Universidade Tecnológica Federal do Paraná - UTFPR
Campus Ponta Grossa - Paraná - Brasil
ISSN 1808-0448 / v. 06, n. 04: p. 153-172, 2010
D.O.I.: 10.3895/S1808-04482010000400008

Revista Gestão Industrial

GERAÇÃO DO MODELO PSM EM UMA FERRAMENTA DE CÓDIGO ABERTO PARA UM SISTEMA *HELP DESK*

PSM MODEL GENERATION USING OPEN SOURCE TOOL FOR *HELP DESK* SYSTEM

Alison Roger Hajo Weber[1]; Cesar Augusto Januario[2]; Simone Nasser Matos[3]
[1]Universidade Tecnológica Federal do Paraná – UTFPR – Ponta Grossa – Brasil
Universidade Estadual de Ponta Grossa - UEPG – Ponta Grossa – Brasil
alisonrogerweber@gmail.com
[2]Universidade Tecnológica Federal do Paraná – UTFPR – Ponta Grossa – Brasil
januario.cesar@gmail.com
[3]Universidade Tecnológica Federal do Paraná – UTFPR – Ponta Grossa – Brasil
snasser@utfpr.edu.br

Resumo

As empresas de desenvolvimento de software possuem dificuldade em criar projetos para plataforma diferentes. Desta forma, este trabalho estudou o Modelo Dirigido a Arquitetura (MDA) e aplicou-o na elaboração de um sistema Help Desk.Os outros modelos que compõem a arquitetura MDA foram elaborados utilizando ferramentas gratuitas tais como: Acceleo e Eclipse, as quais facilitaram a geração do sistema Help Desk. O uso destas ferramentas forneceu uma maior produtividade em termos de modelagem, geração de código, facilidade de manutenção, agregando interoperabilidade, portabilidade e reusabilidade por meio da separação dos modelos gerados. Além disso, permite que o modelo abstrato possa ser gerado para diferentes plataformas de desenvolvimento.

Palavras-chave: modelo dirigido a arquitetura (MDA); *help desk*; modelos; PHP.

1. Introdução

As empresas de desenvolvimento de software buscam por metodologias de construção mais práticas, simples e otimizadas, com o objetivo de caracterizar o software como um produto que possa ser produzido em escala. Mas, os riscos de tempo de desenvolvimento, requisitos e plataformas variadas dificultam esse processo.

As arquiteturas têm desempenhado um papel fundamental para se obter sucesso nos projetos de software. Por meio delas, consegue-se uma melhor estruturação na produção de sistemas mais ágeis e robustos, possibilitando uma engenharia detalhada e consistente no gerenciamento do projeto.

Dessa forma, a OMG criou uma arquitetura orientada a modelos que possuem em sua essência a construção de sistema em que sua base está na criação do modelo abstrato de sistema, no qual as especificações podem ser implementadas em várias plataformas orientadas a objetos (OMG, 2010).

A perspectiva que esta arquitetura proporciona aos desenvolvedores é que praticamente todo o código fonte é gerado por meio de transformações do modelo PIM (*Platform Independent Model*) para o modelo PSM (*Platform Specific Model*) (OMG, 2003, p 13).

Este artigo relata a geração do modelo PSM no desenvolvimento de um Sistema *Help Desk*, elaborado e gerado por meio da ferramenta Acceleo (*ACCELEO*, 2010).

Por meio da ferramenta foi possível aumentar a produtividade do desenvolvimento do sistema *Help Desk*, pois gerou uma quantidade de códigos em que as alterações foram mínimas.

2. Arquitetura MDA

As primeiras propostas de desenvolvimento de software usando a arquitetura MDA (*Model Driven Architecture*) surgiram no final da década de 1990, início de 2000 (MDA GUIDE, 2010).

Esta abordagem de projeto de software oferece uma definição de aplicação baseada em modelos, permitindo uma flexibilidade em longo prazo. Desta forma, mostra-se como o melhor caminho para o desenvolvimento de sistemas baseados em linguagens orientadas a objetos, visto que a implementação pode se integrar a diversas plataformas, o que proporciona ao modelo mais flexibilidade (OMG, 2003, p. 12).

Esta abordagem de construção de projetos de software orientada a modelos e abstrações de domínios distintos busca a melhor análise e consequentemente propicia uma produtividade maior, maximizando uma integração mais compatível entre os sistemas.

Portanto, a proposta desta arquitetura prevê a idéia de separar a especificação das operações do sistema, alterando desde detalhes do canal de comunicação do sistema até as capacidades de sua plataforma.

A MDA é um padrão que pode ser considerado um framework aberto criado e mantido pela OMG (*Object Management Group*) (VERNER, PUIA, 2004, p. 17). A OMG tem como principal foco o desenvolvimento da excelência tecnológica, comercialmente viável e independente das especificações proprietárias para indústria de software.

Suas principais diretrizes são fundamentadas num conjunto de boas práticas e designações que consistem nas especificações de modelos, centrados em arquiteturas de software (SILVA, 2008, p. 1).

A MDA propõe que os artefatos gerados possam ser usados na geração de programas, *scripts* de banco, documentação de usuário e de software, ou seja, qualquer elemento que faça parte do processo de desenvolvimento de um sistema.

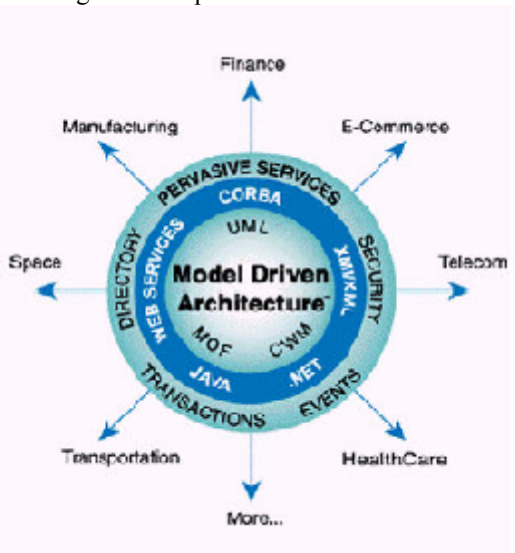
A principal característica do MDA pela OMG é propiciar produtividade com qualidade aos softwares, agregando interoperabilidade, portabilidade e reusabilidade por meio da separação do modelo gerado da plataforma a ser usada, rastreando qualquer um dos requisitos. Fornece também a garantia necessária ao modelo através da documentação gerada, dando um suporte mais apurado para manutenções futuras dos artefatos produzidos durante o desenvolvimento, na fase de implementação do software.

Mas, o fundamento mais importante deste tipo de arquitetura é a criação de um modelo de sistema, dentro de um domínio específico, no qual sua independência permite ser uma referência para qualquer tipo de plataforma ou linguagem, preservando as características do modelo inicial, adequando-se o modelo a linguagem preterida.

A perspectiva que esta arquitetura proporciona aos desenvolvedores é que praticamente todo o código fonte é gerado por meio de transformações do modelo PIM (*Platform Independent Model*) para o modelo PSM (*Platform Specific Model*) (OMG, 2003, p 13).

O centro da arquitetura, ilustrado na figura 1, é baseado nos padrões de modelagem da OMG: UML (*Unified Modeling Language*), MOF (*Meta Object Facility*) e CWM (*Common Warehouse MetaModel*). Além disso, tem-se duas camadas que completam esta arquitetura. A primeira representa a plataforma que é o alvo deste framework e a segunda relaciona-se aos serviços que devem existir independentes da plataforma adotada.

Figura 1- Arquitetura MDA



Fonte: OMG (2003)

Um modelo de sistema descreve e especifica a abstração de um problema através de textos, linguagens de modelagem ou linguagem natural. Eles também são visões simplificadas e contextualizadas do mundo real, como por exemplo, uma planta de uma casa ou de uma instalação hidráulica e elétrica. Os diagramas e fluxogramas são modelos de um sistema em que são utilizados em grande escala pela engenharia de software.

Na arquitetura MDA existem três modelos na qual são criados a concepção, análise e transformação de um sistema. Os modelos são descritos a seguir.

2.1. Modelo CIM

Sua base é o entendimento do sistema, em que a análise possibilita uma visão computacional com uma perspectiva independente, não visualizando detalhes estruturais do sistema, mas sim o domínio onde o modelo será empregado. O objetivo é gerar uma visão unificada para analistas e desenvolvedores, com o foco nos requisitos pertinentes para o sistema atender o domínio (OMG, 2003, p. 15).

2.2. Modelo PIM

O PIM é uma visão de modelo independente de plataforma baseado em uma análise de alto nível, representando somente funcionalidades e comportamentos do sistema. Neste caso, a abstração do sistema atinge um grau de máquina virtual neutra abrangendo todos os componentes e serviços relacionados ao modelo, tais como: comunicação, nomes, programação, preservando sua independência de plataforma, em relação às sintaxes utilizadas para implementação das diferentes linguagens, sendo a modelagem baseada somente nas especificações do sistema (OMG, 2003, p. 16).

2.3. Modelo PSM

Com a possibilidade de mapear o PIM se pode gerar o modelo PSM. Este compreenderá a transformação do modelo abstrato em codificado, se ajustando aos detalhes que definem como o sistema se desenvolverá na plataforma especificada.

O modelo especifica também os requisitos pertinentes à conexão e o uso dos elementos da plataforma junto ao seu aplicativo, permitindo a visualização da tecnologia de implementação utilizada (OMG, 2003, p. 17).

Desse modo, o modelo representa a unificação do modelo abstrato com a plataforma que foi escolhida.

2.3.1. Ferramentas de transformação

A ferramenta utilizada neste artigo foi a *Acceleo* que funciona como um *plugin* da IDE Eclipse (ECLIPSE, 2010; ACCELEO, 2010). Esta ferramenta, *open source* de modelagem, atualmente é uma das mais utilizadas para a geração de modelos MDA. As vantagens principais oferecidas por esta ferramenta são: alto poder de customização, interoperabilidade, fácil aprendizado, direcionamento as principais tecnologias existentes no mercado, entre outras.

3. Geração do Modelo PSM usando a ferramenta Acceleo

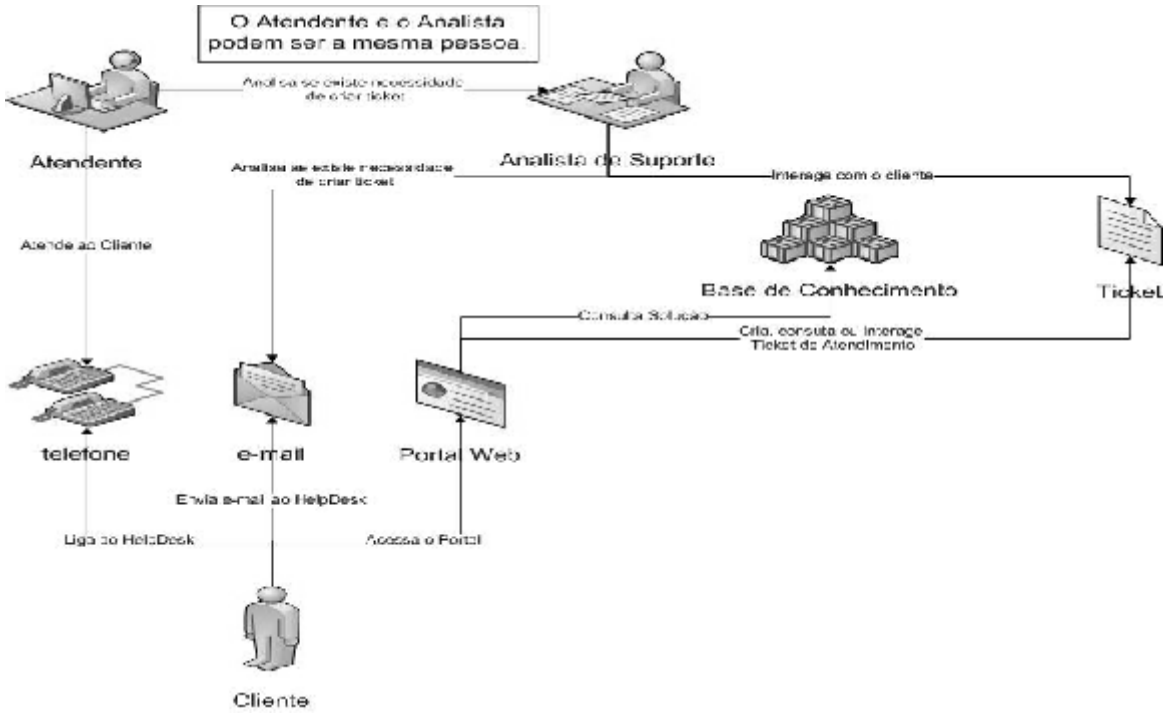
Para exemplificação da arquitetura MDA, este artigo traz o desenvolvimento de um software para controles de centros de suporte conhecidos como *Help Desk*. Este atenderá às principais características desses centros, no qual será possível controlar chamadas, soluções, clientes, entre outros. Para se chegar à construção do modelo PSM deste software deve-se antes passar pelos modelos CIM e PIM, descritos brevemente a seguir.

3.1. Modelos CIM e PIM

No modelo CIM o foco é a abstração do domínio, nesse caso, um Sistema *Help Desk* cujo objetivo é oferecer às empresas um software que controle o serviço de apoio e suporte para as pessoas, tecnologias e processos computacionais (COHEN, 2008).

Os processos de atendimento ao cliente se dão através de três canais de comunicação: via telefone (usuário e atendente), via portal de atendimento (usuário x aplicação web), e correio eletrônico (usuário x e-mail). Os tipos de canais de comunicação estão ilustrados na Figura 2.

Figura 2 - Fluxo de trabalho geral do *Help Desk* proposto



Fonte: Autoria Própria

As pessoas envolvidas no fluxo de trabalho do sistema de *Help Desk* são: cliente, atendente e analista de suporte. O cliente pode contatar o suporte por telefone ou e-mail, em que um funcionário o atenderá, com um perfil de solucionador, ouvindo o problema do usuário e tentando resolvê-lo.

Caso o problema não possa ser resolvido em um primeiro contato é efetuada uma análise para verificar se existe a necessidade de criação de um *ticket* de atendimento, que é um documento virtual onde são registradas todas as informações do chamado. O *ticket* tem um cabeçalho contendo os dados do cliente e os do problema. O corpo do *ticket* é composto por uma descrição que corresponde às interações entre o atendente e o usuário, tentando solucionar o problema.

Caso o cliente queira consultar ou interagir com seu *ticket*, ele poderá acessar o portal na web. Ainda pelo portal o cliente poderá criar um *ticket* de atendimento que seguirá o mesmo padrão dos criados pelos atendentes. A única diferença no processo é a análise, que será feita depois da criação do registro do *ticket*, ou seja, o analista de suporte irá verificar se esse registro criado pelo cliente é mesmo necessário.

Quando um *ticket* é finalizado, ou seja, o problema foi solucionado, o analista de suporte poderá adicionar a informação deste *ticket* em uma base de conhecimento com o objetivo de esclarecer dúvidas aos clientes. Essa base de conhecimento poderá ser consultada tanto pelos atendentes como pelos clientes através do portal.

Como citado anteriormente, existem três tipos básicos de atendimento nesse domínio, através de telefone, e-mail ou do portal, que basicamente irão resultar no mesmo atendimento.

Após o estudo do domínio, iniciou-se a construção do modelo PIM, que consiste na identificação de todos os atributos, métodos, generalizações e composições necessárias para que o modelo abstrato possa ser implementado.

Dessa forma, o modelo pode deixar visível as especificações que serão utilizadas pelo sistema. O modelo PIM criado para o sistema *Help Desk* está ilustrado na Figura 3.

As classes Cliente e Atendente representam os dois usuários de um sistema *Help Desk*. Estes possuem atributos e métodos comuns, desta forma criou-se uma generalização, denominada de Física. A classe Fisica instancia um objeto da classe Login, que fará o controle e validação dos usuários que acessam o sistema.

Como o *Help Desk* pode atender clientes físicos e empresas, contextualizou-se estes elementos do mundo real através das classes Fisica e Juridica e como estas possuem comportamentos e características comuns criou-se a classe Pessoa. A classe Pessoa também se relaciona com classe Cep identificando os endereços de qualquer tipo de cliente.

A classe Classificacao contém as prioridades e os tipos de serviço que o *Help Desk* oferece aos clientes, diferenciando o atendimento. Por exemplo, um hospital que possui um aplicativo que não pode parar porque é usado no tratamento intensivo de vários pacientes, estes tipos de chamado tem prioridade em relação aos outros.

Figura 3 - Modelo PIM *Help Desk*

Fonte: Autoria Própria

As classes Cargo e Departamento têm como tipo de relacionamento uma composição, além de estarem ligadas com classe Cliente no modelo. Elas são responsáveis por identificar e registrar as origens dos chamados.

No fluxo atendimento via portal o atendente pode criar um *ticket* de atendimento, paralelamente, o cliente também pode fazer esta ação via web, desta forma, estes requisitos foram modelados na classe Ticket, que é o centro do modelo ou núcleo do sistema. Os métodos e atributos de Ticket, bem como seus relacionamentos são responsáveis pela especificação e registro dos chamados através das relações com as classes SubCategoria e Categoria, as quais classificam as descrições dos equipamentos que estão com defeito. Por exemplo, um incidente com ADSL é uma categoria, e o problema identificado tal como: modem queimado é uma subcategoria.

A classe *Ticket* contém ainda os registros dos incidentes, deste modo, a cada interação com o *ticket*, tanto registrado pelo cliente ou pelo atendente, poderá haver uma consulta posteriormente pelos atendentes de nível superior.

A classe TempoDeResolução resolve o problema do controle de tempo que os incidentes levam para serem resolvidos com intuito de melhorar o serviço de suporte prestado aos clientes. Esta classe se relaciona com a classe SubCategoria.

A classe Acompanhamento foi modelada para gerenciar o andamento e os últimos registros dos tickets abertos e finalizados, possuindo uma relacionamento com as classes Ticket, Fisica e Email.

A classe Email será usada para enviar um e-mail para o cliente avisando que o ticket foi finalizado e esclarecendo como o problema foi resolvido. A classe Solucao foi modelada para atender ao requisito de soluções prontas e alimentar a base de conhecimento do *Help Desk*. Além disso, registra as soluções encontradas nas interações feitas na classe Ticket.

Neste artigo, não será detalhado como foi a elaboração tanto do modelo CIM quanto do PIM, pois o foco é o modelo PSM, descrito a seguir.

3.2. Modelo PSM

Gerar o modelo PSM é a última fase da arquitetura MDA, pois é nesse momento que se deve validar o modelo PIM, ilustrado na Figura 3.

O projeto do PSM é formado pela pasta do modelo, o qual é composto por dois arquivos. O primeiro é o ‘modelHelpDesk.uml’ que contém o modelo PSM e será utilizado para a geração de código-fonte.

O outro arquivo é o ‘modelHelpDesk.umlclass’ que é o PSM, representado pelo diagrama de classe (estes arquivos estão ilustrados posteriormente na Figura 3).

Para criar o diagrama de classe (arquivo.umlclass) é necessário que o arquivo 'modelHelpDesk.uml'(arquivo.uml) já esteja definido.

Qualquer alteração que ocorra em um arquivo, irá também afetar o outro, mas ressalta-se que o arquivo que realmente gera o esqueleto da aplicação é o 'modelHelpDesk.uml'.

Nesta etapa, iniciou-se a operação de mapeamento e marcação do modelo PIM (mostrado na Figura 3). Esse mapeamento é o processo de importação dos estereótipos de entidades, dados, interface e diversos componentes fundamentais para a realização da geração dos códigos fontes pela ferramenta, garantindo a transformação do modelo em sua totalidade ou parcialmente, dependendo de sua plataforma. Neste artigo os códigos gerados para a plataforma PHP.

A importação de componentes que será viabilizada pela própria Acceleo, utilizará "pathmap", mostrado na Figura 4. São em sua existência as referências para os arquivos .uml, tal como "pathmap://ACCELEO_PROFILES/Acceleo.UI.profile.uml", neste caso, representando os tipos de dados de interface gráfica pertinentes para a transformação.

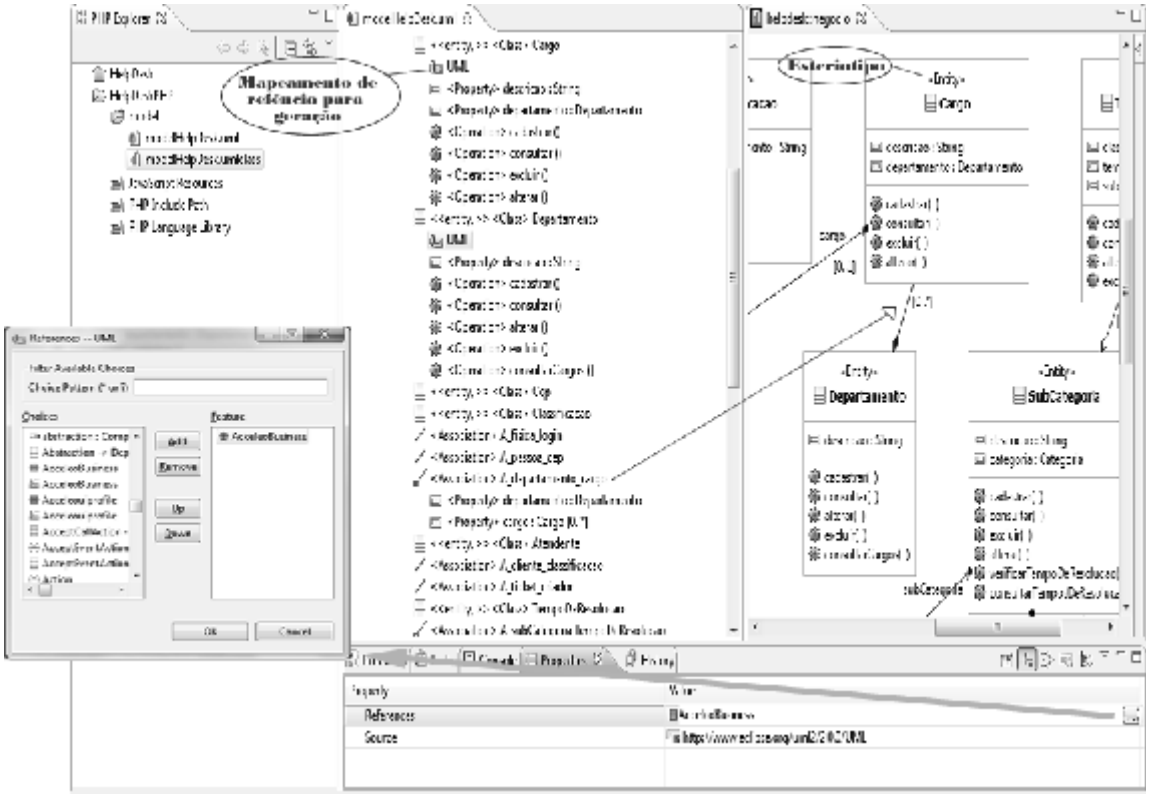
Figura 4 - Mapeamento do modelo

Fonte: Autoria Própria

Caso não se tenha a existência do dado requisitado pelo modelo abstrato, ou a importação não seja possível, o desenvolvedor poderá criar o dado dentro do modelo, como ilustrado na Figura 4 na criação de DATE e TIME na raiz do pacote <Model> *Help Desk*. Esses dados funcionarão sem nenhum problema após a transformação em um exemplo de criação.

No diagrama 'modelhelpdesk.umlclass', visualizado ao lado direito da Figura 5, é possível classificar as classes com o seus estereótipos pressionando o botão direito do mouse e selecionando o estereótipo desejado. Mas para realizar a geração do código é necessário também criar uma referência para geração na UML. Para isso, abre-se o arquivo 'modelHelpDesk.uml' e insere-se um componente chamado *EAnnotation.*

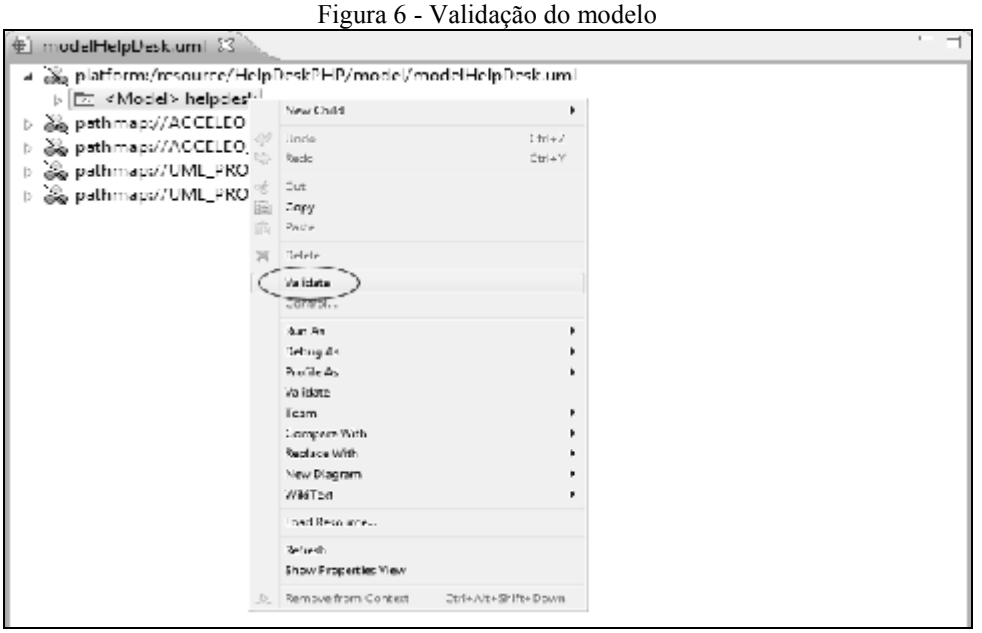
Figura 5 - Mapeamento e marcação



Fonte: Autoria Própria

Por meio desta propriedade, coloca-se uma referência para o pacote ‘Acceleo Business’’ e a propriedade ‘source’ para http://www.eclipse.org/uml2/2.0.0/UML’, conforme mostra a Figura 5, e na sequência para a finalização da marcação para as classes, é fundamental executar a validação deste modelo mostrado na Figura 6.

Figura 6 - Validação do modelo



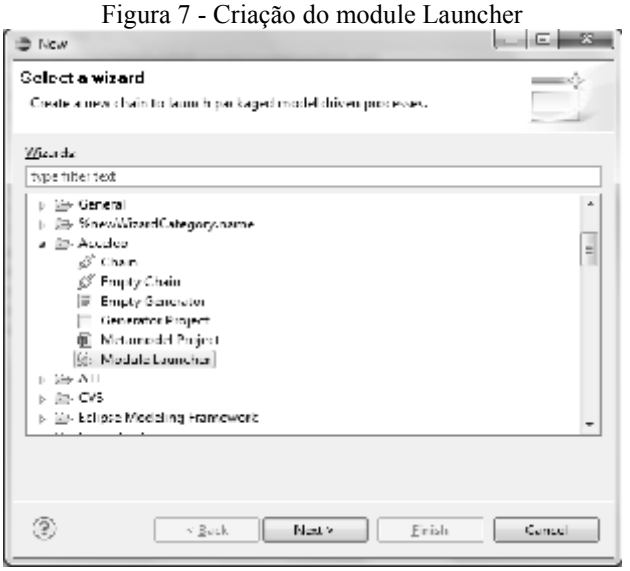
Fonte: Autoria Própria

Para isto, sobre qualquer item do modelo e escolhe-se a opção '*Validate*'. Desta forma, a ferramenta realiza a validação para os itens que estão selecionados.

3.2.1. Transformação e geração de códigos-fonte

A próxima operação é a geração do código. Nesta etapa, é necessário criar um Module *Launcher* dentro do projeto, visualizado na Figura 7.

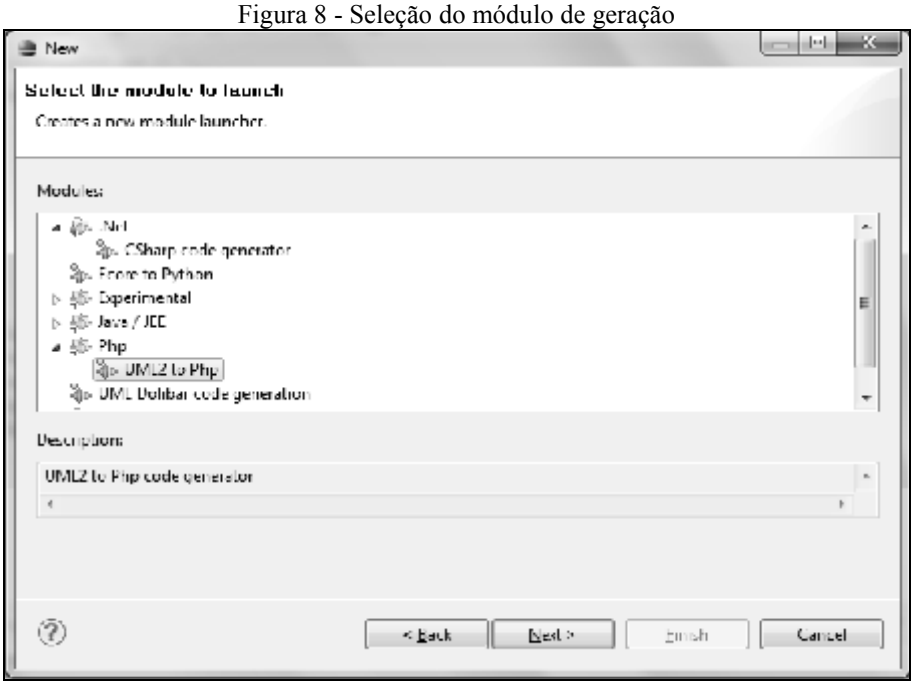
Figura 7 - Criação do module Launcher



Fonte: Autoria Própria

A próxima etapa é selecionar o módulo da ferramenta *Acceleo*, que irá ser utilizado para gerar o código fonte, conforme mostrado na Figura 8.

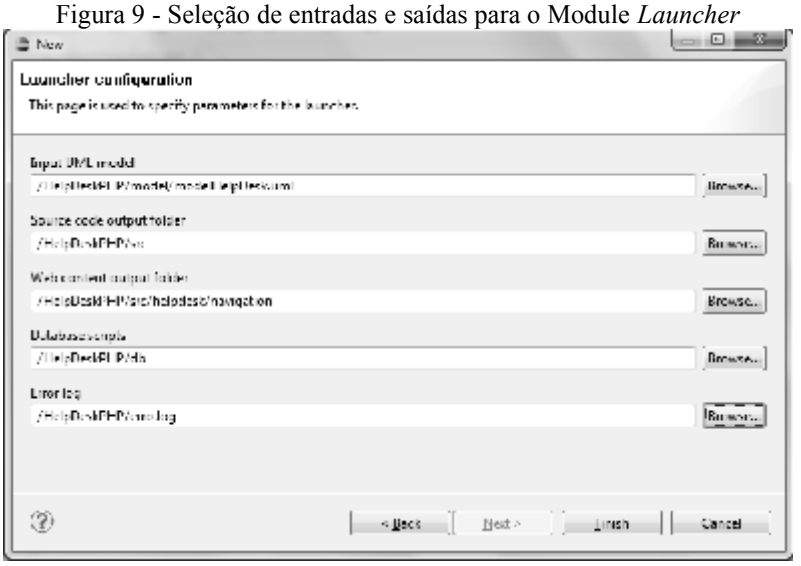
Figura 8 - Seleção do módulo de geração



Fonte: Autoria Própria

Logo após, a última etapa da criação deste arquivo é a seleção de entradas e saídas do modelo. Para o módulo PHP são exigidos os campos ilustrados na Figura 9.

Figura 9 - Seleção de entradas e saídas para o Module *Launcher*



Fonte: Autoria Própria

Essas opções ilustradas anteriormente representam:

- *Input UML model*: Selecionar o modelo de entrada, o modelo PIM.
- *Source Code output folder*: Pasta na qual será gerado o código fonte da aplicação.
- *Web content output folder*: Pasta na qual serão inseridos os controles de navegação da aplicação (controles, visões, e outros).
- *Database scripts*: Pasta na qual serão gerados os *scripts* de banco de dados.
- *Error log*: Arquivo criado que conterá os erros encontrados durante a geração.

Ao terminar o processo, o arquivo *transformação.chain* foi criado e a transformação do modelo poderá ser iniciada por meio da geração de códigos.

3.2.1.1. Geração de códigos

A partir do modelo gerado na UML 2.0, a ferramenta *Acceleo*, utilizando seu gerador para a plataforma PHP, resultou em uma aplicação utilizando PEAR (PEAR, 2010) e SMARTY (*SMARTY,* 2008).

O PEAR (PHP *Extension and Aplication Repository*) é uma plataforma e um sistema de distribuição para a codificação de componentes em PHP. As vantagens na utilização do projeto PEAR são: os sistemas de distribuição de código e a gerência de pacotes aliados a um padrão para a escrita de código em PHP. Os pacotes PEAR existem para executar muitas funções, como de autenticação, controle de erros, *caching*, acesso a base de dados, criptografia, configuração, HTML,

Web Services e XML (PEAR, 2010).

O *SMARTY* é conhecido como uma classe de *templates*, na qual ele busca separar a interface da lógica de programação e possui o objetivo de aumentar a qualidade de qualquer aplicação em PHP. Foram constatadas as vantagens de um melhor desempenho de execução baixando o *overhead* de *template*, compilando apenas uma vez. Contém também um modo de *cache* embutido, que cria funções próprias, ou seja, é extensível (SMARTY, 2008).

3.2.1.2. Descrição dos códigos-fonte e da transformação

Durante a geração de código pela ferramenta foram criadas várias pastas, entre elas a: 'db' - que contém todos os *scripts* de banco de dados no arquivo 'createTable.sql' -, a 'model' - contém o modelo UML -, e a 'src' - possuiu todos os códigos fontes do projeto. A 'src' está dividida em várias subpastas que possuem as seguintes funcionalidades:

- 'common': Ficam os arquivos que possuem métodos, importações, configurações, variáveis, e outros, que sejam comuns a todo projeto.
- 'dao': Contém as classes que fazem a comunicação com o banco de dados.
- 'entity': Localizam-se as entidades do modelo.
- 'navigation': Possuem os arquivos de navegação do software, por exemplo, as visões e as classes de controles que fazem a comunicação de todo o software.
- 'test': Encontram-se os arquivos de testes para o modelo. Esta não foi explorada neste artigo.

Além das pastas, a ferramenta gerou o arquivo 'erros.txt' que possui os erros que aconteceram na geração do código. Esses erros podem compreender entre algum campo que não esteja com sua tipagem correta, associações incorretas, estereótipos inválidos ou qualquer outro erro que possa acontecer no modelo.

A transformação do módulo do *Acceleo* na geração para PHP5 fica estruturado com as camadas de DAO (*Data Access Object*), *Business Objects* e Apresentação.

A partir do modelo, o gerador utiliza as classes para gerar a aplicação. Essa identificação procura por dois tipos de estereótipos: os de Entidade, que permitem a geração dos negócios na parte da aplicação (*Entity*), e os de Tela, que permitem a geração da parte de apresentação da aplicação (*Screen*).

Os arquivos gerados a partir dos estereótipos de Entidade foram:

- Parte dos *scripts* de criação do banco de dados.
- Classes DAO (na pasta 'dao').
- Classes de Entidade (na pasta 'entity') .

- *EntityFactory* classes que permitem criar instâncias das entidades
- SMARTY *template* que permite exibir o objeto de negocio (na pasta 'navegacao/view').

Os estereótipos de Tela criados foram:

- Parte do arquivo de índice.
- Controladores de classes (na pasta 'navegacao/controler').
- SMARTY *template* (na pasta 'navegacao/view').

Outros arquivos gerados pelo módulo de geração para PHP do *Acceleo* foram:

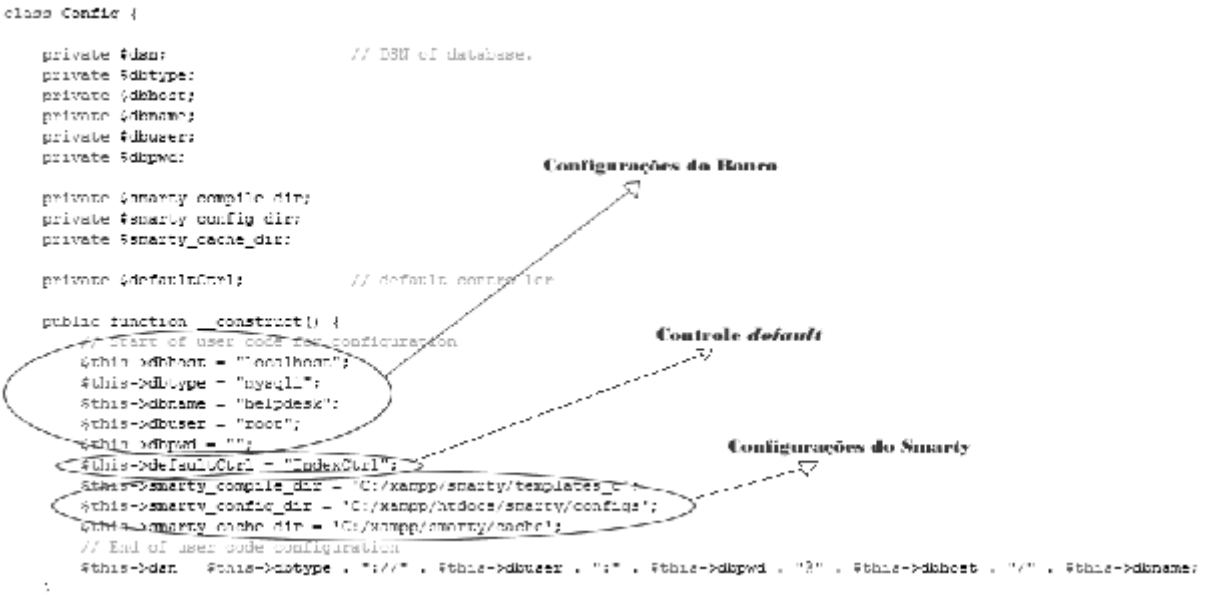
- Arquivo de importação de todas as dependências necessárias ('common/common.php5')
- Configuração de classes ('common/config.php5')
- Arquivo incluindo instruções para criação de sequências de banco de dados ('common/dbsequ.php5')
- Inteface para classe entidade ('entity/IEntity.php5')
- Entidade de classe abstrata ('entity/Entity.php5')

4. Resultados da transformação

A primeira etapa foi criar o banco de dados, sendo que o *MySQL* (2010) foi o adotado para o sistema *Help Desk*. A ferramenta *Acceleo* gerou parte deste *script* de banco de dados, composta por todas as tabelas e relacionamentos que existiam no modelo. Esse *script* estava correto, a única alteração antes de sua importação para o *MYSQL* foi remover o atributo '*VERSION'*, o qual não foi implementado neste trabalho.

A segunda etapa realizada foi a configuração do projeto através do arquivo 'helpdesk/common/config.php5', ilustrado na Figura 10, no qual foi configurado o banco de dados, *controle default* e as configurações do SMARTY.

Figura 9 - Configuração do projeto



Fonte: Autoria Própria

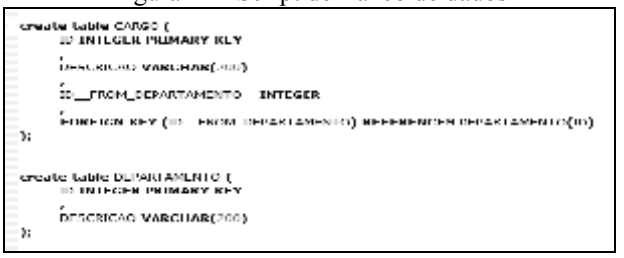
Para exemplificar a geração do código e a implementação do software utiliza-se a Classe Cargo a qual possui um relacionamento com a Classe Departamento, como ilustrado na Figura 3.

As tabelas do sistema que possuem uma chave primária, formada por um atributo inteiro e incremental, são controladas pelo código fonte. Para isso, no banco, devem existir tabelas auxiliares para salvar o número do último registro inserido na tabela que ela corresponde. Essas tabelas auxiliares são criadas com o mesmo nome de sua respectiva classe, adicionando no fim destas as letras '_seq'. Mas para criar as tabelas auxiliares o arquivo 'helpdesk/common/dbsequ.php5' deve ser executado através do navegador.

Tendo que a classe Cargo foi classificada como um tipo de estereótipo de Entidade tem-se as seguintes saídas:

- *Script* de banco de dados: Tabela de cargos, com uma chave estrangeira da tabela de departamentos com a qual possuía uma composição. O *script* gerado está ilustrado na Figura 11.

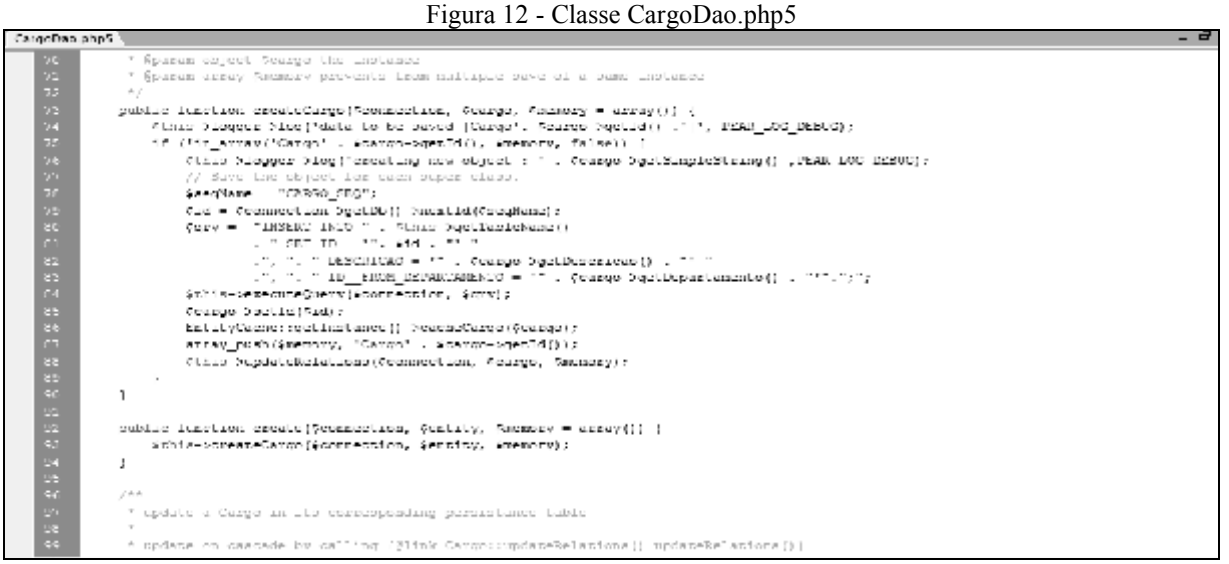
Figura 11 - Script de Banco de dados



Fonte: Autoria Própria

- Classe Dao: Esta classe é responsável pelo acesso a base de dados e está localizada na pasta (‘helpdesk\dao\negocio\pear\CargoDao.php5’. A Figura 12 mostra um trecho de código desta classe responsável pela inserção no banco de dados.

Figura 12 - Classe CargoDao.php5



Fonte: Autoria Própria

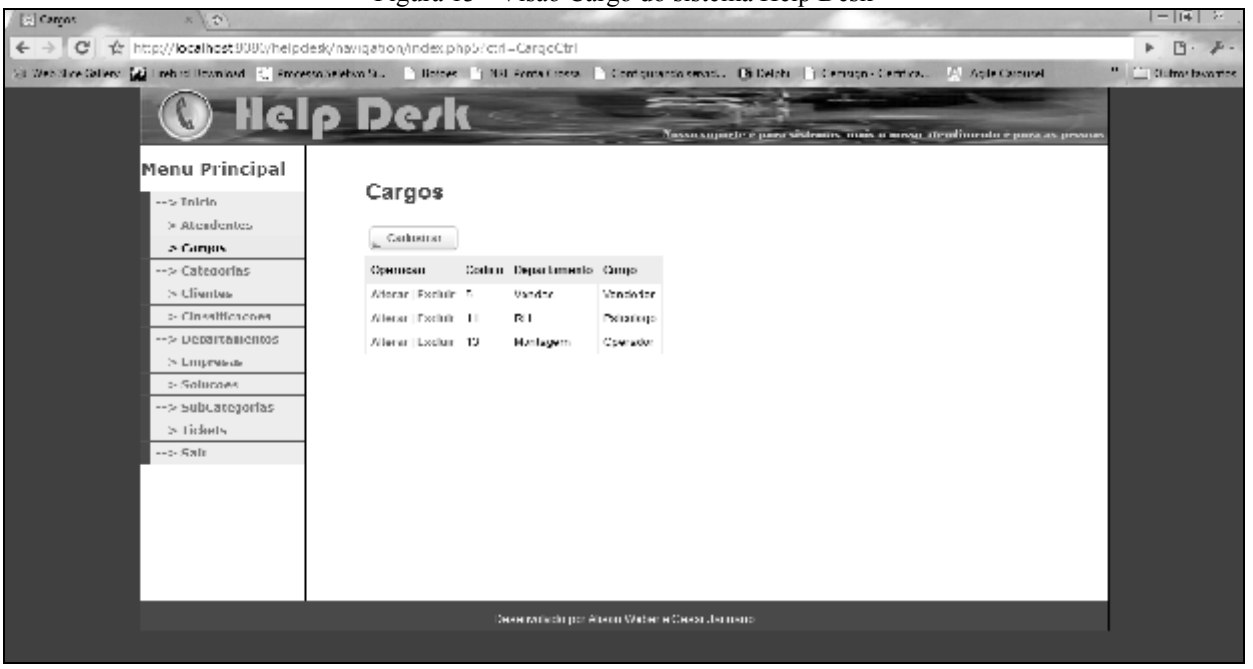
- Classe *Entity*: Esta classe é a Cargo, que possui os atributos, métodos, sets e gets e está localizada na pasta (‘helpdesk\entity\negocio\Cargo.php5’).
- Classe *Factory*: A Classe *Factory* é responsável pela criação do objeto cargo tendo nele os atributos da classe *Entity*.

Além desses arquivos mencionados, há também vários outros que compõe este projeto, como por exemplo, arquivos de captura de erros, as Classes de Interface, Classe *Entity,* Conexão de banco de dados, entre outros.

Neste ponto do projeto a ferramenta conseguiu gerar 100% da parte de negócios da aplicação. Como descrito na seção anterior, a parte de visão deste projeto foi gerada no modelo do *framework SMARTY*. Os arquivos de interface ficam todos situados dentro da pasta ‘navigation’, na qual há um arquivo chamado ‘índex.php5’, que controla as chamadas das páginas, ou seja, a navegação.

A *templat*e está localizada no arquivo ‘main-template.xml’, com uma estrutura HTML, que possui tags para importações de outros arquivos dentro dele. O resultado da transformação pode ser visualizado na Figura 13.

Figura 13 - Visão Cargo do sistema Help Desk



Fonte: Autoria própria

Nesta seção ilustrou-se a geração de código para a classe Cargo desde a parte de *scritp* de banco de dados até a geração da camada visão, ilustrada na Figura 13. O processo realizado com a classe Cargo se repetiu para as demais classes que formavam o sistema, ilustradas na Figura 3.

Constatou-se que o resultado obtido com a utilização da MDA foi muito bom, onde se ganha em produtividade na implementação do software, principalmente na parte de negócios da aplicação. Já a parte de visão e controle foi onde se teve inserção e alteração de código por se tratar de algo específico e que com certeza mudará dependendo da aplicação.

A ferramenta Acceleo também mostrou grande eficiência na geração do modelo PSM de forma incremental, ou seja, construir o software por partes ou incrementando novas funcionalidades ao modelo, garantindo a integridade do que já foi desenvolvido.

A MDA prove que o modelo PIM deverá estar correto antes de ser transformado para o modelo PSM, mas se acontecer do modelo PIM ser alterado depois da geração do código fonte e ocorrer uma nova geração, a ferramenta se porta de maneira muito interessante, ou seja, ela não sobrescreve os arquivos já gerados, mas, salva o arquivo antigo com um nome diferente, acrescentando ao seu fim a extensão ‘.lost’.

5. Conclusão

Este artigo mostrou a construção do modelo PSM utilizando a arquitetura MDA no desenvolvimento de um sistema *Help Desk*.

Para uma produtividade maior no uso da MDA, foi utilizada uma ferramenta própria para este tipo de arquitetura denominada *Acceleo*. Por meio desta ferramenta, pode-se verificar que a quantidade de códigos que deve ser escrito em determinadas situações é mínima, como exemplificado para a classe Cargo, em que para consultar cargos a quantidade de linhas programadas foram três. Além disso, todo o código gerado possuiu um padrão de comentário o qual facilita a manutenção do sistema.

O uso da MDA permite produzir artefatos que promovam a rastreabilidade, a redução considerável de falhas humanas na implementação e a integridade da documentação. Como trabalho futuro pode-se utilizar o modelo PIM e transformá-lo para outra plataforma, além disso pode-se utilizar indicadores para medir a qualidade de código gerado.

Abstract

The software development corporations have difficulty in creating designs for different platforms. Thus, this study investigated the Model Driven Architecture (MDA) and applied it to design a Help Desk system. The system requirements model was created with the workflow technology. The other models that comprise the MDA architecture were developed using free tools such as Acceleo and Eclipse, which facilitated the generation of Help Desk system. The use of these tools provided a higher yield in terms of modeling and code generation, maintainability, increment interoperability, portability and reusability through the separation of the generated models. Besides it allows the model will be generated for different development platforms.

Key-words: Model Driven Architecture (MDA); *Help Desk*; Model; PHP.

Referências

ACCELEO. Disponível em: <http://www.acceleo.org>. Acesso em: 20 mar 2010.

COHEN, R. **Implantação de *Help Desk* e Service Desk**. São Paulo: Novatec Editora Ltda, 2008.

ECLIPSE. Disponível em: <http://www.eclipse.org/downloads/> Acesso em: 24 mar 2010.

MDA GUIDE. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01/>. Acesso em: 25 fev 2010.

MYSQL. Disponível em: http://wb.mysql.com. Acesso em 03 mai 2010.

OMG. Model Driven Architecture. Disponível em: <http://www.omg.org/mda/>. Acesso em: 11 mar 2010.

PEAR. Disponível em: <http://www.pear.php.net/>. Acesso em: 02 mai 2010.

SMARTY. Disponível em: <http://www.smarty.net/>. Acesso em: 01 mai 2010.

SILVA, J. B.; SAMPAIO, M.; PEZIN, J. **Usando Ontologias na construção de Modelos MDA (Model-Driven-Archteture)**. Universidade Salvador (UNIFACS) Salvador, BA, Brasil. 2008.

VERNER, A. C.; PUIA, H. S. **Melhoramento no desenvolvimento de software com a utilização do MDA**. Santa Catarina, 2004. 40 f.

Dados dos autores:

Nome completo: **Alison Roger Hajo Weber**

Filiação institucional: Universidade Tecnológica Federal do Paraná Campus Ponta Grossa

Universidade Estadual de Ponta Grossa

Departamento: COINF

Função ou cargo ocupado: Estudante de Graduação (UTFPR) e Mestrado (UEPG)

Endereço completo para correspondência (bairro, cidade, estado, país e CEP):

Av. Monteiro Lobato, km 4, s/n
Santa Mônica
Ponta Grossa
Brasil
84010-500

Telefones para contato: (42) 99248797

e-mail: *alisonrogerweber@gmail.com*

Nome completo: **Cesar Augusto Januario**

Filiação institucional: Universidade Tecnológica Federal do Paraná Campus Ponta Grossa

Departamento: COINF

Função ou cargo ocupado: Estudante de Graduação

Endereço completo para correspondência (bairro, cidade, estado, país e CEP):

Av. Monteiro Lobato, km 4, s/n
Santa Mônica
Ponta Grossa
Brasil
84010-500

Telefones para contato: (42) 30287815

e-mail: *januario.cesar@gmail.com*

Nome completo: **Simone Nasser Matos**

Filiação institucional: Universidade Tecnológica Federal do Paraná Campus Ponta Grossa

Departamento: COINF

Função ou cargo ocupado: Professora

Endereço completo para correspondência (bairro, cidade, estado, país e CEP):

Av. Monteiro Lobato, km 4, s/n
Santa Mônica
Ponta Grossa
Brasil
84010-500

Telefones para contato: (42) 3220-4827

e-mail: snasser@utfpr.edu.br

Recebido para publicação em: 26/10/2010
Aceito para publicação em: 02/12/2010