UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

DIEGO MARTINHAGO

DESENVOLVIMENTO DE SISTEMAS COM MODEL DRIVEN ARCHITECTURE

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2012

DIEGO MARTINHAGO

DESENVOLVIMENTO DE SISTEMAS COM MODEL DRIVEN ARCHITECTURE

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de Software, da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Medianeira.

Orientador: Prof. Alan Gavioli.

Co-orientador: Prof. Juliano Rodrigo Lamb.

MEDIANEIRA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Especialização em Engenharia de Software

TERMO DE APROVAÇÃO

Produtividade no Desenvolvimento de Sistemas com Model Driven Architecture

Por

Diego Martinhago

Esta monografia foi apresentada às 13:00 h do dia 23 de março de 2012 como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de Software, da Universidade Tecnológica Federal do Paraná, *Câmpus* Medianeira. O acadêmico foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M.Sc Alan Gavioli
UTFPR – Câmpus Medianeira
(orientador)

_____
Prof. M.Sc Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(co-orientador)

_____
Prof M.SC. Fernando Schutz
UTFPR – Câmpus Medianeira

AGRADECIMENTOS

A Deus pelo dom da vida, pela fé e perseverança para vencer os obstáculos.

A todos os meus colegas de classe, pelo companheirismo durante toda a jornada.

Ao professor orientador Alan Gavioli, pela orientação neste projeto e pelos ensinamentos proporcionados durante o curso.

Ao professor co-orientador Juliano Rodrigo Lamb. Sua orientação foi fundamental para a realização deste projeto.

Agradeço aos pesquisadores e professores do curso de Especialização em Engenharia de Software, professores da UTFPR, *Câmpu*s Medianeira.

Enfim, sou grato a todos que contribuíram de forma direta ou indireta para realização desta monografia.

“Concentre-se naquilo que você é bom,

delegue todo o resto”.

(STEVE JOBS)

RESUMO

MARTINHAGO, Diego. Desenvolvimento de Sistemas com Model Driven Architecture. 2012. 51 folhas. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

O presente trabalho descreve e realiza um estudo sobre *framework* Model Driven Architecture (MDA). Uma abordagem de desenvolvimento de software dirigido por modelos que apresenta diferentes níveis de abstração com o intuito de separar a arquitetura conceitual do sistema de sua implementação específica. A modelagem fica no centro do processo de desenvolvimento, permitindo a geração automática do código fonte através dos diagramas Unified Modeling Language (UML), reduzindo o tempo de desenvolvimento, aumentando a qualidade do *software* e evitando que a modelagem fique defasada.

Palavras-chave: Desenvolvimento de Software, Modelagem, Arquitetura, Qualidade, Agilidade.

ABSTRACT

Martinhago, Diego. Systems Development with Model Driven Architecture. 2012. 51 folhas. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

This paper describes and performs a study on Model Driven Architecture (MDA). An approach to software development driven models featuring different levels of abstraction in order to separate the conceptual architecture of the system of its specific implementation. The modeling is in the center of the development process, allowing the automatic generation of source code through Unified Modeling Language (UML) diagrams, reducing development time, increasing software quality and preventing the modeling becomes outdated.

Keywords: Software Development, Modeling, Architecture, Quality, Agility.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

LISTA DE SIGLAS

CIM	-	Computation Independent Model
IM	-	Implementation model
JSF	-	Java Server Faces
MDA	-	Model Driven Architecture
MOF	-	Meta-Object Facility
OMG	-	Object Management Group
PIM	-	Platform Independent Model
POM	-	Project Object Model
PSM	-	Platform Specific Model
RSM	-	Rational Software Modeler/Architect
UML	-	Unified Modeling Language

1 INTRODUÇÃO

As organizações que desenvolvem *software* vêm reconhecendo a importância de se reaproveitar o esforço empregado no desenvolvimento de sistemas para reduzir prazo, recursos, e aumentar a produtividade e a qualidade.

Um grande desafio enfrentado pelas equipes é a escolha das tecnologias a serem empregadas no desenvolvimento. O problema é que artefatos desenvolvidos podem apresentar uma redução no seu potencial de reutilização, pois seu tempo de vida está associado ao tempo de vida da plataforma escolhida. Além disso, a incompatibilidade entre as plataformas limita o potencial de reutilização apenas às aplicações desenvolvidas na mesma plataforma (SILVA e BARRETO, 2008).

Model Driven Architecture (MDA) é uma abordagem de desenvolvimento de software dirigido por modelos que apresenta diferentes níveis de abstração com o intuito de separar a arquitetura conceitual do sistema de sua implementação específica. Dessa forma, o foco é centrado no desenvolvimento da aplicação e não nos detalhes de tecnologia. A definição de mecanismos de transformação permite que os modelos descritos em linguagem de alto nível sejam traduzidos, em um ou mais passos, em código executável, o que tende a facilitar o reuso e a manutenção das aplicações (SILVA e BARRETO, 2008).

O Desenvolvimento MDA concentra-se primeiro sobre a funcionalidade do sistema, independente da tecnologia ou plataforma em que será implementado. Desta forma, MDA separa detalhes de implementação de funções de negócios. Assim, não é necessário repetir o processo de definição das funcionalidades do sistema cada vez que uma nova tecnologia é apresentada, enquanto que outras metodologias estão geralmente vinculadas a uma tecnologia específica. Com MDA, as funcionalidades são modeladas apenas uma única vez. O mapeamento dos modelos para as linguagens suportadas pelo MDA é efetuado por ferramentas, facilitando a tarefa de utilizar tecnologias novas ou diferentes (OMG Model Driven Architecture, 2011).

1.1 OBJETIVO GERAL

Explorar a metodologia MDA por meio da apresentação dos diversos conceitos relacionados a ela e também através da modelagem completa de um estudo de caso.

1.2 OBJETIVOS ESPECÍFICOS

- Apresentar os diversos conceitos correspondentes a MDA, em especial abordando modelos, níveis, transformações e benefícios dessa metodologia;
- Avaliar a utilização prática da metodologia MDA em um estudo de caso;
- Avaliar os benefícios da abordagem MDA em ralação ao método tradicional de desenvolvimento.

1.3 JUSTIFICATIVA

Durante o processo de desenvolvimento de *software*, as fases iniciais produzem basicamente especificações e diagramas. Ao se codificar, esses documentos acabam perdendo valor, pois o desenvolvedor naturalmente, ao realizar uma manutenção, toma um atalho e efetua as mudanças diretamente no código, sem atualizar os modelos. Com o surgimento de novas tecnologias de forma cada vez mais rápida, os sistemas acabam sendo refeitos, aproveitando muito pouco do trabalho já feito, extinguindo a necessidade de aproveitamento de parte do trabalho já realizado em desenvolvimentos anteriores. A metodologia MDA tenta resolver este problema, colocando a modelagem no centro do processo de desenvolvimento, permitindo a geração automática do código fonte através da modelagem, reduzindo

o tempo de desenvolvimento e evitando que a modelagem fique defasada (SOUZA, 2008).

1.4 ESTRUTURA DO TRABALHO

O trabalho é constituído de 6 capítulos. O primeiro capítulo apresenta uma introdução sobre o assunto a ser tratado, assim como sua organização, objetivos e justificativas.

No capítulo 2 tem-se um breve conteúdo sobre os métodos tradicionais de desenvolvimento de *software* bem como os problemas e dificuldades encontrados para deixar a documentação atualizada com a codificação do sistema.

No capítulo 3 são apresentados conceitos, técnicas e benefícios do *framework* MDA.

O capítulo 4 apresenta o estudo de caso a ser implementado, utilizando a ferramenta AndroMDA.

O Capitulo 5 e 6 conclui este trabalho, apresentando os resultados e uma conclusão sobre o tema abordado.

.

2 DESENVOLVIMENTO DE SOFTWARE TRADICIONAL

2.1 MODELO EM CASCATA

O modelo em cascata, algumas vezes chamado de ciclo de vida clássico, sugere uma abordagem sistemática e sequencial para o desenvolvimento de *softwares* que começa com as especificações dos requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção e implantação, culminando na manutenção progressiva do *software* acabado (PRESSMAN, 2006).

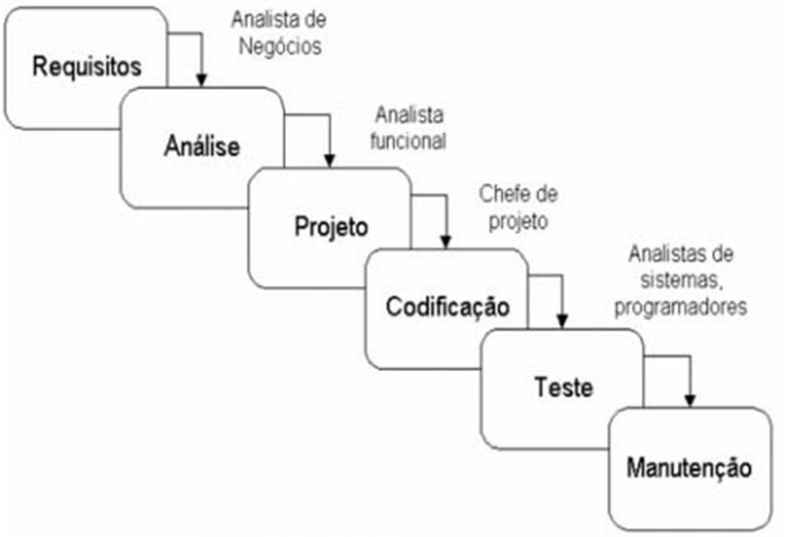


Figura 1 - Modelo em Cascata.

Fonte: Pressman (2006)

> O modelo em cascata é o paradigma mais antigo da engenharia de *software*. No entanto, nas ultimas décadas, a crítica a esse modelo de processo tem provocado, mesmo em seus mais ardentes adeptos, questionamentos sobre sua eficácia. Entre os problemas que são algumas vezes encontrados quando o modelo em cascata é aplicado estão (PRESSMAN, 2006):
>
> - Projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Apesar de o modelo linear poder acomodar a iteração, ele o faz indiretamente. Como resultado, as modificações podem causar confusão à medida que a equipe de projeto prossegue.
> - Em geral, é difícil para o cliente estabelecer todos os requisitos explicitamente. O modelo em cascata exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projetos.
> - O cliente precisa ter paciência. Uma versão executável dos programas não vai ficar disponível até o período final do intervalo de

> tempo do projeto. Um erro grosseiro pode ser desastroso se não for detectado até que o programa executável seja revisto.

Numa análise interessante de projetos reais, descobriu-se que a natureza linear do modelo em cascata leva a “estados de bloqueio” nos quais alguns membros da equipe de projeto precisam esperar que outros membros completem as tarefas dependentes. Na realidade, o tempo gasto em espera pode exceder o tempo gasto no trabalho produtivo. O estado de bloqueio tende a ocorrer mais no início e no fim de um processo sequencial linear (PRESSMAN, 2006).

O modelo em cascata tem uma pouca flexibilidade na divisão do projeto nos seus estágios de desenvolvimento. Os acordos devem ser feitos em um estágio inicial do processo, e isso significa que é difícil responder aos requisitos do cliente, que sempre se modificam. Portanto o modelo em cascata deve ser utilizado somente quando os requisitos forem bem compreendidos. Ele reflete a prática da engenharia. Consequentemente, os processos de software com base nessa abordagem ainda são utilizados no desenvolvimento de software (SOMMERVILLE, 2007).

2.2 MODELO INCREMENTAL

O modelo incremental aplica sequencias lineares se uma forma racional à medida que o tempo passa. Cada sequencia linear produz incrementos do *software* passíveis de serem entregues.

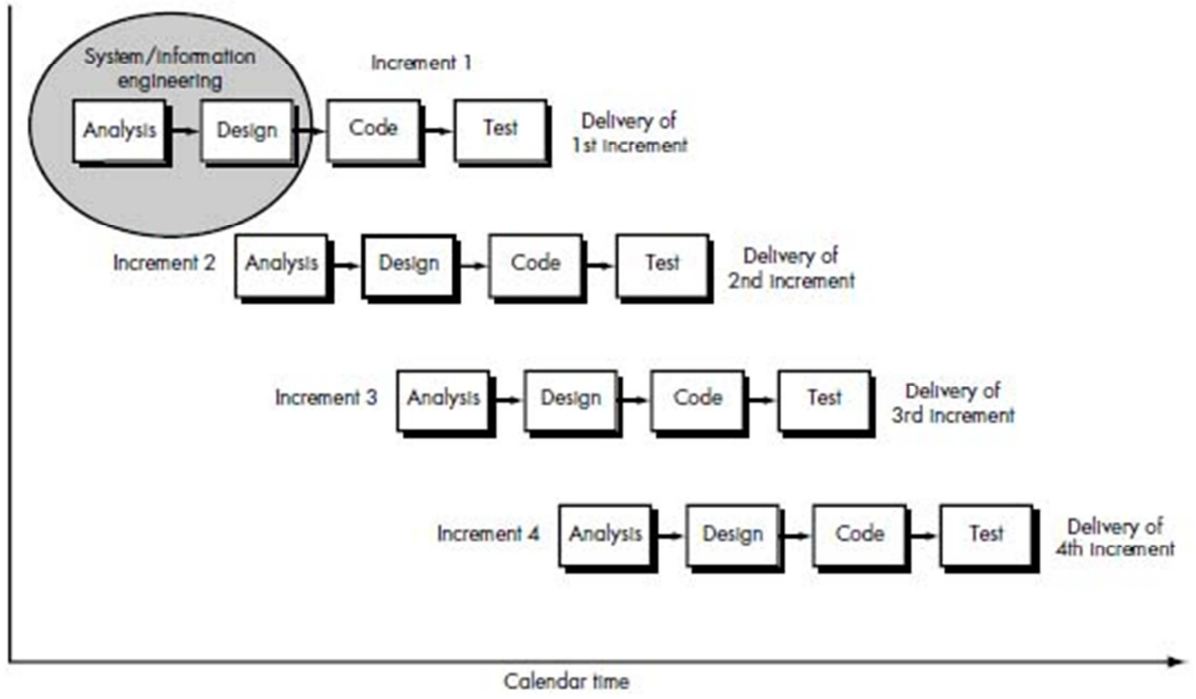


Figura 2 - Modelo Incremental

Fonte: Pressman (2006)

> Quando um modelo incremental é usado, o primeiro incremento é frequentemente chamado de núcleo do produto. Isto é, os requisitos básicos são satisfeitos, mas muitas características suplementares (algumas conhecidas, outras desconhecidas) deixam de ser elaboradas. O núcleo do produto é usado pelo cliente (ou passa por uma revisão detalhada). Um plano é desenvolvido para o próximo incremento como resultado do uso ou avaliação. O plano visa a modificação do núcleo do produto para melhor satisfazer às necessidades do cliente e à elaboração de características e funcionalidades adicionais. Esse processo é repetido após a realização de cada incremento, até que o produto completo seja produzido (PRESSMAN, 2006).

Em um processo de desenvolvimento incremental, os clientes identificam em um esboço, as funções a serem fornecidas pelo sistema. Eles identificam quais funções são mais importantes e quais são menos importantes. Em seguida é definida uma série de estágios de entrega, com cada estágio fornecendo um subconjunto das funcionalidades do sistema. A alocação de funções aos estágios depende da prioridade da função. As mais prioritárias são entregues primeiro ao cliente (SOMMERVILLE, 2007).

O desenvolvimento incremental é particularmente útil quando não há mão de obra disponível para uma implementação completa, dentro do prazo comercial de entrega estabelecido para o projeto. Os primeiros incrementos podem ser implementados com menos pessoas. Se o núcleo do produto for bem recebido,

então pessoal extra (se necessário) pode ser adicionado para implementar o próximo incremento. Além disso, os incrementos podem ser planejados para gerenciar os riscos técnicos. Por exemplo, um sistema importante pode exigir a disponibilidade de um *hardware* novo, que está em desenvolvimento, e cuja data de entrega é incerta. Pode ser possível planejar os primeiros incrementos de modo que seja evitado o uso desse *hardware*, permitindo, assim, que uma parte da funcionalidade seja entregue aos usuários finais sem demora excessiva (PRESSMAN, 2006).

2.3 MODELO ESPIRAL

O processo é representado como uma espiral, ilustrado na Figura 3. Cada espiral representa uma fase do processo de software. Assim, o *loop* mais interno pode estra relacionado à viabilidade do sistema; o *loop* seguinte à definição de requisitos do sistema; o próximo *loop*, ao projeto do sistema e assim por diante (SOMMERVILLE, 2007).

> Um modelo espiral é em um conjunto de atividades definidas pela equipe de engenharia de *software*. Cada uma das atividades representa um segmento de caminho espiral. À medida que esse processo evolucionário começa, a equipe de *software* realiza atividades que são indicadas por um circuito em volta da espiral em sentido horário, começando pelo centro. O risco é considerado a medida que cada evolução é feita. Os marcos de ancoragem – uma combinação de produtos de trabalho e condições que são obtidas ao longo do caminho espiral – são indicados para cada passagem evolucionária (PRESSMAN, 2006).

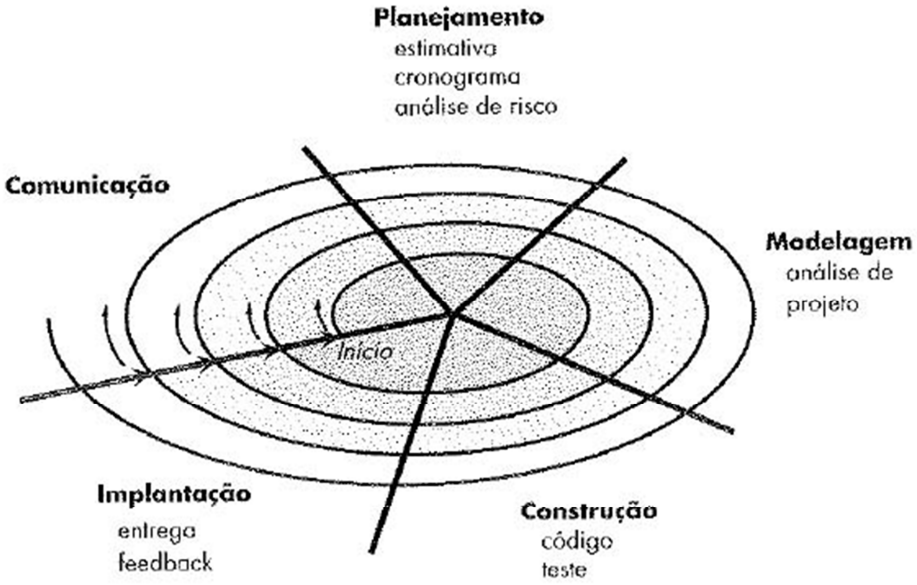


Figura 3 - Modelo Espiral.
Fonte: Pressman (2006)

Diferentemente de outros modelos de processo que terminam quando o *software* é entregue, o modelo espiral pode ser adaptado para aplicação ao longo da vida do *software*. Assim, o primeiro circuito em volta da espiral poderia representar um “projeto de desenvolvimento de conceitos” que começa no centro da espiral e continua por várias interações” até que o desenvolvimento do conceito seja completado. Se o conceito for desenvolvido em conceito real, o processo prossegue para fora na forma espiral e um “ projeto de desenvolvimento de um novo produto” começa. O novo produto vai evoluir por meio de um certo número de iterações, em torno da espiral. Depois, um circuito em volta da espiral poderia ser usado para representar um “projeto de aperfeiçoamento de produto”. Em resumo, a espiral, quando caracterizada desse modo, permanece até que o *software* seja retirado de serviço. Há momentos em que o processo fica adormecido, mas sempre que uma modificação é iniciada, o processo começa no ponto de entrada adequado (PRESSMAN, 2006).

Um ciclo da espiral começa com a elaboração de objetivos, como desempenho e funcionalidades. Meios alternativos de atingir esses objetivos e as restrições impostas a cada uma dessas alternativas são enumerados. Cada alternativa é examinada em relação a cada objetivo. Isso normalmente resulta na identificação de causas de riscos para o projeto. A próxima etapa é avaliar esses riscos por

atividades, como análises mais detalhadas, prototipação, simulação, entre outras. Uma vez avaliados os riscos, alguma parte do desenvolvimento é realizada, seguida por uma atividade de planejamento para a próxima fase do processo (SOMMERVILLE, 2007).

O modelo espiral é uma abordagem realista do desenvolvimento de sistemas de *softwares* de grande porte. Como o *software* evolui à medida que o processo avança, o desenvolvedor e o cliente entendem melhor e reagem aos riscos de cada nível evolucionário. O modelo espiral usa a prototipagem como um mecanismo de redução de riscos, porém, permite o desenvolver aplicar a abordagem de prototipagem em qualquer estágio da evolução do produto. Ele mantem a abordagem sistemática passo-a-passo, sugerida pelo ciclo de vida clássico, mais o incorpora em um arcabouço iterativo que reflete mais realisticamente o mundo real. O modelo espiral exige a consideração direta dos riscos técnicos em todos os estágios do projeto e, se aplicado adequadamente, deve reduzir os riscos antes que se tornem problemáticos (PRESSMAN, 2006).

2.4 O PROBLEMA DA PRODUTIVIDADE

Se for usado um modelo interativo e incremental, ou o modelo em cascata, documentos e diagramas serão produzidos durante as primeiras fases. Estes documentos incluem requisitos e diagramas da Linguagem de Modelagem Unificada (UML). Diagramas de casos de uso, diagramas de classe, diagramas de sequência e assim por diante. A pilha de papel produzido é muitas vezes impressionante. No entanto, a maioria dos artefatos destas fases é apenas papel e nada mais.

Os documentos e diagramas correspondentes, criados nas fases iniciais perdem rapidamente seu valor ao longo do tempo em que a codificação é produzida. Em vez de ser uma especificação exata do código, os diagramas geralmente tornam-se imagens não relacionadas. Mudanças muitas vezes são feitas apenas no nível do código só porque leva um tempo maior para atualizar os diagramas e outros documentos de alto nível. A documentação foi sempre um elo fraco no processo de desenvolvimento de *software*. Muitas vezes, é feito depois do desenvolvimento. Escrever documentação durante o desenvolvimento consome tempo e retarda o processo (KLEPPE, WARMER e BAST, 2003).

A manutenção do sistema torna-se mais difícil devido ao uso de documentação inadequada. Quando se realizam alterações, quase sempre se esquece de fazer as atualizações correspondentes na documentação. Essa então não mais reflete o sistema.

Então, ou utiliza-se o tempo nas primeiras fases de desenvolvimento de *software*, na construção de documentação e diagramas, ou usa-se o tempo na fase de manutenção, para descobrir o que o *software* está realmente fazendo. Escrever código é uma tarefa produtiva. Escrever documentação não é. Ainda assim, em um projeto de *software* maduro estas tarefas precisam ser feitas (KLEPPE, WARMER e BAST, 2003).

Projetistas experientes de aplicações usualmente investem mais tempo na construção de modelos do que em atividades de programação, pois a definição e o uso de modelos precisos e completos facilitam o desenvolvimento de sistemas corporativos dentro dos prazos e custos previstos, mesmo quando tais sistemas são grandes e complexos (MILLER e MUKERJI, 2001).

A documentação deve ser vista como uma ferramenta de auxílio, pois será utilizada nos momentos de necessidade por todas as pessoas envolvidas no sistema.

3 MODEL DRIVEN ARCHITECTURE

O Model Driven Architecture (MDA) é um *framework* para desenvolvimento de *software* definido pela Object Management Group (OMG). A chave para o MDA é a importância dos modelos no processo de desenvolvimento de *software*. O processo de desenvolvimento de *software* é impulsionado pela atividade de modelagem do sistema de *software*. MDA é capaz de especificar um sistema de modo independente da plataforma a ser adotada, especificar características de várias plataformas, permitir a escolha de uma plataforma particular e permitir a transformação da especificação deste sistema em uma implementação para a plataforma escolhida (MILLER e MUKERJI, 2001).

A MDA representa uma visão das necessidades de interoperabilidade expandida para abranger completamente o ciclo de desenvolvimento de aplicações, e permite que os desenvolvedores construam sistemas de acordo com a lógica de negócios e os dados existentes, independentemente de qualquer plataforma particular, cujos detalhes tecnológicos são considerados irrelevantes na definição dos aspectos essenciais da funcionalidade desejada (MILLER e MUKERJI, 2001).

O ciclo de vida de desenvolvimento do MDA, que é mostrado na Figura 4, não parece muito diferente do ciclo de vida tradicional. As mesmas fases são identificadas. Uma das grandes diferenças reside na natureza dos artefatos que são criados durante o processo de desenvolvimento. Os artefatos são modelos formais, ou seja, modelos que podem ser entendidos por computadores. (KLEPPE, WARMER e BAST, 2003).

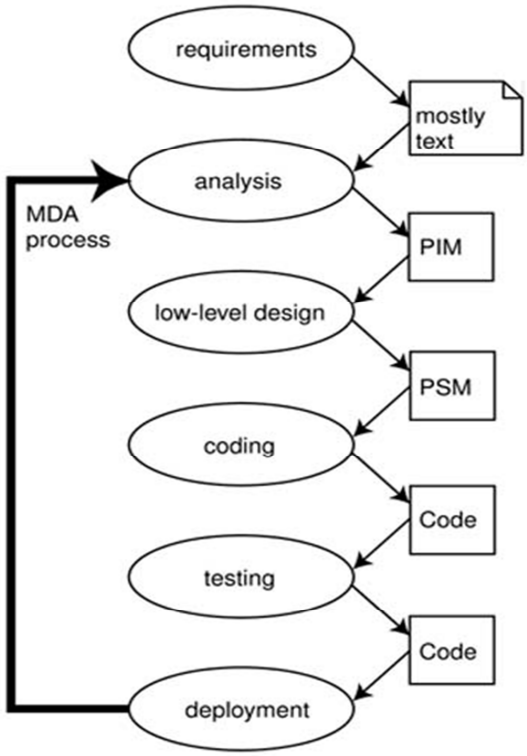


Figura 4 - Ciclo de vida do desenvolvimento MDA.
Fonte: Kleppe, Warmer e Bast (2003)

O *framework* MDA é baseado na Unified Modeling Language (UML). Mas diferentemente dos modelos puramente UML, a proposta da MDA é promover a criação de modelos abstratos que possam ser processados automaticamente, desenvolvidos independentemente das tecnologias de implementação e também armazenados em repositórios padronizados. Ferramentas apropriadas podem acessar tais modelos, transformando-os automaticamente em esquemas, esqueletos de código, scripts de implantação, entre outros elementos (KLEPPE, WARMER e BAST, 2003).

3.1 MODELOS

Bons modelos servem como meio de comunicação, eles são mais baratos de construir do que a coisa real, e eles adequam-se ao plano de ataque que a equipe leva para resolver o problema em questão. Modelos podem executar a gama de

esboços de projetos bastante detalhados e totalmente executáveis, todos são úteis no contexto apropriado (MELLOR, SCOTT, *et al.*, 2004).

Uma chave para a modelagem eficaz no contexto de desenvolvimento de sistemas é bom uso da abstração e classificação. Abstração envolve ignorar informações que não são de interesse em um contexto particular; classificação envolve agrupamento de informações importantes com base em propriedades comuns, mesmo que as coisas em estudo são, naturalmente, diferentes umas das outras (MELLOR, SCOTT, *et al.*, 2004).

A base do MDA é a noção de criação de diferentes modelos em diferentes níveis de abstração e, em seguida, liga-los juntos para formar uma implementação. Alguns destes modelos vão existir independentes de plataformas de *software*, enquanto outros serão específicos para plataformas específicas. Cada modelo utiliza uma combinação de texto e vários diagramas complementares e inter-relacionados (MELLOR, SCOTT, *et al.*, 2004).

Para permitir a transformação automática de um modelo, é preciso restringir os modelos adequados para o MDA aos modelos que são escritos em uma linguagem bem definida. Uma linguagem bem definida tem um significado que pode ser interpretado automaticamente por um computador (KLEPPE, WARMER e BAST, 2003).

Para um dado sistema podem haver muitos modelos diferentes, que variam nos detalhes. É óbvio que dois modelos do mesmo sistema têm uma certa relação. Existem muitos tipos de relacionamentos entre os modelos. Por exemplo, um modelo pode descrever apenas uma parte do sistema completo, enquanto um outro modelo descreve outra, possivelmente sobrepostas. Um modelo pode descrever o sistema com mais detalhe do que outro. Um modelo pode descrever o sistema a partir de um ângulo completamente diferente do outro (KLEPPE, WARMER e BAST, 2003).

3.2 O PAPEL DA UML NO MDA

A Unified Modeling Language (UML) é a linguagem de modelagem padrão para o MDA, entretanto o MDA também permite o uso de outras linguagens de modelagem. O mecanismo de Perfil UML em particular é muito usado no MDA para introduzir anotações específicas da plataforma. Ele fornece um mecanismo de

extensão leve para UML que permite refinar a semântica UML. Isto significa que cada perfil tem que respeitar a semântica geral UML. Ele utiliza estereótipos para atribuir um significado especial para os elementos do modelo designado. Sempre que um estereótipo é aplicado a um elemento do modelo, isso é mostrado por um ícone ou o nome entre aspas *Stereotype*. A Figura 5 ilustra o uso de estereótipos no diagrama de classe (WAGELAAR, 2008).

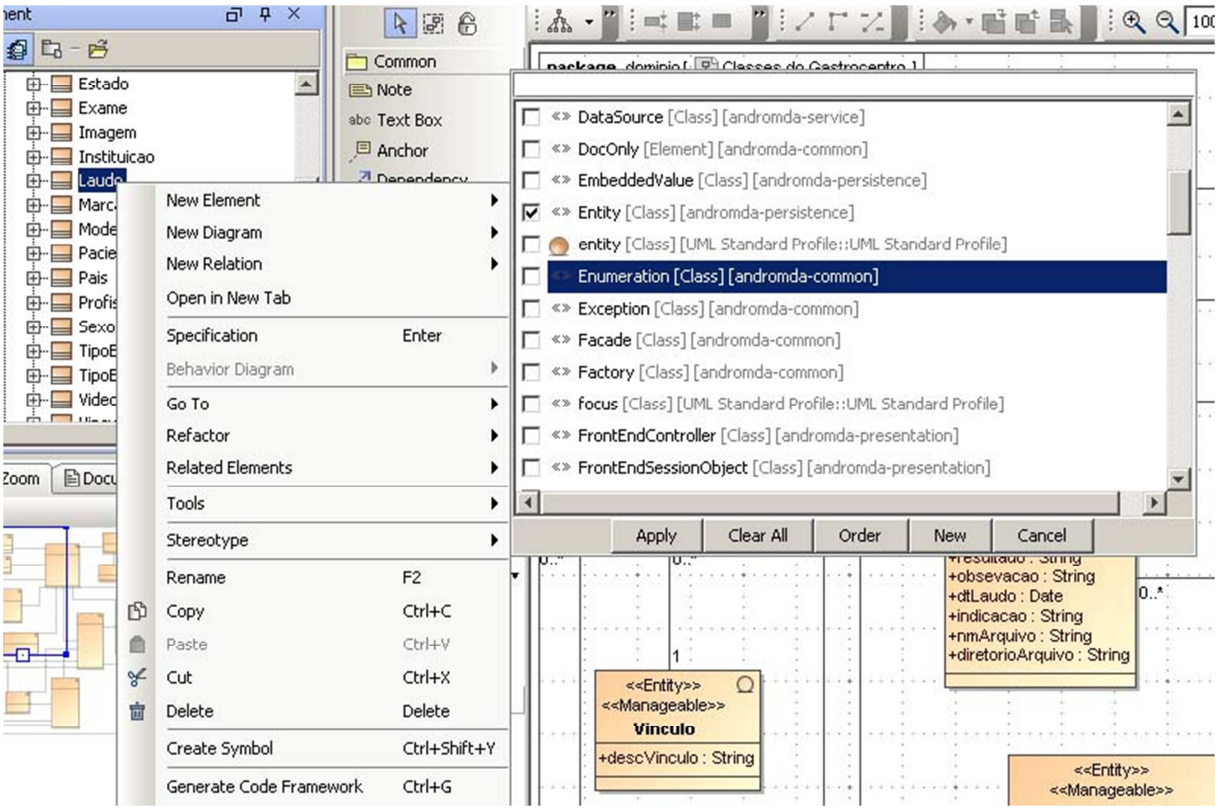


Figura 5 - MagicDraw - Estereótipos

3.3 NÍVEIS DO MDA

MDA recomenda determinados modelos. A OMG descreve diferentes níveis e suas relações, mas não especifica como criar estes níveis abstratos e quais os modelos exatos e notações a serem usados para a sua representação e como transformá-los um com o outro. Existem algumas recomendações de vários pesquisadores da área que podem ser semelhantes em alguns pontos e diferentes em outros (KARDOS e DROZDOVÁ, 2010).

Níveis básicos de abstração são chamados de: *Computation Independent Model* (CIM), *Platform Independent Model* (PIM), *Platform Specific Model* (PSM) e

Implementation model (IM) - Código. No processo de desenvolvimento MDA os modelos são criados de acordo com a ordem descrita na Figura 6.

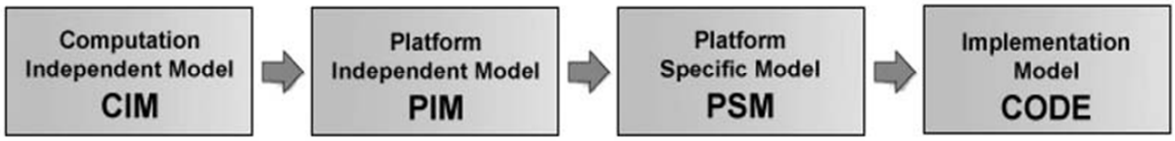


Figura 6 - Níveis básicos de abstração em MDA.
Fonte: Kardos e Drozdová (2010)

3.3.1 Computation Independent Model (CIM)

CIM é o nível que não apresenta detalhes da construção do Sistema, mas especifica as atividades que estão sendo processados no Sistema. Em outras palavras, este nível representa os processos de negócio da organização para a qual o sistema será desenvolvido. CIM é feito para os analistas no nível superior que pode ser analistas de negócios, especialistas de domínio ou usuários de domínio do sistema. CIM é às vezes chamado de modelo de domínio. CIM não tem uma informação sobre modelos ou artefatos que serão utilizados para a implementação do Sistema. Ele descreve o ambiente em que opera o Sistema e ajuda a reconhecer o que esperamos do Sistema. É útil não apenas como uma ajuda para compreender os problemas do projeto, mas, assim como um "vocabulário" para o uso desse sistema em outros sistemas. Na especificação MDA os requisitos de CIM devem ter relações com PIM e PSM e vice-versa (KARDOS e DROZDOVÁ, 2010).

3.3.2 Platform Independent Model (PIM)

Todos os projetos de desenvolvimento MDA começam com a criação de um modelo independente de plataforma (PIM). Um modelo MDA terá vários níveis de PIMs. Embora que todos sejam independentes de qualquer plataforma em particular, cada um, exceto o modelo básico, inclui aspectos do comportamento tecnológico (SIEGEL, 2001).

PIM mostra certo nível de independência de tal forma que os modelos que estão representados lá (na sua maioria modelos UML), sejam devidamente escolhidos para uso em diversas plataformas. PIM descreve o sistema, mas esconde detalhes concretos no uso de tecnologia. PIM cria especificações para os serviços necessários do sistema sem detalhes técnicos de plataforma dependente. Analistas de software estão envolvidos na criação deste nível de MDA (KARDOS e DROZDOVÁ, 2010).

Por causa de sua independência tecnológica, o PIM de base mantém todo o seu valor ao longo dos anos, exigindo mudanças apenas quando tem alterações de regras de negócio (SIEGEL, 2001).

3.3.3 Platform Specific Model (PSM)

Uma vez que a primeira iteração do seu PIM está completa, ele é armazenado na *Meta-Object Facility* (MOF) e é dada a entrada para a etapa de mapeamento, que irá produzir um modelo específico de plataforma (PSM). Especializações e extensões UML dão-lhe o poder de expressar tanto PIMs e PSMs. Denominado um perfil UML, um conjunto padronizado de extensões (que consiste em estereótipos e valores etiquetados) define um ambiente UML sob medida para uma utilização específica (SIEGEL, 2001).

PSM conecta especificação de PIM com detalhes que especifica que tipo de plataforma será usada pelo sistema. PSM pode oferecer mais ou menos especificações técnicas detalhadas, dependendo da sua finalidade. PSM será realizado pela transformação em um modelo de implementação, código que define todas as informações para a criação de Informações do sistema e seu lançamento. Este nível é criado por desenvolvedores de software. A idéia básica do PSM é realizar a transformação do PIM em modelos PSM (KARDOS e DROZDOVÁ, 2010).

3.3.4 Código

O código é o produto final da MDA e deve ser o resultado da transformação de um dado PSM considerando uma tecnologia de implementação específica. A geração de código é uma etapa relativamente simples dada a proximidade do PSM com a tecnologia particular em uso. Em algumas circunstâncias, quando existir suporte para múltiplas linguagens de programação, deverá ser efetuada a seleção da linguagem de implementação desejada. Além do código também poderão ser gerados outros artefatos necessários ao sistema, tais como arquivos de configuração, entradas de registro, scripts, etc. Após a geração do código será provável a necessidade de alguns ajustes e complementações, que deverão ser feitos por uma equipe de programação, mas espera-se que em quantidades bastante menores do que com as atuais ferramentas (KLEPPE, WARMER e BAST, 2003).

3.4 TRANSFORMAÇÃO AUTOMÁTICA

O grande diferencial da MDA reside na forma de realização das transformações entre os modelos PIM, PSM e o código. Tradicionalmente as transformações entre modelos de um processo de software são realizadas manualmente. Diferentemente no MDA, os modelos deverão ser usados para geração automática da maior parte do sistema (SIEGEL, 2001). No MDA todas as transformações devem ser realizadas automaticamente por ferramentas apropriadas, o que pode significar maior rapidez e flexibilidade na geração de aplicações de melhor qualidade, caracterizando assim os benefícios imediatos de sua aplicação. As transformações entre modelos ou mapeamento (*mappings*) são entendidas como o conjunto de regras e técnicas aplicadas em um modelo de modo que seja obtido um outro com as características desejadas. A MDA considera a existência de cinco tipos de transformações diferentes (MILLER e MUKERJI, 2001):

- PIM para PIM. Utilizada para o aperfeiçoamento ou simplificação dos modelos sem a necessidade de levar em conta aspectos dependentes de plataforma.

- PIM para PSM. Transformação “padrão” do modelo independente de plataforma para outro específico durante o ciclo de desenvolvimento típico de aplicações.
- PSM para PSM. Esta transformação permite a migração da solução entre plataformas diferentes, bem como o direcionamento de partes da implementação para tecnologias específicas, usadas por questões de interoperabilidade ou benefícios obtidos através do uso de certas plataformas.
- PSM para PIM. Quando é necessário obter-se uma representação neutra em termos de tecnologia de soluções específicas.
- PSM para CODE. Transformação “padrão” do modelo específico de plataforma para o código fonte.

Os principais passos no processo de desenvolvimento MDA são mostrados na Figura 7.

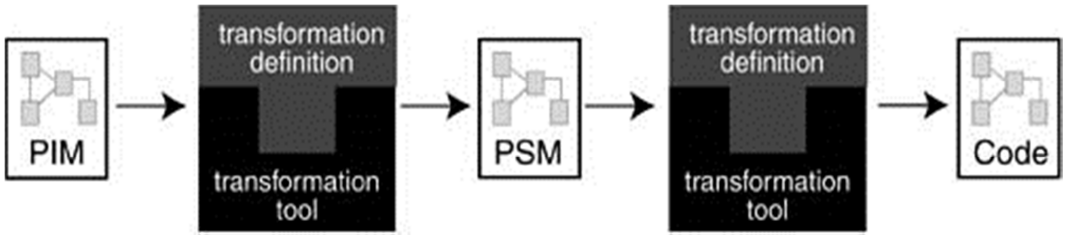


Figura 7 - Os três principais passos no processo de desenvolvimento MDA.
Fonte: Kleppe, Warmer e Bast (2003)

Entre as transformações pode existir o processo de marcação (*marking*), que constitui uma forma “leve” e pouco intrusiva de extensões dos modelos com elementos voltados a facilitar uma transformação particular (MELLOR et al., 2004). Por exemplo, um PIM (sem marcações) pode receber anotações destinadas a uma plataforma A ou B, originando *marked* PIMs e mantendo o PIM original sem qualquer “contaminação”.

Embora hoje existam muitas ferramentas capazes de gerar algum código a partir de modelos particulares, usualmente tal código é pouco mais que um “esqueleto” (um template), exigindo que a finalização da implementação seja feita através de atividades de programação convencional (KLEPPE, WARMER e BAST, 2003).

A geração de código não é considerada o aspecto mais importante da MDA , pois este é bastante próximo à estrutura declarativa e atributos, sendo simples a

estrutura funcional de métodos de acesso (*setter and getter methods*); por outro lado é bastante complexa a geração das características comportamentais do sistema como um todo. Exatamente por isso, projetistas que utilizam a MDA não devem esperar que a primeira versão gerada para o sistema seja perfeita, pois o próprio processo MDA assume a necessidade de múltiplas iterações entre o projeto e a implementação obtida, ou seja, a necessidade de refinamento como meio de produzir sistemas de qualidade (MILLER e MUKERJI, 2001).

3.5 BENEFÍCIOS DO MDA

3.5.1 Produtividade

No MDA o foco para um desenvolvedor se desloca para o desenvolvimento de um PIM. Os PSMs que são necessários são gerados por uma transformação de PIM para PSM. É claro, alguém ainda precisa definir a transformação exata, que é uma tarefa difícil e especializada. Mas essa transformação só precisa ser definida uma vez e pode então ser aplicado no desenvolvimento de muitos sistemas (KLEPPE, WARMER e BAST, 2003).

A maioria dos desenvolvedores se concentrará no desenvolvimento de PIMs. Uma vez que eles podem trabalhar de forma independente dos detalhes das plataformas, há um monte de detalhes técnicos que eles não precisam se preocupar. Estes detalhes técnicos serão adicionados automaticamente pelas transformações PIM para PSM. Isso melhora a produtividade de duas maneiras (KLEPPE, WARMER e BAST, 2003).

Em primeiro lugar, os desenvolvedores do PIM têm menos trabalho a fazer, porque detalhes específicos da plataforma não precisam ser concebidos e escritos, pois eles já estão abordados na definição de transformação. No nível PSM e no código, há muito menos código a ser escrito, porque uma grande quantidade de código já foi gerada a partir do PIM.

A segunda melhoria vem do fato de que os desenvolvedores possam mudar o foco do código para o PIM, assim prestando mais atenção para resolver o problema

em questão. Isso resulta em um sistema que se encaixa muito melhor com as necessidades dos usuários finais. Os usuários finais obtêm uma melhor funcionalidade em menos tempo.

Tal ganho de produtividade só pode ser alcançado através da utilização de ferramentas que automatizam totalmente a geração de um PSM a partir de um PIM (KLEPPE, WARMER e BAST, 2003).

3.5.2 Portabilidade

Como um mesmo PIM pode ser transformado em diferentes PSMs, possibilitando que um sistema possa operar em diferentes plataformas, então a construção do PIM é portável para qualquer plataforma para a qual exista a definição da transformação PIM para PSM especificamente envolvida. Para plataformas menos populares é possível a definição particular das transformações desejadas, assim como o surgimento de novas plataformas irá requerer apenas a definição do mapeamento específico, o que em tese maximiza a portabilidade de qualquer PIM com relação às tecnologias atuais e futuras (SOLEY, 2000).

A portabilidade está limitada as ferramentas de geração automática que estão disponíveis. Para as plataformas mais populares existe um grande número de ferramentas, para as menos populares deve-se ter uma ferramenta que suporte um plugin para definição de transformações e essas definições devem ser descritas manualmente. Para novas tecnologias e plataformas que irão chegar no futuro a industria de software precisa disponibilizar os mapeamentos de transformações a tempo. Isso nos permite rapidamente desenvolver novos sistemas com novas tecnologias baseadas nos modelos de PIMs já existentes (VERNER e PUIA, 2004).

3.5.3 Interoperabilidade

Múltiplas PSMs geradas a partir de um PIM pode ter relações. Em MDA estes são chamados de *bridges*. Quando PSMs são direcionados a diferentes plataformas, eles não podem falar diretamente uns com os outros. Uma forma ou de outra,

precisamos transformar conceitos de uma plataforma em conceitos usados em outra plataforma. Isto é tudo o que a interoperabilidade é. MDA resolve esse problema, gerando não só os PSMs, mas as pontes necessárias entre eles também, mostrado na Figura 8 (KLEPPE, WARMER e BAST, 2003).

De forma que é possível transformar um PIM em dois PSMs orientados para duas plataformas, todas as informações que precisamos para fazer a ponte entre os dois PSMs está disponível. Para cada elemento em um PSM sabemos de qual elemento do PIM tem sido transformada. A partir do elemento PIM sabemos qual é o elemento correspondente no segundo PSM. Podemos, portanto, deduzir como elementos de um PSM se relacionam com elementos no segundo PSM. Desde que nós também sabemos todas as especificações da plataforma e os detalhes técnicos de ambos os PSMs (caso contrário, não poderia ter realizado as transformações PIM para PSM), temos todas as informações que precisamos para gerar uma *bridge* entre os dois PSMs (KLEPPE, WARMER e BAST, 2003).

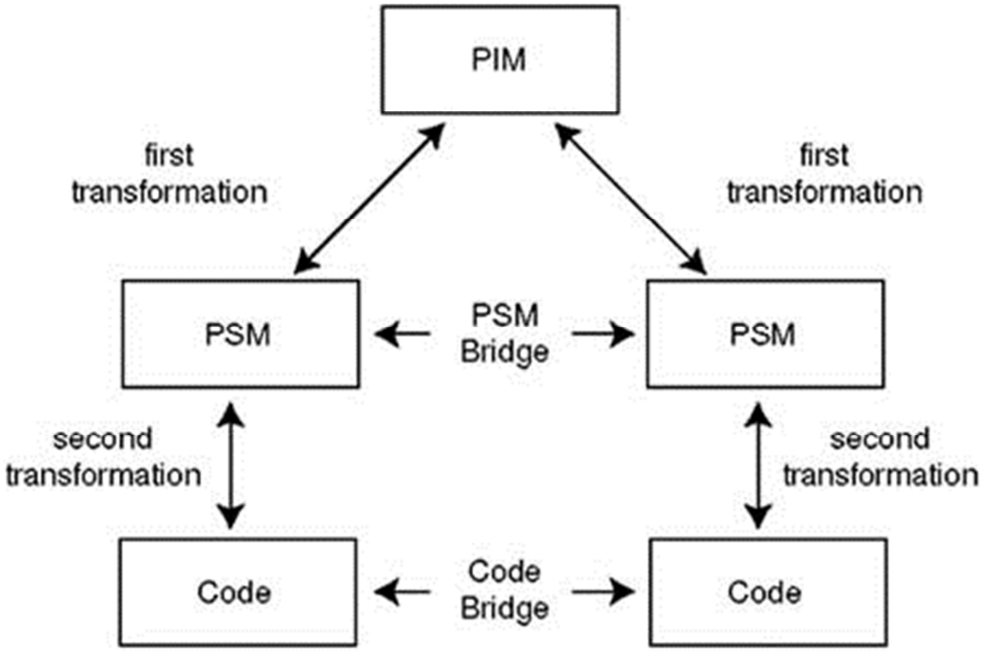


Figura 8 - MDA usando bridges de interoperabilidade.
Fonte: Kleppe, Warmer e Bast (2003)

Interoperabilidade entre plataformas pode ser realizada por ferramentas que não apenas geram PSMs, mas as pontes entre eles, e possivelmente para outras plataformas.

3.5.4 Manutenabilidade.

Através de alterações no PIM do sistema é possível efetuar a geração de novos PSMs e código correspondente muito rapidamente, agilizando e barateando os procedimentos de manutenção do sistema. Com isto, correções, adaptações ou mesmo a adição de novas funcionalidades tornam-se tarefas mais simples de serem realizadas, prolongando a vida útil do sistema. (SOLEY, 2000).

O PIM não é abandonado depois de escrever. As alterações feitas no sistema acabarão sendo feitas por alteração no PIM e regeneração do PSM e do código. Na prática, hoje, muitas das mudanças são feitas para o PSM, e o código é regenerado de lá. Boas ferramentas, no entanto, serão capazes de manter a relação entre a PIM e PSM, mesmo quando mudanças são feitas no PSM. Portanto, mudanças no PSM serão refletidas no PIM, e o alto nível documentação continuará a ser coerente com o código real (KLEPPE, WARMER e BAST, 2003).

3.5.5 Documentação

MDA é baseado na construção de modelos formais, que sob muitos aspectos correspondem a uma importante documentação do sistema. O PIM é o artefato mais importante, pois corresponde a uma documentação de alto nível. Além disso, como tais modelos podem ser visualizados, armazenados e processados automaticamente, não são abandonados após a finalização do sistema, pois tanto o PIM, no nível de abstração mais alto, quanto o PSM, num nível de abstração intermediário, podem ser reutilizados para a incorporação de alterações no sistema ou mesmo sua migração para outras plataformas. Embora a formalização dos modelos necessários a MDA cumpra o papel de documentação do sistema, outras informações deverão ser adicionadas aos modelos, tais como os problemas e necessidades diagnosticadas (KLEPPE, WARMER e BAST, 2003)

Os modelos da MDA são exibidos num maior ou menor nível de detalhes, permitindo a análise, conversão e comparação de modelos. Além disso, o mapeamento explícito entre os modelos, que possibilita sua transformação

automática, permite a rastreabilidade e controle de versão dos artefatos utilizados (MILLER e MUKERJI, 2001).

A modelagem em alto nível favorece as tarefas de validação, pois os detalhes de implementação não estão inclusos no PIM; que a aplicação ou integração de plataformas diferentes tornam-se mais claramente definidas, facilitando a produção de implementações particulares (MILLER e MUKERJI, 2001).

Tais benefícios se refletem diretamente na redução dos custos e de tempo de desenvolvimento, aumento da qualidade das aplicações produzidas, aumento do retorno dos investimentos realizados e aceleração do processo de adoção de novas tecnologias, bem como simplificação dos problemas associados com a integração de sistemas.

4 PROCEDIMENTOS METODOLÓGICOS DO ESTUDO

4.1 ESCOPO DA APLICAÇÃO

O exemplo utilizado neste trabalho refere-se às operações CRUD de um sistema de uma clínica médica. O sistema efetua o gerenciamento de informações sobre pacientes, profissionais da área da saúde, exames e laudos. Na parte de gerenciamento de exames, o sistema inclui a captura de imagens e vídeos, durante a sua realização. Permite também o acesso, por meio da Internet, aos dados de pacientes e aos exames por profissionais autenticados, possibilitando o acompanhamento de exames e disponibilizando recursos tecnológicos no auxílio à realização de diagnósticos à distância. As principais características são: (MACHADO, LEE, *et al.*, 2011).

- Acessibilidade do sistema deve ser feita somente por profissionais cadastrados e com permissão para utilizar o sistema;
- Manutenibilidade das informações sobre profissionais da área da saúde, pacientes e exames, aplicando-se critérios eficientes de segurança e de privacidade;
- Disponibilidade para a utilização da solução por meio da Internet;
- Permissão do armazenamento e do gerenciamento de informações relativas à pacientes por meio de registros históricos sobre o paciente e exames clínicos realizados;
- Permissão do cadastro e da manutenção de dados sobre profissionais médicos e pesquisadores, os quais possuem acesso ao sistema para a realização e/ou para a análise de exames;
- Capacidade de gerenciamento de dados, imagens e videos capturados durante a realização de exames de colonoscopia;
- Implementação de recursos que permitam a comunicação do sistema com o videocolonoscópio, oferecendo mecanismos para a captura e o armazenamento de imagens e videos dos exames durante a sua realização;

- Implementaçào de recursos que permitam a transmissão e acompanhamento dos exames em tempo real via streaming.
- Disponibilização de interface amigável para a realização e acompanhamento dos exames de colonoscopia;
- Implementação de funcionalidades que permitam a visibilidade e a análise dos exames realizados, após o término dos mesmos. Para o projeto deste requisito deve-se considerar a utilização remota do sistema, por meio de navegador e conexão com a Internet.

O diagrama de caso de uso do estudo de caso pode ser visualizado na Figura 9 e o diagrama de classe pode ser visualizado no ANEXO A

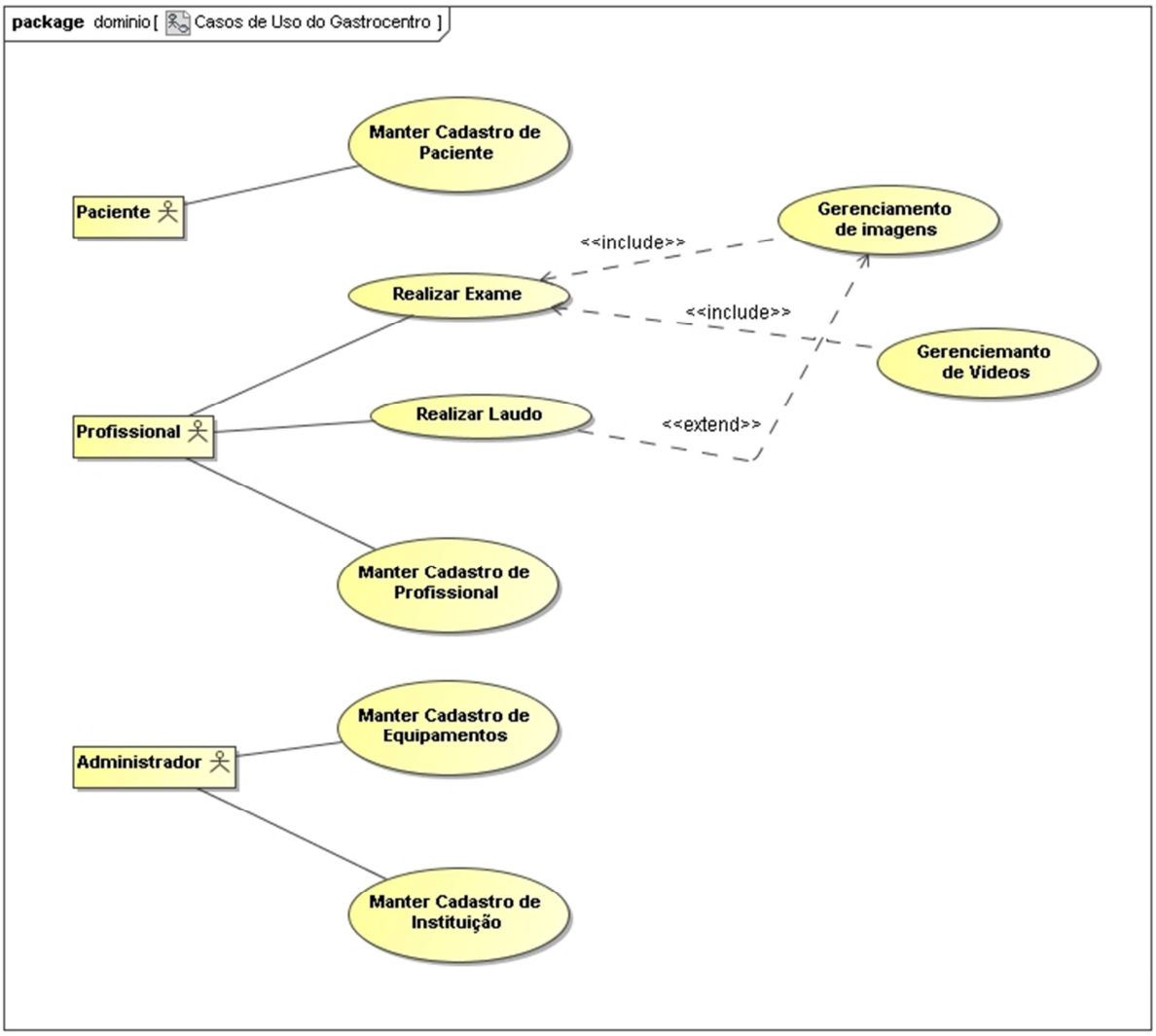


Figura 9 - Diagrama de Casos de uso Geral do estudo de caso.

Fonte: MACHADO, LEE, et al. (2011).

4.2 TECNOLOGIAS UTILIZADAS

4.2.1 Apache Maven

Apache Maven é uma ferramenta de gestão de projeto de *software*. Baseado no conceito de *Project Object Model* (POM), Maven pode gerenciar a documentação e a construção de um projeto a partir de uma peça central de informação.

Maven permite que um projeto seja construído a partir do POM. Utiliza um conjunto de *plugins* que são compartilhados por todos os projetos Maven, proporcionando um sistema de compilação uniforme (APACHE MAVEN, 2002).

É um projeto da Apache Software Foundation que inicialmente começou como uma tentativa de simplificar os processos de construção do projeto Jakarta Turbine. Necesitavam de uma forma padrão para construir os projetos, uma definição clara do que o projeto consistia, uma maneira fácil de publicar informações sobre o projeto e uma maneira de compartilhar JARs em vários projetos.

O resultado é uma ferramenta que pode ser usado para a construção e gerenciamento de qualquer projeto baseado em Java. Algo que vai fazer o trabalho do dia-a-dia dos desenvolvedores mais fácil e, geralmente, ajudar com a compreensão de qualquer projeto baseado em Java (APACHE MAVEN, 2002).

4.2.2 AndroMDA

AndroMDA é um *framework open source* para desenvolvimento MDA. É utilizado para a criação do projeto básico e geração de artefatos a partir dos modelos UML. A utilização do Andromda permite que grande parte do código necessário para a execução da aplicação seja gerado sem a interferência manual, acelerando muito o processo de desenvolvimento e garantindo a qualidade do código através da padronização do código e do uso de muitos dos "*design patterns*" reconhecidos.

AndroMDA compreende um conjunto de cartuchos com diversas soluções de projeto e arquitetura incorporadas nos procedimentos de transformação de modelos seguindo a abordagem MDA. Pode-se optar por gerar o código para o Hibernate,

EJB, Spring, WebServices, e Struts. O código gerado é automaticamente integrado no processo de construção, mantendo seu foco na lógica de negócios. A Figura 10 mostra como o AndroMDA separa a geração por cartuchos (ANDROMDA, 2003).

AndroMDA é basicamente um *plugin* que permite executar projetos MDA dentro da ferramenta Apache Maven.

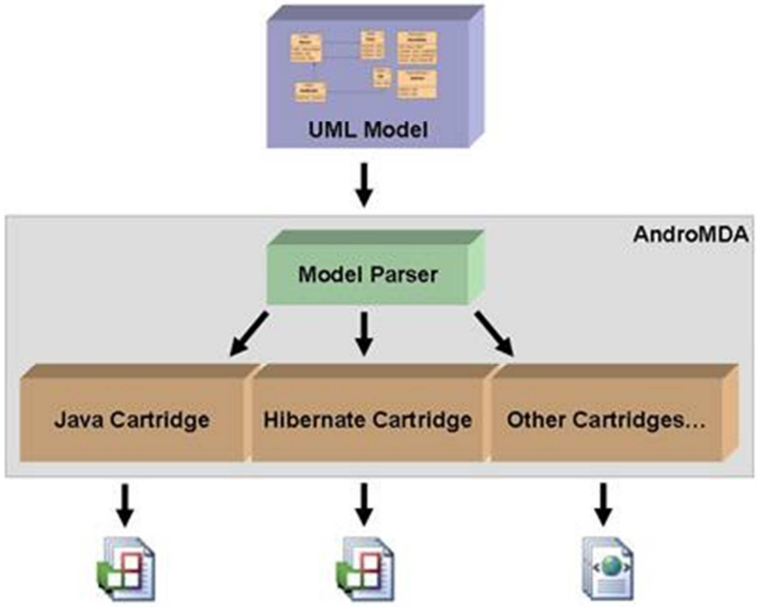


Figura 10 - Geração separada por cartuchos.
Fonte: Getting Started AndroMDA (2003)

4.2.3 MagicDraw UML

MagicDraw é uma ferramenta desenvolvida pela No Magic Inc para modelagem, processos de negócios e arquitetura de software. Uma ferramenta de desenvolvimento dinâmico e versátil que facilita a análise e projeto de orientada a objetos, bem como modelagem de banco de dados, a geração de DDL e engenharia reversa (NO MAGIC INC, 2000).

4.3 INSTALAÇÃO E CONFIGURAÇÃO DO ANDROMDA

O processo de instalação inicia-se com o download do Apache Maven e com a descompactação em um diretório. Após a descompactação é necessário configurar as variáveis de ambiente no sistema operacional. As variáveis necessárias são mostradas no Quadro 1.

Propriedade	Valor
JAVA_HOME	Diretório de instalação Java. Ex. `C:\Program Files\Java\jdk1.6.0_31`
M2_HOME	Diretório de instalação Maven. Ex. `C:\Programs\maven3`
PATH	`%JAVA_HOME%\bin;%M2_HOME%\bin`

Quadro 1 - Variáveis de Ambiente

Na Figura 11 é mostrado o processo de configuração no sistema operacinal.

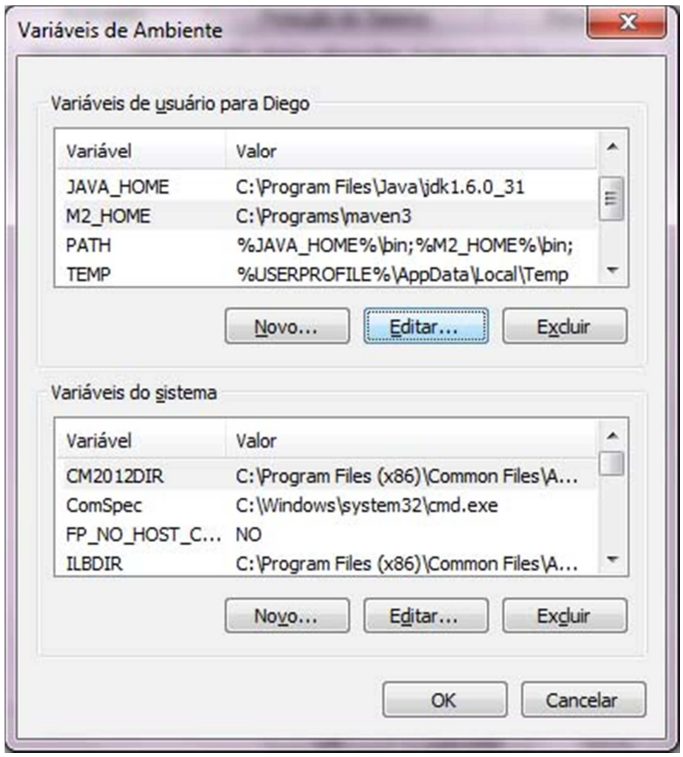


Figura 11 - Variáveis de ambiente

Para certificar-se que a configuração está correta deve-se abrir o *prompt*[1] de comando e executar `mvn –v`. O comando exibe a versão do Maven conforme mostrado na Figura 12.

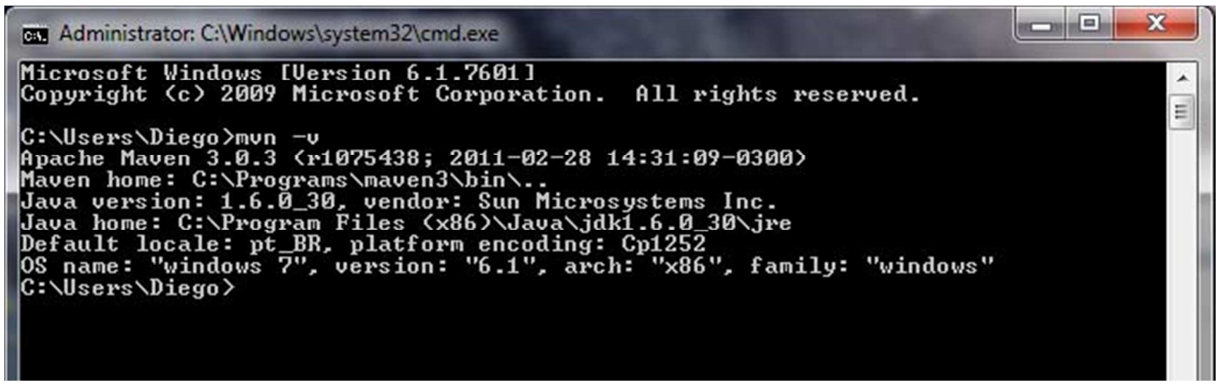


Figura 12 - Versão do Apache Maven

4.4 CRIAÇÃO DO PROJETO BASE

Para iniciar a criação de um projeto MDA, deve-se executar o gerador de projetos Maven do AndroMDA através do *prompt* de comando.

```
Mvn org.andromda.maven.plugins:andromdapp-maven-plugin:3.4-
SNAPSHOT:generate
```

Quadro 2 - Comando gerador de projetos Maven do AndroMDA

Neste instante o maven faz algumas perguntas de configuração de projeto, tais como:

- Diretório da aplicação;
- Autor do Projeto;
- Versão da UML;
- Nome do Projeto;
- Nome do pacote;

[1] Interpretador de linha de comando de sistemas baseados no Windows NT

- Tipo de aplicação;
- Framework de persistência;
- Banco de dados.

O gerador de projetos maven é ilustrado na Figura 13.

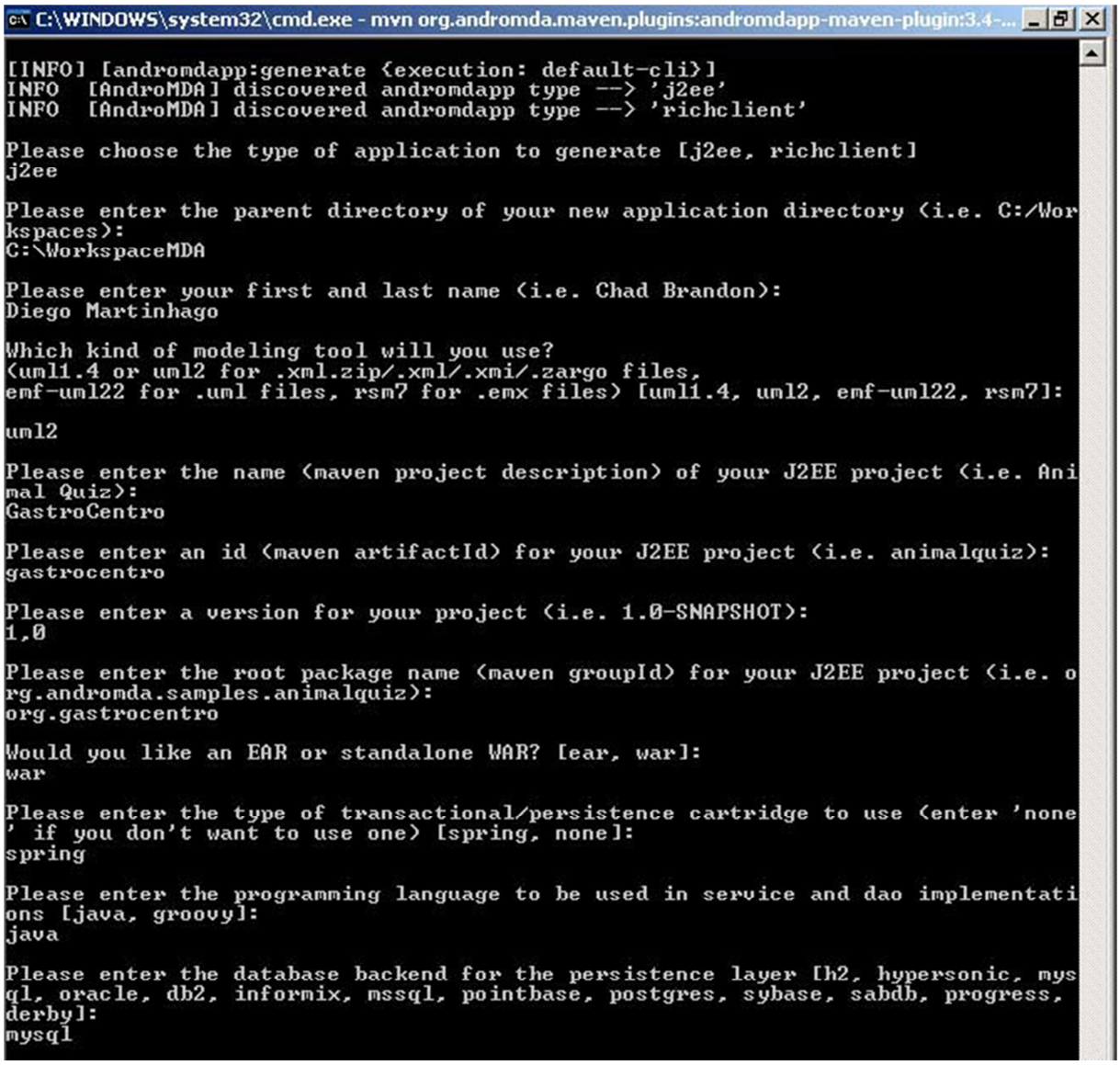


Figura 13 - AndroMDA - Configuração de Projeto

4.4.1 Estrutura do projeto

O projeto AndroMDA é subdividido em subprojetos. Na pasta do projeto são criadas as pastas *common*, *core*, *web* e *mda*, onde cada pasta representa um subprojeto.

O subprojeto *core* contém os artefatos relativos à camada de negócios: classes de serviço, classes de persistência, e configurações que permitem o seu funcionamento e acesso a outras camadas.

No subprojeto *common* ficam os artefatos gerados para comunicação entre as camadas web e core, tais como interfaces, *value objects*, e *enums*. Nesse projeto só existem artefatos que sempre são gerados no ato da compilação.

O subprojeto *web* é responsável pelo empacotamento da aplicação, contendo todas as configurações e dependências para a aplicação ser executada. Neste subprojeto ficam as classes de controle e as configurações do *Java server Faces*, *Facelets* e outros, para a renderização das páginas web.

O subprojeto *mda* contém a configuração do processo de geração de artefatos pela ferramenta AndroMDA. Neste projeto constam arquivos de fonte UML, arquivos de configuração e mapeamentos para geração de artefatos com características específicas do projeto.

4.5 IMPLEMENTANDO CASOS DE USO

Deve-se utilizar uma ferramenta que permitirá modelar as aplicações em UML e exportar estes modelos em um formato que o AndroMDA possa entender.

Para iniciar o processo de desenvolvimento dos diagramas, deve-se editar a arquivo fonte UML que está no subprojeto *mda*, /mda/src/main/uml/nomeArquivo.xml.

Nas classes persistentes deve-se adicionar o estereótipo `<<Entity>>` para que sejam gerados códigos para o mapeamento objeto-relacional. Através de um plugin adicional o Maven gera e executa o script de criação das tabelas associadas a essas entidades.

Quando adiciona-se o estereótipo <<Manageable>>, o Andromda gera códigos adicionais que permitem ao usuário final a manipulação direta dos dados dessa entidade, ou seja: a aplicação terá telas e código para a inserção, edição e remoção de instâncias dessas entidades. A Figura 14 mostra a adição dos estereótipos na classe.

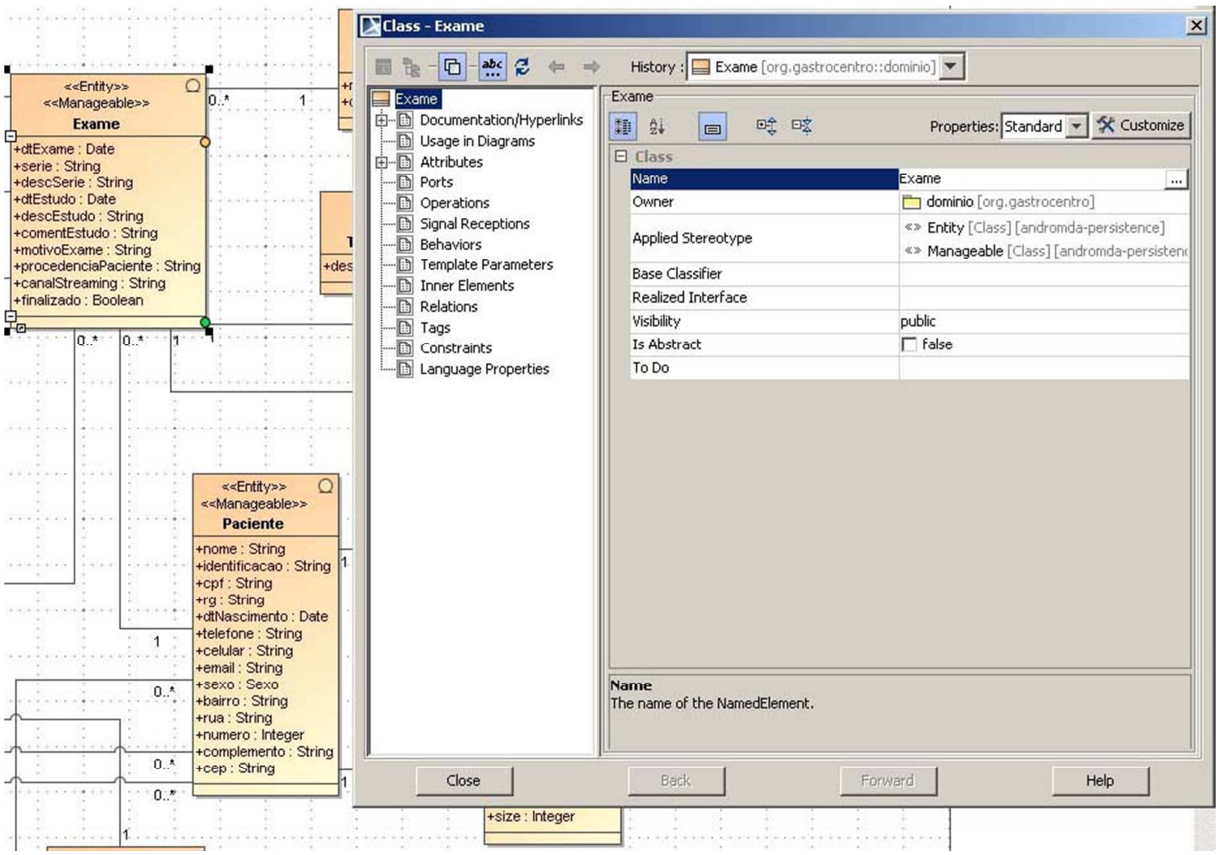


Figura 14 - MagicDraw - Propiedades da classe

4.6 GERAÇÃO DO CÓDIGO FONTE

Para iniciar o processo de geração deve-se exportar os diagramas UML para o formato EMF UML2(v2.x) XMI File, ilustrado na Figura 15. Após a exportação é executado na pasta raiz do projeto o comando `mvn` com o *prompt* de comando, para que os arquivos UML sejam compilados pelo AndroMDA.

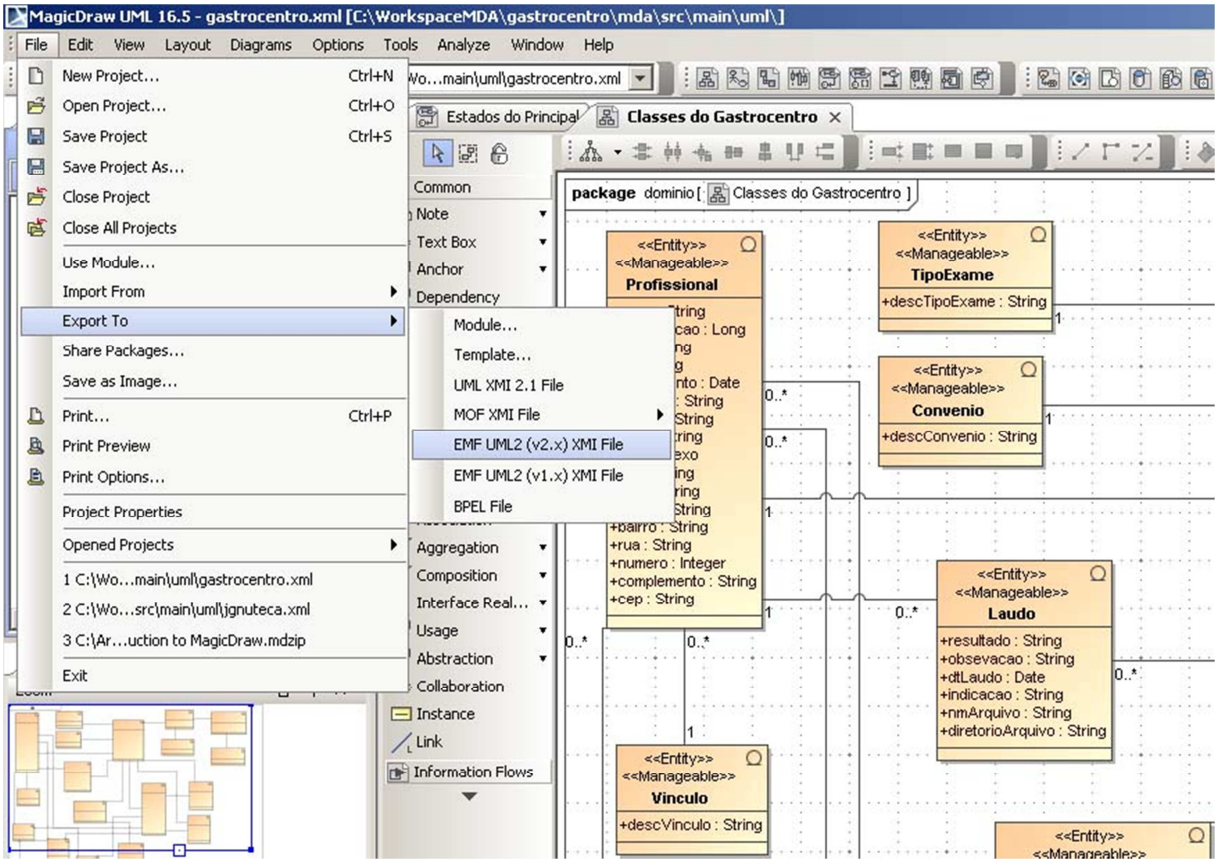


Figura 15 - MagicDraw - Exportar diagramas UML

No processo de compilação, AndroMDA gera classes de persistência com arquivos de mapeamento, *value objects*, classes controller, interfaces, páginas *Java Server Faces* (JSF) entre outros.

Os arquivos fontes podem ser visualizados e editados pelo eclipse, conforme mostra a Figura 16. Para que isto seja possível, deve-se gerar os arquivos de configuração do projeto eclipse executando o comando `mvn –Peclise` no *prompt* de comando. Para importar o projeto todo para dentro do Eclipse de modo que ele entenda todos os fontes e pacotes basta criar o projeto dentro do Eclipse apontando para a pasta do projeto.

AndroMDA também gera o arquivo WAR, pronto para ser executado em um servidor de aplicações.

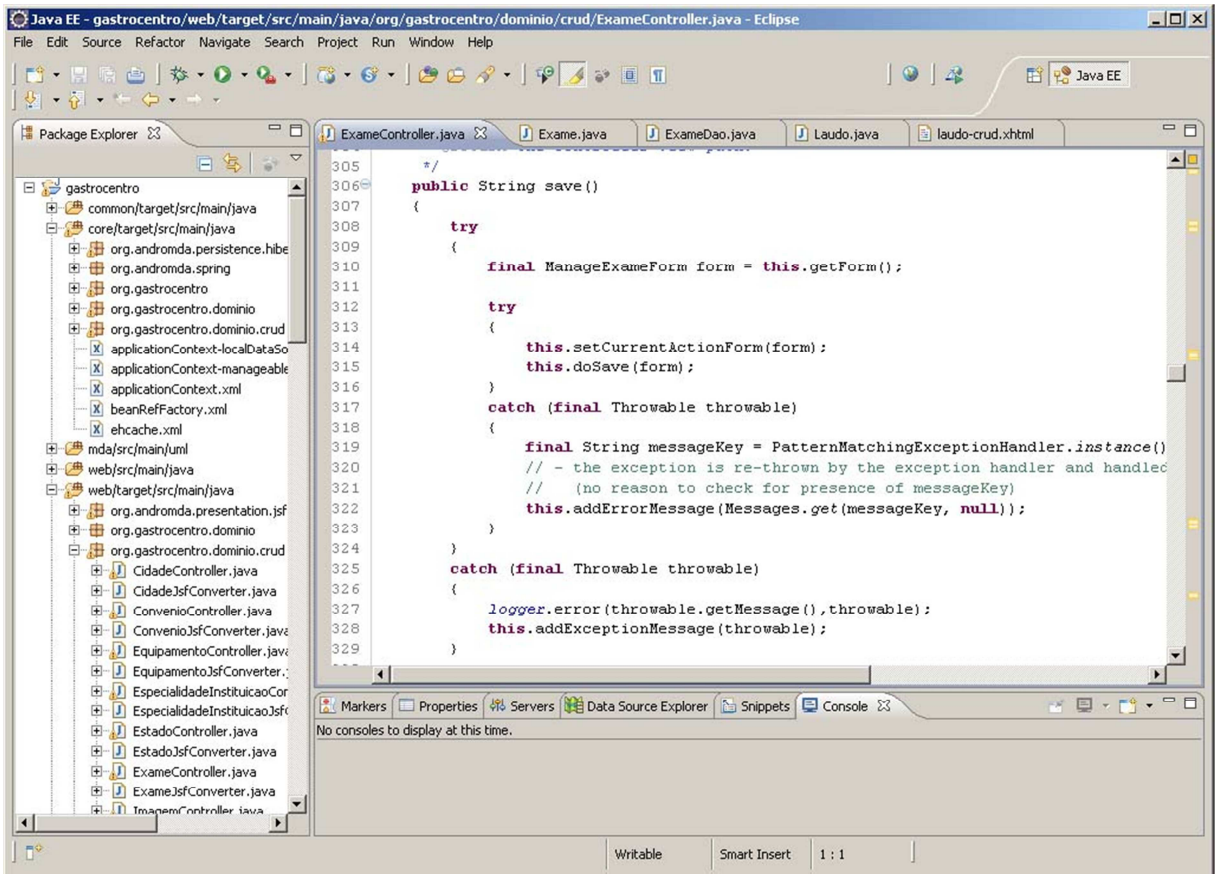


Figura 16 - Eclipse - Projeto importado

Com isso, pode-se considerar que os testes dos principais elementos do framework MDA foram exemplificados. A seguir serão apresentados os resultados obtidos e as considerações finais do capítulo.

5 RESULTADOS E DISCUSSÃO

O processo de desenvolvimeto dos digramas UML é feito normalmente como em qualquer outro projeto de software. Com a diferença de que são aplicados os estereótipos nas classes, para informar ao MDA o tipo de classe e o que ela faz.

O desenvolvimento concentra-se apenas na criação dos diagramas UML, sem se preocupar com os detalhes das plataformas. Com isso, leva-se menos tempo no desenvolvimento do sistema, melhorando a produtovidade.

A ferramenta de geração e compilação não é complicada de configurar e o seu processamento no ato da geração e compilação é rápido. Através de alterações no PIM do sistema é possível efetuar a geração de novos PSMs e código correspondente rapidamente. Correções e adaptações tornam-se tarefas mais simples.

Dependendo dos requisitos do sistema. Será provável que exista a necessidade de alguns ajustes, que deverão ser feitos manualmente no codigo fonte, mas em quantidades muito menores do que no método tradicional de desenvolvimento.

A ferramenta não tem uma interface intuitiva. Funciona apenas por comandos no *prompt* de comando. Em consequência disso, existe uma certa dificuldade na utilização de todos os recursos que o MDA oferece. MDA ainda necessita melhorar a usabilidade das ferramentas.

Outro problema em relação à ferramenta é que ela não tem uma boa documentação, nem comunidade de suporte. O que torna um pouco difícil sua utilização caso queira explorar todos os recursos do MDA ou encontre algum tipo de problema no processo de geração.

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi possível observar vários problemas encontrados no desenvolvimento de software atual bem como as dificuldades de se produzir software com qualidade e rapidez.

A proposta da MDA é oferecer uma forma de melhorar a produtividade. Para isto a arquitetura estabelece seu foco na modelagem, separando os aspectos particulares das tecnologias que serão de fato usadas em sua implementação. Com o foco na modelagem evita-se que os diagramas fiquem defasados. As alterações serão feitas diretamente nos diagramas UML e replicadas ao código fonte.

O principal benefício do MDA é o ganho de qualidade e produtividade. A automatização do processo de transformação destes modelos em outros poderá trazer grandes vantagens.

REFERÊNCIAS

APACHE Maven. **Apache Maven**, 2002. Disponivel em: <http://maven.apache.org/index.html>. Acesso em: 08 fev. 2012.

GETTING Started AndroMDA. **AndroMDA**, 2003. Disponivel em: <http://www.andromda.org/docs/andromda-documentation/getting-started-java/index.html>. Acesso em: 08 fev. 2012.

KARDOS, M.; DROZDOVÁ, M. Analytical Method of CIM to PIM Transformation in Model Driven Architecture, jan. 2010.

KLEPPE, A.; WARMER, J.; BAST, W. **MDA Explained:** The Model Driven Architecture. 1. ed. [S.l.]: Pearson Education, 2003.

MACHADO, R. B. et al. Protótipo de um Sistema Computacional para o Gerenciamento de Dados e Exames de Videocolonoscopia. **Revista Brasileira de Coloproctologia**, Campinas - SP, 2011.

MELLOR, S. et al. **MDA Distilled:** Principles of Model-Driven Architecture. [S.l.]: Pearson Education, 2004.

MILLER, J.; MUKERJI, J. Document -- ormsc/01-07-01 - Model Driven Architecture. **Object Management Group**, 2001. Disponivel em: <http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01>. Acesso em: 05 jan. 2012.

NO MAGIC INC. Introducing MagicDraw. **MagicDraw**, 2000. Disponivel em: <https://www.magicdraw.com/what_is>. Acesso em: 26 mar. 2012.

OMG Model Driven Architecture. **Object Management Group**, 2011. Disponivel em: <http://www.omg.org/mda/>. Acesso em: 06 set. 2011.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. [S.l.]: McGraw-Hill, 2006.

SIEGEL, J. Developing in OMG's Model-Driven Architecture. **Object Management Group White Paper**, Novembro 2001.

SILVA, J. B.; BARRETO, L. P. Separação e Validação de Regras de Negócio MDA, Salvador- BA, Brasil, 2008.

SOLEY, R. Model Driven Architecture. **Object Management Group White Paper**, 27 nov. 2000.

SOMMERVILLE, I. **Software Engineering**. 8. ed. [S.l.]: Pearson Education, 2007.

SOUZA, T. S. D. Model Driven Architecture – Conceitos Fundamentais. **Linha de Código**, 2008. Disponivel em: <http://www.linhadecodigo.com.br/artigo/1953/model-driven-architecture---conceitos-fundamentais.aspx>. Acesso em: 15 out. 2011.

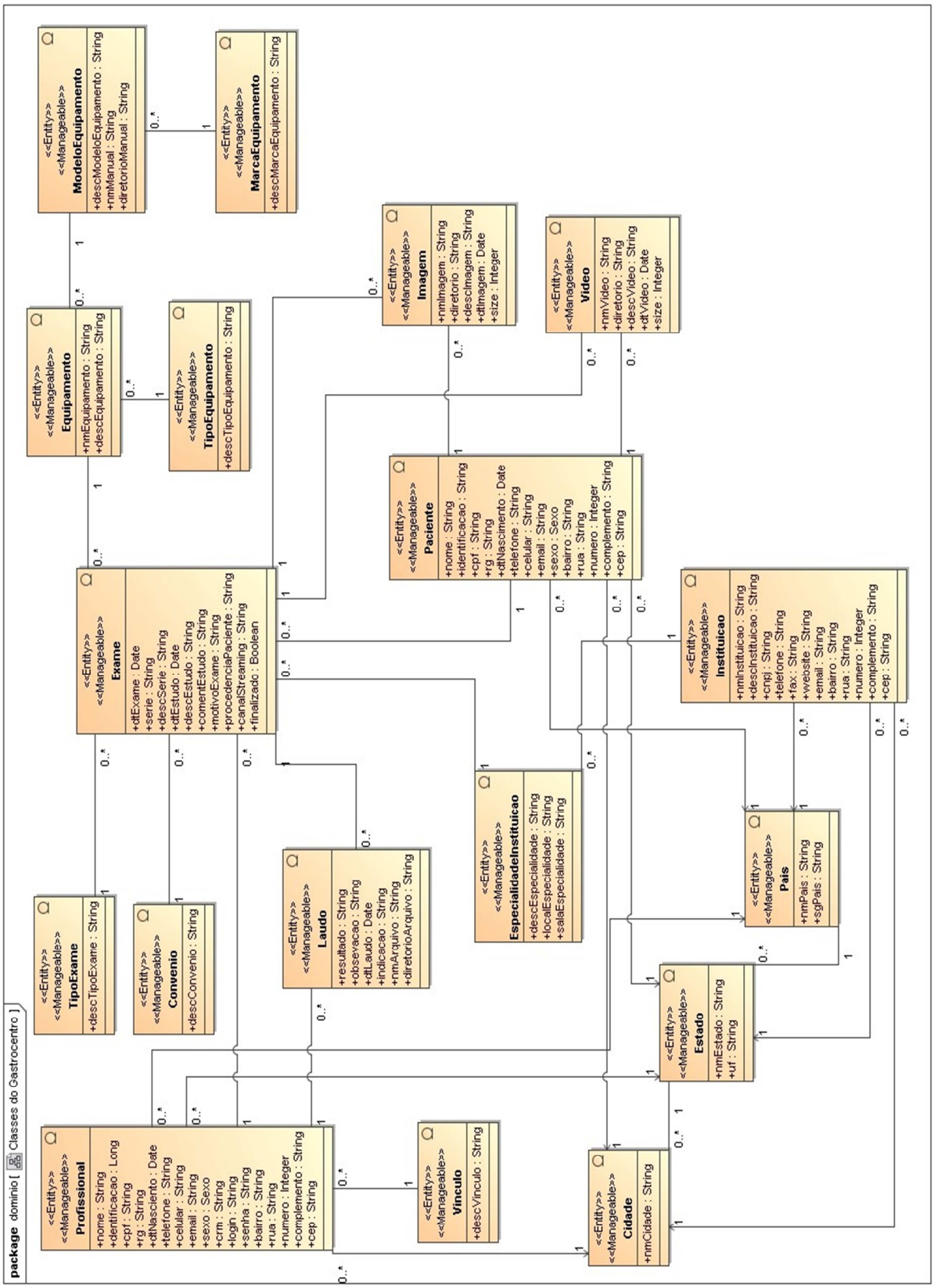
VERNER, A. C.; PUIA, H. D. S. Melhoramentos no desenvolvimento de software com a utilização do MDA, Florianópolis, 11 jun. 2004.

WAGELAAR, D. Platform Ontologies for the Model Driven Architecture. **Brussels University Press**, 04 jul. 2008.

WHAT is AndroMDA? **AndroMDA**, 2003. Disponivel em: <http://www.andromda.org/docs/whatisit.html>. Acesso em: 08 fev. 2012.

ANEXO

ANEXO A – Diagrama de Classe do exemplo utilizado no projeto



Fonte: MACHADO, LEE, *et al.* (2011).