definition Language) que será gerada de forma a representar o que foi modelado.

De forma simplória, replica-se o que foi modelado no pacote objetos e a seguir no modelo serão criadas referências de uso entre as classes, com apenas um detalhe, os campos que pertencentes ao entity deverão ser do tipo “Data” devido a correspondência para os tipos mais comuns existentes nos bancos de dados. O nome desta nova classe será ClienteEntity e deve estar no pacote data.

Toda aplicação necessita de um ponto de entrada, que geralmente é implementado como interface, para isolar a lógica interna da aplicação, logo será criado a classe ClienteService, que será responsável por expor os serviços referentes a clientes considerando o acesso a partir de outros pontos da aplicação, por isto serão criados os métodos a seguir:

+	buscarCliente(clienteID : Integer):Cliente
+	salvarCliente(cliente : Cliente):Cliente
+	excluirCliente(cliente : Cliente):void

Para criar o Diagrama de Classes basta clicar com o botão direito no pacote primeirapp e na seqüência Criar Diagrama >> Novo Diagrama de Classes. Neste diagrama devem ser definidas todas as dependências entre as classes e para isto deve ser utilizada a ferramenta Nova Dependência. Na figura 5.7, tem-se o resultado da modelagem e dependência entre as classes do projeto.

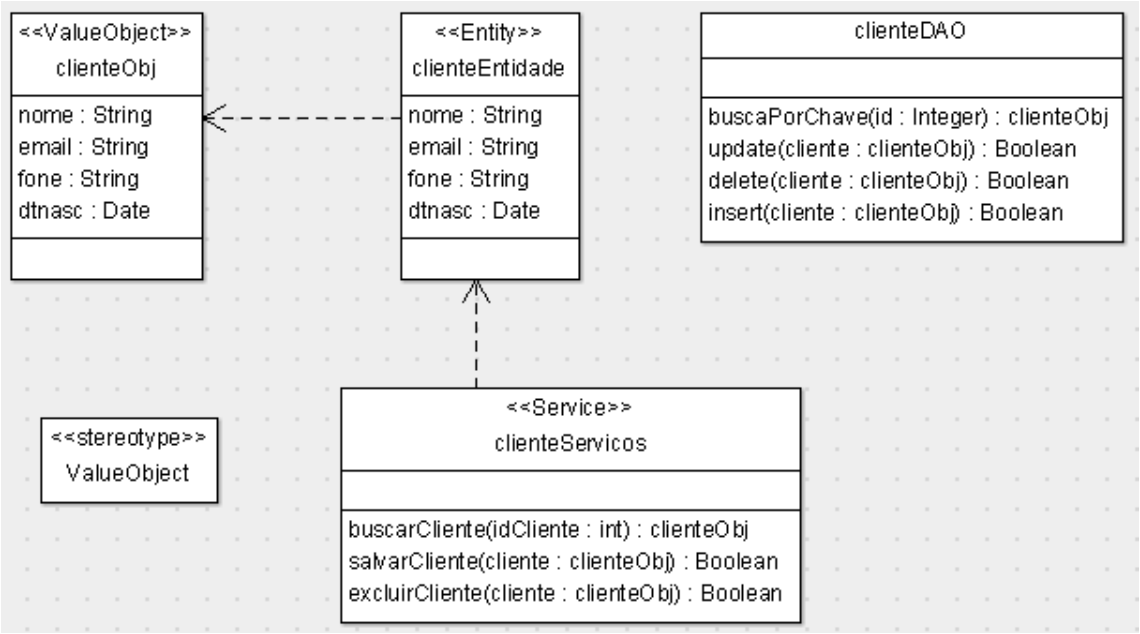

Figura 5.7 – Modelagem do Sistema

Um estereótipo indica como uma classe é utilizada pelo projeto, eles são mecanismos de extensibilidade da UML que permitem classificar elementos que possuem semelhanças. No mundo da MDA tudo funciona a partir dos estereótipos, por isto, cada classe deve ser marcada com seu respectivo estereótipo, adicionando assim as funcionalidades previstas para cada um. Começando pela classe ClienteEntity, após sua seleção, na aba Estereótipo adiciona-se o estereótipo “Entity” a classe. Este procedimento deve ser repetido para a classe ClienteService, porém adicionado o esterótipo <<Service>>.

No caso deste tutorial, este modelo representa o PIM do processo MDA, então agora pode-se executar o Maven para gerar os artefatos a partir deste diagrama. A partir de um prompt do DOS, de dentro do diretorio: **%MDA_HOME%\dev\AplicacaoTeste\PrimeiraAplicacao,** deve ser executado o comando **mvn install**. O comando com função contraria ao **install**, é o **clean,** e ao utilizar o comando **mvn clean** todos os artefatos, com exceção das implementações, serão excluídos.

O último passo será a geração da DDL (Data Definition Language) para a criação propriamente dita das entidades no banco de dados. O comando utilizado para essa geração, dentro do diretório **%MDA_HOME%\dev\AplicacaoTeste\PrimeiraAplicacao** é o **mvn –f**

core/pom.xml andromdapp:schema -Dtasks=create, obviamente, é necessário que antes da execução do comando o banco de dados esteja ativo.

Na pasta %MDA_HOME%apps\hsqldb\bin, encontra-se o arquivo runserver.bat, o objetivo é configurar o servidor para prover a base de dados que foi configurada anteriormente, portanto o novo conteúdo do arquivo deverá ser:

```
cd ..\data
rem @java -classpath ../lib/hsqldb.jar org.hsqldb.server.Server %1 %2 %3 %4 %5
%6 %7 %8 %9
java -cp ../lib/hsqldb.jar org.hsqldb.server.Server –database.0 file: bancodeteste –
dbname.0 bancodeteste
```

Uma vez feito isso, deve-se iniciar o servidor através do runserver.bat, conforme figura 5.8, para que automaticamente o Hypersonic crie a base "bancodeteste" e posteriormente deve ser executado o comando especificado anteriormente para criar as entidades no banco de dados, que neste caso é apenas a tabela CLIENTE_ENTITY. Para excluir todas as entidades do modelo, o comando reverso a criação de tabelas é o **mvn -f core/pom.xml andromdapp:schema -Dtasks=drop.**

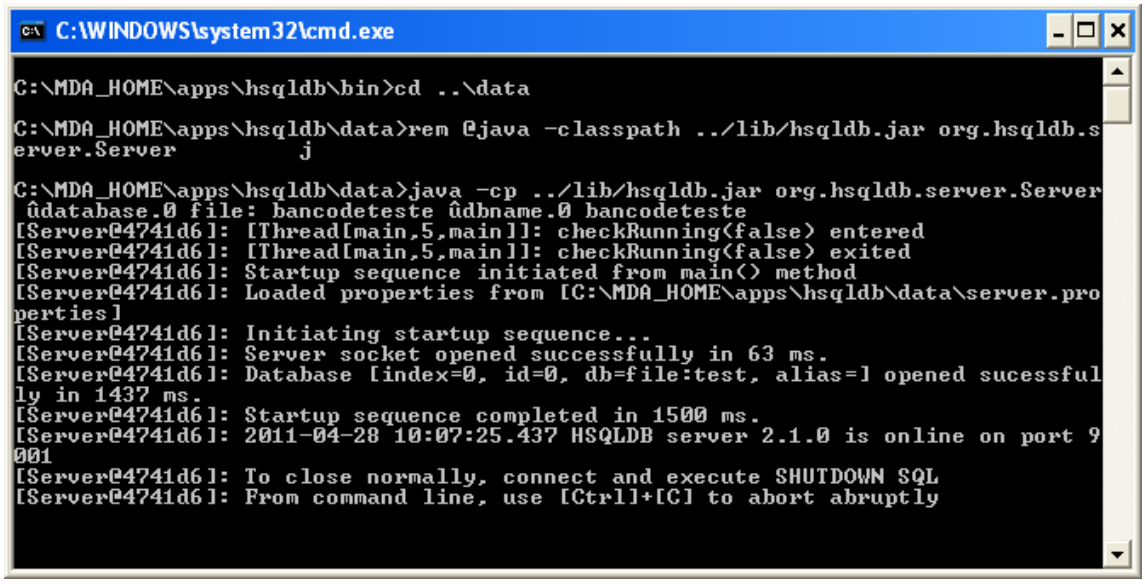

Figura 5.8 – Resultado do arquivo runserver.bat

5.5 CONCLUSÃO

Este capítulo descreveu a utilização da ferramenta AndroMDA para implementar o processo MDA. Foram abordados a criação e configuração do ambiente bem como a construção do modelo e ainda a utilização da ferramenta para gerar o código fonte.

Pode-se perceber um certo trabalho para a configuração do ambiente, porém sem grandes dificuldade técnicas. Para a criação dos modelos, foi preciso apenas familiaridade com as interfaces da ferramenta visto que o processo se mostrou intuitivo.

O capítulo seguinte apresenta as conclusões deste trabalho.

6. CONCLUSÃO

Esta trabalho apresentou um estudo sobre desenvolvimento de software baseado em modelos, sendo discutidos os desafios pertinentes a construção de sistemas. Este estudo abordou a especificação MDA do OMG, descrevendo seus fundamentos tecnológicos e conceitos essências.

6.1 CONCLUSÃO E TRABALHOS FUTUROS

O desenvolvimento de software baseado em modelos tem provado sua eficiência ao longo dos anos, sendo a Modelagem Dirigida a Modelos - MDA, uma iniciativa do Object Management Group, uma proposta para este segmento que se diferencia das demais pela separação entre modelagem e plataforma e ainda pela geração automática do código a partir do modelo do sistema.

Este trabalho apresentou os principais aspectos referentes a MDA, destacando seus modelos, mapeamentos e transformações. Foi possível perceber que a especificação MDA é bastante flexível e pelas características do seu processo, várias vantagens podem ser agregadas ao desenvolvimento de software, tais como portabilidade e interoperabilidade bem como rápida adequação a necessidade de alterações.

Como contribuições deste trabalho, pode-se relacionar a descrição do processo de desenvolvimento de software utilizando-se MDA, a especificação de um catálogo de ferramentas para desenvolvimento baseado em modelos disponíveis no mercado e ainda a descrição de uma experiência prática com a abordagem MDA.

Como trabalhos futuros pretende-se, estudar mais detalhadamente a abordagem MDA para desenvolvimento de software para efetivamente poder criar sistemas utilizando este mecanismo. Em seguida, ambiciona-se identificar quais ferramentas, de código aberto, implementam com precisão os conceitos da MDA e apresentam os melhores mecanismos e funcionalidades.

Por fim, com as informações obtidas neste trabalho, iremos direcionar a abordagem para o desenvolvimento de Software como Serviço (SaaS), desenvolvendo-se uma ferramenta para desenvolvimento automático de software.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Naur, P., Randell, B. **Software Engineering: Report on a Conference sponsored by the NATO Science Committee**. Scientific Affairs Division, NATO, 1969.

[2] Miller, J., Mukerji, J. **Model Driven Architecture**. Architecture Board ORMSC, 2001.

[3] OMG. **MDA Guide Version 1.0.1**. 2003. Disponível em: < http:// www .omg .org / docs/omg/03-06-01.pdf>. Acesso em: 12 fev 2011.

[4] OMG. **Object Management Architecture**. Disponível em: < http://www.omg. org / oma/>. Acesso em: 15 fev 2011.

[5] OMG. **Common Object Request Broker.** Disponível em: < http://www.omg.org/ spec/CORBA/ >. Acesso em: 12 fev 2011.

[6] OMG. **Unified Modeling Language**. Disponível em: <http://www.uml.org/>. Acesso em: 12 fev 2011.

[7] FAVRE, J. **Towards a Basic Theory to Model Driven Engineering**. Workshop in Software Model Engineering (WISME 2004), 2004.

[8] OMG. **XML Metadata Interchange**. Disponível em: <http://www.omg.org/ spec/ XMI/>. Acesso em: 12 fev 2011.

[9] OMG. **Common Warehouse Metamodel.** Disponível em: <http://www.omg.org/ spec/CWM/>. Acesso em: 13 fev 2011.

[10] CALIARI, G. **Transformações e mapeamentos da MDA e sua implementação em três ferramentas**. USP. 2007.

[11] MELLOR, J. **MDA Distilled – Principles of Model Driven Architecture**. Addison-Wesley. 2004.

[12] KLEPPE, A. WARMER, J. BAST, W. **MDA Explained – The Model Driven Architecture: Practice an Promise**. Addison -Wesley. 2003.

[13] OMG. **Meta Object Facility (MOF) Core Specification – Version 2.0**. 2006. Disponível em: <http://www.omg.org/docs/formal/06-01-01.pdf>. Acesso em: 14 fev 2011.

[14] OMG. **OMG's MetaObject Facility**. Disponível em: <http://www.omg.org/mof>. Acesso em: 16 fev 2011.

[15] PRESSMAN, R.S. **Engenharia de Software**. 5.ed. McGraw-Hill, 2001. 843p.

[16] SELIC, B. **The pragmatics of model-driven development**. IEEE Software, v.20, n.5, p. 19-25. 2003.

[17] FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3a. edição. Addison-Wesley, 2004.

[18] LIMA, B., SOUSA J., LOPES, D., **Using MDA to Support Hypermedia Document Sharing**. IEEE International Conference on Software Engineering Advances (ICSEA 2007), French Riviera, France, 2007.

[19] ANDROMDA. **AndroMDA Model Driven Architecture Framework**. Disponível em: < http://www.andromda.org/docs/index.html>. Acesso em: 28 fev 2011.

[20] BLUAGE. **Blu Age Model Driven Modernization.** Disponível em: < http://www.bluage.com/>. Acesso em: 12 mar 2011.

[21] OPENARCHITECTUREWARE. **OpenArchitectureWare Working Group.** Disponível em: < http://www.eclipse.org/workinggroups/oaw/>. Acesso em: 12 mar 2011.

[22] PATHFINDER. **Pathfinder Solutions**. Disponível em: < http://www. pathfindermda.com/>. Acesso em: 08 mar 2011.

[23] IBM. **IBM Rational Rhapsody.** Disponível em: < http://www.ibm.com/ developerworks/rational/products/rhapsody/>. Acesso em: 05 mar 2011.

[24] INNOQ. **iQgen – The Model Driven Software Generator**. Disponível em: < http://www.innoq.com/iqgen/>. Acesso em: 06 mar 2011.

[25] ABSTRACT SOLUTIONS. **iUML Modeller ans Simulator**. Disponível em: < http://www.kc.com/PRODUCTS/iuml/index.php >. Acesso em: 17 mar 2011.

[26] OPEN MDX. **openMDX**. Disponível em: < http://www.openmdx.org/#HOME >. Acesso em: 01 mar 2011.

[27] JAMDA. **The Jamda Project**. Disponível em: < http://jamda.sourceforge.net/>. Acesso em: 01 mar 2011.

[28] BITPLAN. **BITPlan smartGENERATOR.** Disponível em: < http://www.bitplan.

com/com/bitplan/web/index.php?topic=products/smartgenerator>. Acesso em: 04 mar 2011.

[29] ALTOVA. **UModel - UML tool for software modeling and application development**. Disponível em: < http://www.altova.com/umodel.html >. Acesso em: 03 mar 2011.

[30] BORLAND. **Together – Visual Modeling for Software Architecture Design**.Disponível em: < http://www.borland.com/us/products/together/>. Acesso em: 12 mar 2011.

[31] AMEOS. **The Next Generation Modeling Tool**. Disponível em: < http://www.aonix.com/ameos.html>. Acesso em: 22 mar 2011.

[32] CODEFUTURES. **Database Access Tool: FireStorm/DAO**. Disponível em: <http://www.codefutures.com/products/firestorm/>. Acesso em: 25 mar 2011.

[33] i3DESIGN. **Constructor MDRAD**. Disponível em: <http://www.i3design.co.uk/Constructor/MDRAD/>. Acesso em: 19 mar 2011.

[34] OBJECTEERING. **The model-driven development tool.** Disponível em: <http://www.objecteering.com/>. Acesso em: 19 mar 2011 .

[35] APACHE. **Apache Maven**. Disponível em: < http://maven.apache.org/>. Acesso em: 19 mar 2011.

[36] TIGRIS. **ArgoUML**. Disponível em: < http://argouml.tigris.org/>. Acesso em: 19 mar 2011.

[37] HYPERSQL. **HyperSQL**. Disponível em: < http://hsqldb.org/>. Acesso em: 19 mar 2011.

[38] JBOSS. **JBoss Application Server**. Disponível em: < http://jboss.org/jbossas/>. Acesso em: 19 mar 2011.

[39] RITU, S., MANU, S. **Cloud SaaS and Model Driven Architecture**. International Conference on Advanced Computing and Communication Technologies (ACCT 2011).

[40] COCHINWALA, M., SHIM, H., WULLERT, R. **A model-driven approach to rapid service introduction**. Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on , vol., no., pp. 659- 672, 15-19 May 2005

[41] XIAOYAN, J., YONG, Z., SHIJUN, L. **A Well-designed SaaS Application Platform Based on Model-driven Approach**. Gcc, pp.276-281, 2010. Ninth International Conference on Grid and Cloud Computing, 2010.

[42] SANZ, M., ACUNA, J., CUESTA, C., ESPERANZA, M. **Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM level**. Wicsa, pp.309-312, Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), 2008.

[43] Xu Xiao; Sun Hailong; Li Xiang; Zhou Chao. **A Basing on Model-Driven Framework of Service-Oriented Software Production Line.** Computational Intelligence and Design, 2009. ISCID '09. Second International Symposium on , vol.2, no., pp.139-145, 12-14 Dec. 2009.