UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

NOELI ROGALSKI

ESTUDO DA MDA - UMA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARES BASEADA EM MODELOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2012

NOELI ROGALSKI

ESTUDO DA MDA - UMA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARES BASEADA EM MODELOS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profª. Simone de Almeida

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

ESTUDO DA MDA - UMA METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARES BASEADA EM MODELOS

por

NOELI ROGALSKI

Este Trabalho de Conclusão de Curso (TCC) foi apresentado(a) em 05 de junho de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O(a) candidato(a) foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Simone de Almeida
Profª. Orientadora

Danillo Leal Belmonte
Membro titular

Eliana Claudia M. Ishikawa
Membro titular

Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

Simone de Almeida
Coordenador do Curso
UTFPR - Câmpus Ponta Grossa

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso

AGRADECIMENTOS

Neste instante o meu agradecimento é direcionado primeiramente a Deus, pois foram muitas dificuldades vencidas ao longo do caminho, agradeço pela sua benção, e por estar me proporcionando esta felicidade de cumprir mais um objetivo.

Agradeço a professora Simone Almeida, pela orientação na realização deste trabalho, pela paciência que teve comigo mediante meus desafios, e mesmo com essa paciência, manteve sua competência e profissionalismo admirável.

Aos professores do Curso de Análise e Desenvolvimento de Sistemas, pela ajuda, atenção e carinho, prestados ao longo dos anos.

Aos meus amigos que participaram desta jornada de trabalhos, provas e companheirismo.

Ao Andre Otto, amigo, companheiro, que esteve sempre disposto a ajudar.

“O sucesso é a soma de pequenos esforços repetidos dia após dia”

(Rober Colliert)

RESUMO

ROGALSKI, Noeli. **Estudo da MDA – Uma metodologia de desenvolvimento de softwares baseada em modelos.** 2012. 49 f. Trabalho de Conclusão de Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012

Este trabalho propõe o estudo da Metodologia MDA (*Model Driven Architecture)* - Arquitetura Dirigida a Modelos e sua aplicação utilizando a ferramenta ACCELEO com a finalidade de estudar uma alternativa à automatização do desenvolvimento de software com o aproveitamento dos modelos gerados na etapa de modelagem do programa. Este estudo demonstra o desenvolvimento de código fonte, gerado de forma automática, assim como os componentes de um artefato de software e documentação textual através do mapeamento automático dos modelos, favorecendo um aumento de produtividade, o incremento qualitativo dos produtos pelo uso de padrões, favorecendo o aproveitamento dos custos de modelagem, pois os modelos passam a ter papel ativo na construção do produto final. O estudo da MDA foi relevante à aplicação, a construção de um Sistema Jurídico, que consiste no controle de processos na área jurídica, e os resultados obtidos foram satisfatórios, pois avaliou o aumento da produtividade na geração de código.

Palavras-chave: MDA. ACCELEO. Geração de código. UML. Melhoria de produtividade. Sistema Jurídico.

ABSTRACT

ROGALSKI, Noeli. **Study of MDA - A software development methodology based on models.** 2012. 49 f. Course Technology Analysis and Systems Development Final Paper – Federal Technology University - Paraná. Ponta Grossa, 2012.

This paper proposes the study of the Model Driven Architecture *(*MDA*)* – Oriented Architecture Model and your application using the ACCELEO tools in order to study an alternative to automation of software development and utilization of models generated in the modeling stage of the program. The study demonstrates the development of source code, generated automatically, well as the components of a software artifact and textual documentation through the automatic mapping of models, favoring an increase in productivity, increased quality of products by the use of standards, favoring the use of cost modeling, as the models start to take an active role in building the final product. The study of MDA was relevant to the application, the construction of a Legal System, which consists of process control in the Legal field, and the results were satisfactory, as assessed increased productivity in code generation.

Keywords: MDA. ACCELEO. Code Generation. UML. Improvement of productivity. Legal System

LISTA DE FIGURAS

LISTA DE SIGLAS

CIM	*Computation Independent Model*
MDA	*Model Driven Architecture*
MOF	*Meta Object Facility*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
UTFPR	Universidade Tecnológica Federal do Paraná
UML	*Unified Modeling Language*
DAO	*Data Access Object*
XML	*Extensible Markup Language*
XMI	*XML Metadata Interchange*
CORBA	*Common Object Request Broker Architecture*
MOFM2T	*Model to Text Transformation Language*
EPL	*Eclipse Public License*
IDE	*Integrated Development Environment*
GPL	*General Public License*

SUMÁRIO

1 INTRODUÇÃO

Com o objetivo de alcançar maior agilidade no desenvolvimento de sistemas, e atingir os benefícios inerentes, têm surgido no mercado muitas ferramentas de geração de código, que se utiliza de *templates*, assistentes, desenvolvimento declarativo e outros recursos.

A arquitetura MDA (*Model Driven Architeture*) é um exemplo de metodologia e pode ser aplicada na modelagem de projetos de sistemas a partir da abstração de seus requisitos até a geração de código (OMG, 2003).

A OMG (*Object Management Group*), em busca de uma alternativa para a automatização do desenvolvimento de *software,* aproveitando os modelos gerados na etapa de modelagem, criou o MDA, ou seja, Arquitetura Orientada a Modelos, como alternativa para o aproveitamento da documentação para a geração de código de forma automatizada, e com o intuito de dar um passo evolutivo no processo de desenvolvimento de *software*.

Esta arquitetura busca, além do aproveitamento dos artefatos de documentação para a geração automática de código, permitir a especificação de um sistema de forma independente da plataforma e prover meios de transformação desta especificação para plataformas específicas.

A busca progressiva por um nível maior de abstração contínua, e junto com ela busca-se também a independência de plataforma, ou seja, a portabilidade, cujo pressuposto é que um artefato produzido em uma plataforma possa ser executado em outra sem nenhuma modificação. A complexidade desta tarefa não pode ser ignorada, visto que os recursos específicos da plataforma devem ser implementados para permitir tal objetivo.

Para construir programas é de relevante importância da modelagem e contextualização dos benefícios da arquitetura, pois a modelagem não somente conduz a sistemas mais fáceis de serem desenvolvidos, integrados e mantidos, mas também porque permite automatizar pelo menos parte de sua construção.

A Arquitetura Dirigida em Modelo possibilita diminuição dos custos da análise e documentação do sistema, que geralmente é significativo, através da diluição destes na etapa de desenvolvimento, fazendo então com que estes artefatos, antes apresentados somente como suporte de documentação e consulta, passem a ter um

papel mais ativo no processo de geração de código. Além disso, a velocidade de desenvolvimento por esta arquitetura é significativa, pois os processos automatizados são mais rápidos e precisos que os processos artesanais, e a arquitetura, graças a separação de conceitos, permite maior facilidade de manutenção, oferecendo a possibilidade de rastreamento entre especificação e código-fonte gerado.

A MDA contém camadas para separar a abstração do sistema do modelo de implementação. A geração de um código fonte é basicamente a ultima etapa no processo MDA e para agilizar este processo, a MDA propõe para que se utilize ferramentas de geração de código automatizadas, como por exemplo, o *plugin* ACCELEO criado pela fundação Eclipse para auxiliar na geração de código fonte a partir de um modelo UML(*Unified Modeling Language)*.

Assim esse trabalho propõe aplicar a metodologia MDA no desenvolvimento de um sistema de controle de processos jurídicos, com o intuito de explorar a metodologia e não de atender a todos os requisitos necessários em uma gestão jurídica.

1.1 PROBLEMA DE PESQUISA

O custo para que um *software* seja um aplicativo de qualidade, que satisfaça o que o cliente realmente deseja, é alto nos dias de hoje. Há todo um processo, que deve ser definido e implementado, tanto para requisitos do sistema, como para a modelagem, arquitetura e implementação do sistema. O processo falho destes artefatos pode ocasionar atrasos na entrega do produto, *bugs* no produto e insatisfação do cliente.

O grande problema na produção de *softwares* são os requisitos do sistema. O cliente deseja algo, que é modelado e implementado de outra forma. O conhecimento sobre o sistema, modelos nem implementações são eficazes. A produção de criação de código fonte é lenta influenciando o atraso da finalização do produto.

1.2 OBJETIVO GERAL

Pesquisar e apresentar a arquitetura MDA demonstrando seus conceitos e aplicando a sua definição em um protótipo de *software* para um sistema jurídico.

1.3 OBJETIVOS ESPECÍFICOS

- Pesquisar e demonstrar conceitos, fases e modelos da metodologia MDA.
- Demonstrar o uso de uma ferramenta para Automatização de código fonte chamada ACCELEO.
- Empregar a arquitetura MDA em um protótipo para um *software* de sistema jurídico.
- Analisar os benefícios da utilização da metodologia MDA na construção de um sistema de informação.

1.4 ESTRUTURA

Este trabalhado está estruturado em cinco capítulos, incluindo a presente introdução. O capítulo 2 apresenta os conceitos necessários para o entendimento da MDA, descreve o Ciclo de vida de *software* com MDA e aborda os conceitos relacionados ao processo de transformação de modelos (CIM, PIM, PSM).

O capítulo 03 traz uma revisão da literatura acerca do tema. O foco principal deste capítulo é a apresentar o procedimento realizado na gestão jurídica no que se refere ao controle de processos.

O capítulo 04 descreve em detalhes a aplicação da metodologia MDA no desenvolvimento de um sistema de controle de processos jurídicos, desde a criação do modelo de domínio utilizado até a geração dos artefatos didáticos gerados por meio da utilização da ferramenta ACCELEO.

No capítulo 05 são apresentadas as conclusões obtidas no desenvolvimento deste trabalho, como algumas propostas de trabalhos futuros.

2 MODEL DRIVEN ARCHITECTURE

Este capítulo aborda os conceitos da arquitetura MDA, todos os processos que envolvem a arquitetura bem como seu ciclo de vida, seus modelos e como ocorre a transformação entre as camadas no MDA. Ele está dividido em três seções. A seção 2.1 apresenta alguns conceitos. A seção 2.2 identifica e discorre sobre as fases da metodologia MDA e a seção 2.3 apresenta algumas considerações sobre o capítulo.

2.1 CONCEITO

Arquitetura de sistema é a especificação de componentes de *software*, o estabelecimento de inter-relações e regras para estes componentes, e sua evolução é nitidamente notado a cada dia (FRAVE, 2010).

Com a emergente aplicação da internet, o problema de interoperabilidade entre os sistemas, se torna um impasse na construção de sistemas, devido à má organização e aplicação de metodologias, no que diz respeito a requisitos do sistema, e a incorreta implementação e modelagem, o qual não possibilita uma visão futura e a integração com novos sistemas.

Em 2001 a OMG (*Object Management Group*) adotou um padrão chamado MDA, sendo ele um *framework* arquitetural para ampliar a proporcionar uma melhor portabilidade, interoperabilidade e reusabilidade, por meio da separação de preocupações (MDA, 2003). A MDA é construída sobre os padrões da OMG incluindo XML (*Unified Modeling Language*), XMI (XML *Metadata Interchange*) e CORBA.

A MDA utiliza para construir seu ciclo de vida, os chamados modelos, que são todos os artefatos como especificação de requisitos, descrição da arquitetura, descrição do modelo e código.

O trabalho da MDA é aproximar um sistema independente de plataforma, produzir primeiramente a abstração do sistema, como seria o sistema verificando em uma visão global. Posteriormente, com esta visão, passaria para o outro módulo o qual definiria a implementação.

2.2 FASES DA MDA

A MDA define *frameworks* que separam as especificações da funcionalidade do sistema de sua implementação em uma plataforma específica. A ideia da MDA é gerenciar estes modelos que possam ser usados para gerar componentes e aplicações executáveis. Os modelos definidos pela MDA são descritos abaixo e serão vistos com mais detalhes nos próximos tópicos.

- **Computação Independentemente de Modelo (CIM):** É um modelo que descreve um sistema a partir da computação independente do ponto de vista (OMG, 2003).
- **Plataforma Independente do Modelo (PIM):** É uma visão abstrata do sistema, de um nível mais alto e que independe de qualquer implementação de tecnologia (OMG, 2003).
- **Plataforma Específica do Modelo (PSM):** É um modelo adaptado para especificar o sistema na construção da implementação e disponível em uma plataforma específica (OMG, 2003).

A Figura 1 demonstra em camadas horizontais como a estrutura do MDA é modelada. Cada camada são fases partindo primeiramente do CIM, e concluindo ciclo no código.

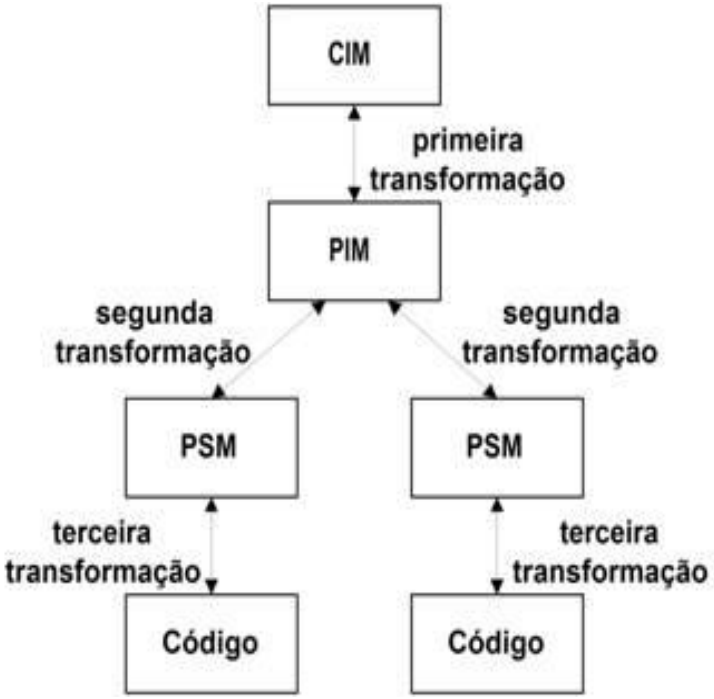


Figura 1: Estrutura MDA
Fonte: Silva (2008, p.1)

A primeira fase inicia-se no CIM, que após a sua transformação segue no ciclo com o PIM. A partir do PIM, podem-se ter muitos PSMs e cada PSM segue na geração do código fonte.

A Figura 2 representa as fases da MDA de forma descritiva. Citando as principais características de cada fase.

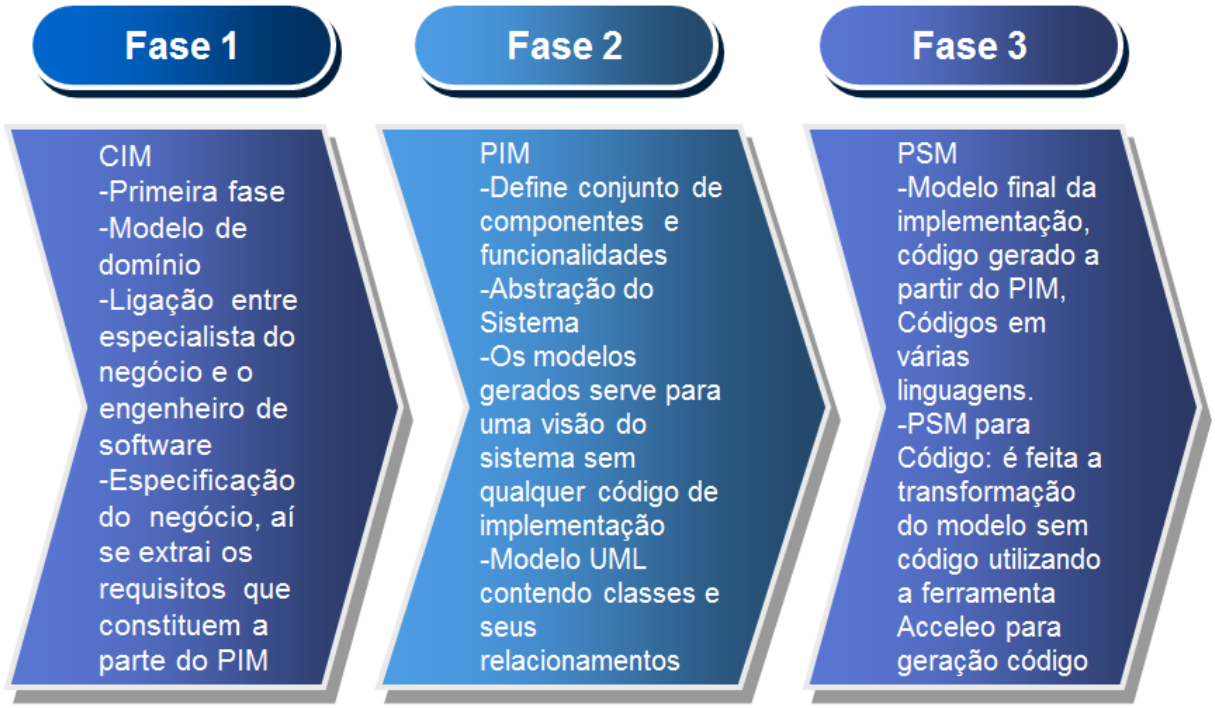


Figura 2: Fases da MDA
Fonte: Autoria Própria

2.2.1 Modelo CIM

O modelo CIM (Computação Independentemente de Modelo) é um importante modelo no MDA. É onde se inicia a primeira fase, ou seja, o entendimento adequado do negócio, especializando-se nos processos e determinando o que necessariamente o sistema que irá automatizar o determinado processo deve fazer. Portanto a partir da especificação do negocio é que se têm os requisitos que constituem a parte do PIM.

O CIM é independente de como o sistema irá ser implementado. Geralmente, ele é chamado de modelo de domínio e pode ser expresso usando modelos de negócios. Ele auxilia na ponte entre os especialistas do domínio e o engenheiro de

software. O CIM pode consistir de modelo UML e outros modelos de requisitos (FAVRE, 2010).

2.2.2 Modelo PIM

O modelo PIM foca no ponto de visão de operações independente de plataforma. Modelos de lógica e análise são tipicamente independem de implementação e de plataforma específica e podem ser considerados PIM (FAVRE, 2010).

O PIM define um conjunto de componentes e funcionalidades, que são definidas independentemente de qualquer plataforma, ou seja, sem a necessidade de adicionar no modelo tecnologias, que serão utilizadas no projeto. Ele por de ser visto como uma abstração do sistema que pode ser realizado por diferentes plataformas específicas.

Os modelos gerados na etapa do PIM podem conter artefatos para que se tenha uma visão do sistema sem qualquer código de implementação, podendo ser uma modelo UML, demonstrando classes e seus relacionamentos, pois com UML é independente de uma plataforma específica.

Para auxiliar a criação da plataforma independente, pode-se destinar um modelo de sistema para uma máquina virtual tecnologicamente neutra. Uma máquina virtual é definida como um conjunto de partes e serviços que independente de qualquer plataforma específica e que podem ser realizadas em diferentes plataformas.

A Figura 3 ilustra um modelo PIM. É um modelo UML de diagrama de classe.

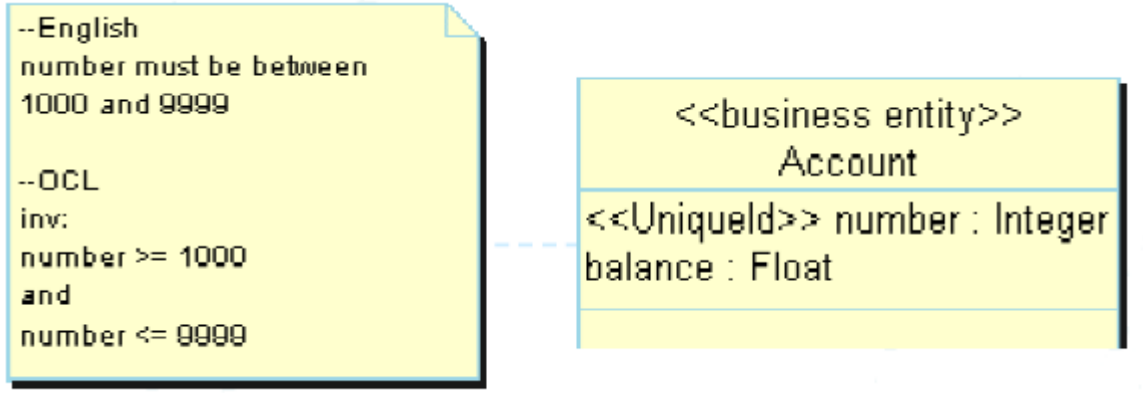


Figura 3: Modelo de PIM
Fonte: Trinta (2010)

A Figura 3 ilustra uma classe com o tipo *business entity*, e um atributo *number*. Este modelo é um exemplo de como o modelo PIM pode ser definido. Esta classe irá ser transformada em código no modelo PSM.

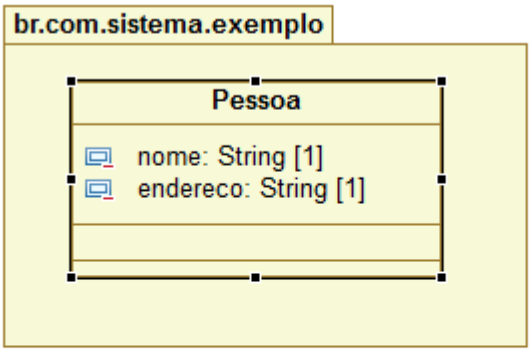
2.2.3 Modelo PSM

O PSM descreve o modelo final da implementação, ou seja, o código fonte gerado a partir do modelo PIM. Como exemplo pode-se ter códigos gerados em Java, JavaEE, .Net, etc. Ele é a visão do sistema em uma plataforma específica que combina o PIM com detalhes específicos de implementação. Ele inclui um conjunto de representações de conceitos de tecnologias de diferentes partes providos pela plataforma.

Um modelo PIM pode prover muitos modelos PSM. Esta é a forma que a arquitetura MDA trabalha. Como o PIM é um modelo, como por exemplo, um diagrama de classe, pode-se ter muitos modelos de diagrama, geralmente *software* é baseado em muitos pacotes, cada um contendo muitos modelos UML e cada modelo UML posteriormente se transformará em um modelo PSM, contendo as classes que foram geradas a partir do modelo PIM.

A Figura 4 ilustra um modelo UML contendo uma classe e logo abaixo o código que foi criado por uma ferramenta de geração de código a partir de um modelo. Posteriormente será comentado e demonstrado o uso da ferramenta ACCELEO para geração de códigos fonte, a partir de um modelo PSM.

A Figura 4 apresenta um modelo PSM contendo a classe Pessoa com seus atributos e logo abaixo o modelo de código gerado, contendo o código fonte na linguagem Java, com atributos e métodos *set* e *get*.




```java
package br.com.sistema.exemplo;

public class Pessoa {

    private String nome;
    private String endereco;

    public String getEndereco() {
        return endereco;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Figura 4: Modelo PSM
Fonte: Autoria Própria

2.2.4 Transformação

O modelo de transformação refere-se ao processo de converter um modelo dentro de outro modelo do mesmo sistema preservando uma relação entre ambos. A transformação pode ocorrer tanto de cima da hierarquia (CIM, PIM e PSM) tanto utilizando a engenharia reversa alterando assim o início da transformação.

A arquitetura MDA provê também transformação entre:

- **PIM para PIM:** É utilizado apenas para refinamento do modelo. Esta transformação é feita manualmente e não há adoção de tecnologia ou implementação de código.
- **PIM para PSM:** É a transformação mais utilizada na arquitetura MDA, passando da abstração do modelo de sistema, para a conversão de modelos, como exemplo, UML.
- **PSM para Código:** É feita a transformação do modelo que define o relacionamento que o sistema terá sem código para o modelo de código fonte, utilizando ferramentas de geração de código a partir de um modelo específico.
- **PSM para PIM:** Se torna útil quando é necessário, a partir de um modelo, como exemplo UML, saber o modelo abstrato do sistema, ou seja, o PIM.
- **PIM para Código:** Nesta transformação o modelo PIM é transformado diretamente para o código manualmente.

Algumas transformações ocorrem manualmente como é o caso do modelo PIM para o código. No caso do PSM para o código, o auxílio de uma ferramenta de geração de códigos fonte é necessário.

O capítulo 4 aborda um sistema jurídico, o qual irá demonstrar os passos para criação do modelo PSM e também uma ferramenta para geração de código fonte.

2.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foram apresentados os conceitos de uma arquitetura MDA, suas fases e transformações. Estes conceitos serão aplicados posteriormente nos próximos capítulos, onde será primeiramente descrito sobre como os processos dentro de um sistema jurídico demonstrando a fase PIM do ciclo de vida da arquitetura MDA. Em seguida, a apresentação do protótipo ilustrando a criação de um sistema jurídico a partir de modelos UML.

3 DESCRIÇÃO DA GESTÃO JURÍDICA EM UM ESCRITÓRIO DE ADVOCACIA

Este capítulo trata da gestão Jurídica, suas particularidades, regras e trâmite de documentos dos escritórios de advocacia. O capítulo contém as seguintes seções: 3.1 Apresentam as principais funcionalidades de um escritório jurídico e vantagens de se ter o processo informatizado, a seção 3.2 identifica as principais demandas dos clientes para processamento pelo escritório, a seção 3.3 apresentação do fluxo de trabalho do sistema jurídico e por fim 3.4 considerações sobre o capítulo.

3.1 INTRODUÇÃO

A arte da tecnologia, desenvolvimento e praticidade, são fatores determinantes à logística necessária às atividades dos contemporâneos operadores do Direito(ROHRMANN, 2008).

A relação custo x benefício em um sistema de gerenciamento de processos é de grande relevância nos dias atuais onde a informação deve estar sempre de forma rápida e presente ao profissional da área jurídica, informações que permite uma precisa decisão em um determinado andamento processual.

Com a digitalização de processos jurídicos e/ou digitalização das informações desses processos em *software* específico é uma realidade em todo o Brasil. Inclusive já existe uma legislação específica vigente sobre o tema.

A digitalização de processos judiciais e documentos administrativos facilita a gestão, procura visualização e compartilhamento da informação. A economia de espaço e os ganhos de produtividade devem ser considerados. O acompanhamento das informações processual proporciona ganho de tempo e organização.

A aplicação da metodologia de estudo, é um *software* que proporciona o gerenciamento dos processos do departamento Jurídico e escritórios de advocacia. O sistema realiza o cadastro de clientes, andamento das ações, pesquisas, entre outras funcionalidades. É possível identificar alguns benefícios na utilização de um sistema de controle de processos jurídicos, tais como:

- Acesso fácil de qualquer computador conectado a rede.

- Garantia de segurança e fidelidade das informações.
- Promove mais rapidez e menos burocracia no dia a dia do profissional jurídico.
- Centralização de documentos, processos e contratos.
- Consulta de andamentos com maior agilidade.
- Agiliza todos os procedimentos, sejam processuais ou administrativos, reduzindo os gastos, aumentando a confiabilidade das informações, e acima de tudo, potencializando a capacidade de ampliação da carteira de clientes.
- Trata-se de um *software* de acompanhamento de processos um mecanismo que efetivamente facilita a grande maioria das rotinas diárias do escritório, permitindo ao gestor um amplo controle de todas as áreas do escritório: cível, trabalhista, previdenciária, penal, etc., e a administração ao controle de publicações de despachos e intimações, do monitoramento à gestão adequada de documentos.

Um sistema de controle de processos jurídicos, geralmente atende as seguintes funcionalidades:

- Controle de contencioso e pessoas.
- Cadastro de pessoa física, jurídica, advogado e órgãos judiciais.
- Cadastro de processos, o cadastro das informações básicas do processo (número, tipo de ação, cliente, etc.) bem como alguns dados relacionados como partes, fases, andamentos e pedidos.

O protótipo deste trabalho visa à organização de processos jurídicos tal como, cadastro de processos e visualização para que o usuário tenha mais controle e informações sobre o mesmo. Propõe uma melhor identificação, uma busca mais rápida aos dados.

3.2 DEMANDAS DOS CLIENTES

O escritório de advocacia ou o órgão jurídico de uma empresa recebe uma série de insumos (documentos, *e-mails*, visitas e pleitos de clientes, etc.) para processamento, por exemplo:

- Consultas que terão como resposta pareceres jurídicos;

- Citações judiciais e autuações extrajudiciais recebidas por seus clientes atuais e futuros, que terão como saída às defesas que se fizerem necessárias nos respectivos feitos, judiciais ou administrativos;
- Pedidos de patrocínio de medida extrajudicial ou judicial para fazer valer o direito dos clientes, que terão como saída às ações cabíveis;
- Ocorrências (andamentos) processuais, administrativas ou judiciais, relativamente aos feitos diversos sob patrocínio ou defesa pelo escritório, seja através de acompanhamento direto nas diversas instâncias judiciais ou administrativas, seja através de notificações, intimações e publicações na imprensa oficial, seja através de acompanhamento pela Internet ou por e-mail recebido, a saída será a providência judicial ou administrativa adequada a cada ocorrência (participação em audiência designada, petição ou interposição de recurso, depósito judicial ou pagamento de custas);
- Pedido de presença e assessoramento jurídico em reuniões, assembleias e atos diversos (compra de imóveis, transações e acordos, etc.).

3.3 APRESENTAÇÃO DO FLUXO DE TRABALHO DO SISTEMA JURÍDICO

Algumas das providências do escritório podem ser realizadas de pronto ou com único ato pelo advogado, como um aconselhamento jurídico ou a emissão de um parecer; outras poderão demandar providências continuadas no tempo, como o patrocínio ou defesa do cliente em ação administrativa ou judicial, nas diversas instâncias.

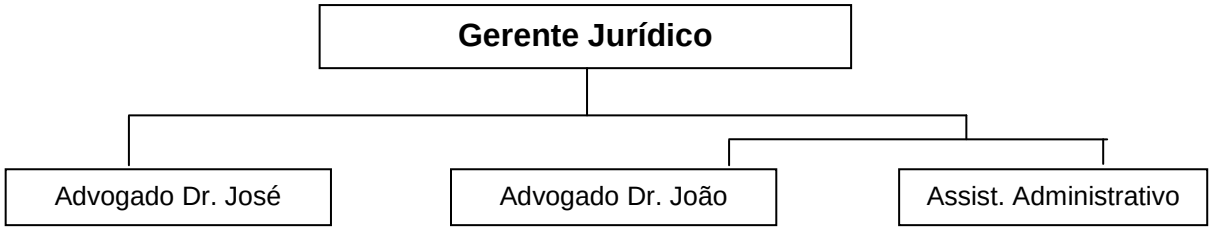


Figura 5: Representação usuários do Sistema Jurídico
Fonte: Autoria Própria

Na figura 5, a título de exemplo, o caso do departamento jurídico de uma empresa A, a representação usuários, que conta com um Gerente Jurídico, dois Advogados empregados e um assistente administrativo.

Pode ser resumido no fluxograma apresentado abaixo o trabalho do escritório que o ali designado de Sistema Jurídico serve para o registro de todas as entradas, após preliminarmente despachadas pelo Gerente Jurídico, que assinala uma prioridade e data esperada de término da providência demandada, nos assuntos não processuais, e também nas consultas que demandarão parecer jurídico.

Nesse sistema são também registradas as saídas e ele é ainda usado para o registro das diversas ações judiciais ou administrativas e suas respectivas ocorrências (andamentos).

A figura 6 ilustra o fluxo de um sistema jurídico, desde quando o escritório recebe a documentação do cliente, este documento é analisado pelo gerente jurídico e despachado para a secretária para realizar o cadastro no sistema, este cadastro é inserido os dados do cliente e a documentação trazida por ele. Através do sistema o advogado verifica se há requisição pendente e realiza todos os efeitos necessários, por exemplo, se necessário emitir citação, preparar uma defesa, parecer técnico. O advogado despacha para o gerente jurídico e determina providencia dando continuidade ao que se deve ser feito, registrado providencia, secretária atualiza dados no sistema e faz a expedição para arquivo.

Caso o documento não tenha nenhuma requisição de serviço o gerente jurídico despacha determinando providencia de registro em sistema pela secretária e expedição ao arquivo.

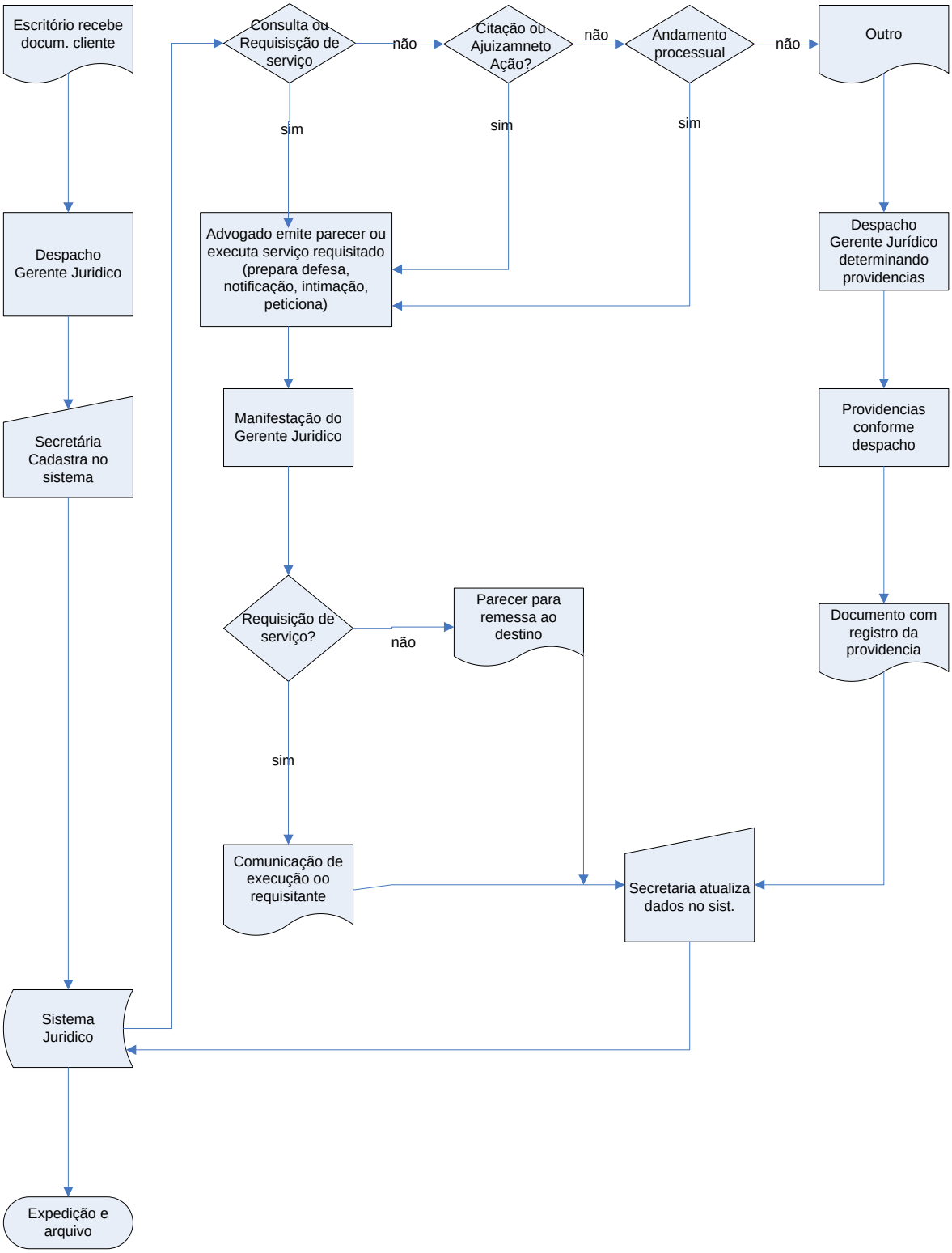


Figura 6: Fluxograma representando o trabalho do escritório
Fonte: Autoria Própria

A aplicação arquitetural MDA, na construção do modelo PIM, o seu desenvolvimento é feito a partir desta descrição acima, através de ideias abstratas relacionadas ao andamento e prosseguimento das ações realizadas.

3.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

Nesta seção foi descrita a gestão jurídica e seu fluxo de organização, mas relevante ainda foi abordado à aplicação e o desenvolvimento do modelo PIM que consiste a aplicação arquitetural MDA na construção do modelo PIM do Sistema Jurídico. O sistema Jurídico de uma empresa abrange outras funcionalidades como exemplo, departamento financeiro, contábil e administrativo, mas somente parte deste Sistema Jurídico será informatizada pelo sistema proposto neste trabalho, como por exemplo, o cadastro, a consulta, a exclusão, a atualização dos processos, e de pessoas.

Em qualquer hipótese, o escritório, no fluxo que desenvolver ou implantar para o trabalho necessário, terá necessidade de registrar a entrada (providências), e a saída ou resultado obtido. Será de grande utilidade um gerenciador de banco de dados, para viabilizar tais registros, mas para processar referidos dados judiciais e extrajudiciais, fornecendo informações necessárias às operações e decisões. Essas informações devem, ainda, ser integradas ao arquivo documental que se faça indispensável.

4 PROTÓTIPO BASEADO NA ARQUITETURA MDA UTILIZANDO ACCELEO

Este capítulo trata sobre os conceitos de ferramentas de geração de código, bem como a utilização destes conceitos na prática, demonstrando um sistema criado a partir de modelos. O capítulo contém as seguintes seções: 4.1 Principais conceitos, 4.2 ACCELEO, 4.3 Construção da arquitetura, e por fim a seção 4.5. Faz algumas considerações sobe o capítulo.

4.1 CONCEITOS

Após a criação do PIM com toda a abstração do sistema e a informação de todo fluxo de processos, inicia-se a fase do PSM, utilizando-se de diagramas para definir escopos referentes à implementação. A partir dos diagramas são geradas as classes padrões do modelo de arquitetura, que após este passo são redefinidas as regras de negócios bem como a persistência de dados e a camada de visão.

A construção de classes autimatizadas é gerada a partir de outras ferramentas externas ou não a IDE. Atualmente existem ferramentas tais como AndroMDA, *Eclipse Modeling Framework*, ACCELEO, etc. Todas estas ferramentas são baseadas na arquitetura MDA.

Para gerar as classes foi utilizada a ferramenta ACCELEO devido a facil documentação e utilização. Ele é um *plugin* que foi criado para o eclipse com o propósito de gerar classes a partir de diagramas UML, podendo definir, como as classes serão geradas para que métodos, comentários e configurações sejam implementados de forma automática ao gerar as classes pela ferramenta deixando assim o sistema padronizado referente à sua arquitetura. Estas informações serão abordadas com mais detalhes no decorrer do capítulo.

Para ilustrar a utilização do ACCELEO foi criado um protótipo que foi implementado na linguagem de programação Java e na IDE eclipse Galileo. O desenho da arquitetura do protótipo foi baseado em MVC, o qual define as três camadas do sistema para a transação de mensagens de uma camada para outra. Como o foco não é o sistema, e sim a ferramenta para a criação da classe a partir de uma modelagem de diagrama, o protótipo terá uma visão básica do

funcionamento de um sistema jurídico, como cadastro de pessoas e de processos bem como o *status* do processo. O sistema irá conter janelas para que se possa caminhar pelo processo. As classes da camada de visão serão ilustradas em diagramas de classe também, mas como a ferramenta que foi utilizada, chamada papyrus, não contém componentes de tela como botões e tabelas para UML foram somente criados os relacionamentos e métodos que cada classe possui.

Este capítulo é destinado à implementação e configuração do sistema proposto a partir da ferramenta de geração de classe. Inicialmente será abordada a ferramenta ACCELEO, como são geradas as classes e o que se pode configurar e definir por meio dela. Também será explicado todo o processo da construção do modelo UML para a camada de modelo e para a camada de controle.

4.2 ACCELEO

O ACCELEO é uma ferramenta que trabalha sobre a perspectiva do MDA e permite transformar modelos UML, MOF (*Meta Object Facility*) e EMF (*Eclipse Modeling Framework*) em códigos de linguagem de programação (ACCELEO). O projeto MDA é *open source,* e é licenciado pela EPL (*Eclipse Public License*). Ele incrementa a produtividade do desenvolvimento e possui características como:

- **Completa integração com ambos ambientes do eclipse e *framework* EMF.** Os *frameworks* EMF são destinados a toda a modelagem de dados de um sistema, ou seja, auxilia na utilização de diagramas de classe, casos de uso, atividade etc.
- **Sincronização do modelo com o código:** A medida com que o sistema é modelado é possível atualizar as classes de acordo com as alterações realizadas no modelo.
- **Fácil atualização e gerenciamento de *templates*:** O desenho da classe é feito por configurações do ACCELEO, estas configuração podem ser de acordo com a modelagem de classes que deseja, ou seja, pode-se adicionar comentários padrões e padronizar implementação de métodos.

A ferramenta consegue transformar a modelagem em várias linguagens de programação utilizando *scripts* criado pela OMG. O ACCELEO irá utilizar este *script*

para gerar códigos para a linguagem que deseja. Dentre os *scripts* para a transformação, se pode citar as linguagens Java, .NET, PHP, C#, C++, entre outras que possuem *scripts* prontos. É possível também criar um *script* para a linguagem desejada através do ACCELEO ou também alterar a implementação de um *script* de uma linguagem já definida. Para o sistema proposto, não será criado um *script* para gerar os código Java e sim utilizar um *script* pronto já definido pelo OMG.

4.2.1 Funcionamento ACCELEO

Um modelo é uma abstração da realidade. É utilizado esquemas de modelagem para representar partes, conceitos de um determinado programa. A modelagem é o inicio para trabalhar com a ferramenta ACCELEO. Modelagens são diagramas para auxiliar na visão do fluxo de dados que pode ser uma visão abstrata ou uma visão concreta de como os dados devem trafegar pelas diferentes entidades.

Para facilitar o entendimento de modelagem, a Figura 6 demonstra um diagrama de classes contendo um relacionamento entre um cliente com suas e compras, produtos e categorias.

A Figura 7 ilustra um exemplo de um diagrama de classe. Este exemplo não faz parte do protótipo do sistema jurídico, mas pode ser utilizado para demonstrar como um modelo UML é criado. O modelo contém classes, atributos e relacionamento. Neste exemplo o atributo endereço pode ser decomposto em outros atributos como UF, número, CEP, bairro. Mas o foco não é enfatizar a definição de construção de classe. A partir deste modelo é que o ACCELEO irá transformar para classes concretas que posteriormente se tornarão o sistema.

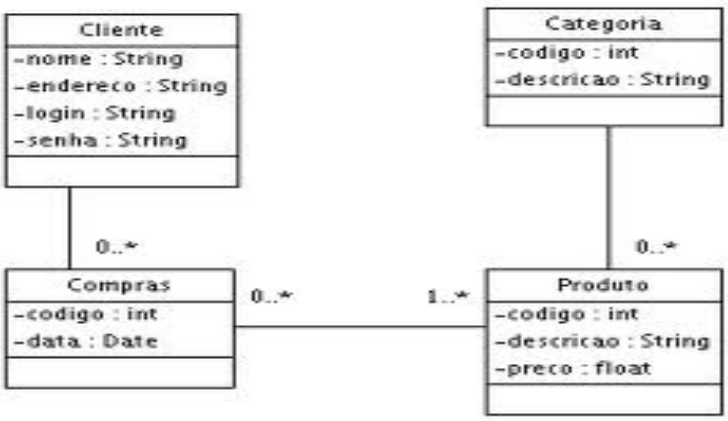


Figura 7: Representação do diagrama de classe
Fonte: Tutorial UFBA

Após o modelo criado, o ACCELEO utiliza *script* e se encarrega de transformar o arquivo UML em classes. Este *script* como citado anteriormente foi criado pela OMG (*Object Managment Group*) que definiu um padrão para transformação de modelo em código fonte. A Figura 8 demonstra as etapas para esta transformação. Depois de criado o modelo como ilustra na Figura 8, *Step 1:Model* , é utilizado um script (*Step 2: Choose your script*) que tem uma extensão de arquivo .mtl, e a partir dele, o modelo é transformado para a linguagem de programação (*Step 3: Generated*) para a qual este *script* foi destinado, gerando as classes do modelo.

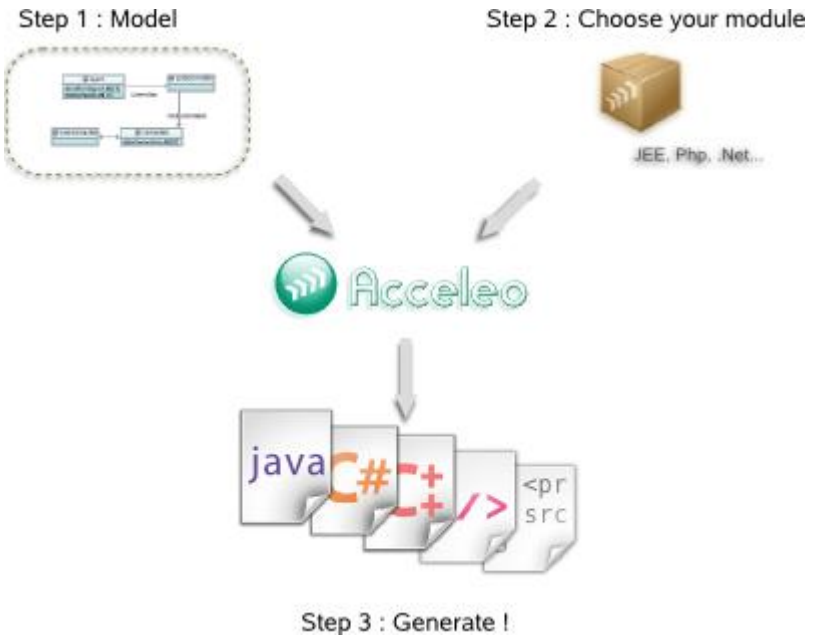


Figura 8: Etapas ACCELEO
Fonte: ACCELEO (2010)

4.2.2 Templates utilizados pelo ACCELEO

Quando as classes são geradas manualmente, é necessário adicionar comentários para que se gere posteriormente uma documentação do sistema que está sendo implementado. Também adicionados alguns métodos que poderiam ser métodos padrões quando a classe é gerada. Todas estas configurações podem ser alteradas utilizando a linguagem XMI (será visto no próximo tópico), a qual foi criada pelo OMG. E o ACCELEO as utiliza para criar o *template* que irá modelar a classe.

Os *templates* tem extensão.mtl e pode ser editado pelo eclipse. A Figura 9 mostra um exemplo da nomenclatura do arquivo.

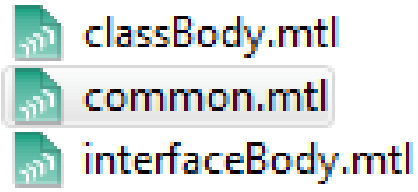


Figura 9: Templates ACCELEO
Fonte: ACCELEO (2010)

Estes arquivos contêm as configurações de como o desenho das classes serão geradas. A Figura 8 ilustra três arquivos, isto quer dizer que foram definidos três configurações, um para o corpo das classes, uma comum e outra para o corpo de uma classe do tipo *interface.*

Podem-se criar quantos *templates* forem necessários, por exemplo, neste caso, poderia criar outro *template* contendo o desenho de uma classe que é do tipo *Enum*.

A Figura 10 ilustra a linguagem utilizada na criação de *templates*. Neste exemplo o número um (1) referencia o nome do autor, e este foi alterado para o nome que deverá aparecer em todas as classes geradas. O número dois (2) demonstra que podem alterar os comentários também em toda a classe. O número três (3) ilustra a visibilidade da classe, no exemplo (Figura 10) está como *public,* então todas as classes serão geradas como *public*, mas pode-se alterar adicionando parâmetros como é feito logo após com o tipo da classe (*abstract*) ainda na Figura 9.

Pode se notar que há códigos que não fazem parte da classe. Estes códigos contêm a implementação da especificação MOFM2T, o qual padroniza uma linguagem para XMI para criação de *templates* (MOFM2T SPECIFICATION) definida pela OMG para criação de *templates*. Conhecendo esta linguagem é possível implementar configurações para cada tipo de classe e estabelecer padrões para todas geradas pela ferramenta ACCELEO.

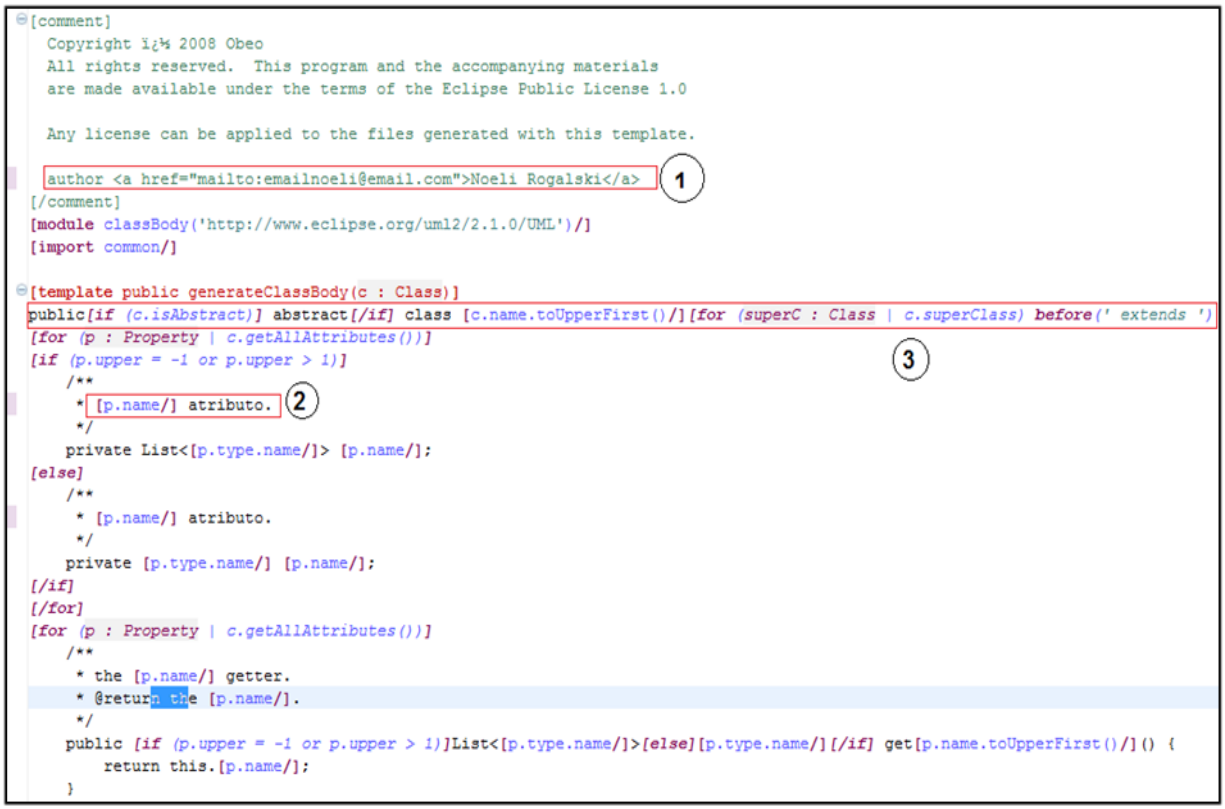


Figura 10: Linguagem para criação de Templates
Fonte: Autoria Própria

4.2.3 XMl

Para que o ACCELEO consiga transformar um modelo UML em classes é necessária à criação de um arquivo contendo metadados que estejam em uma linguagem padrão. Para isto, a OMG, criou um padrão de linguagem XMI (XML *Metadata Interchange*) que é um formato utilizado para compartilhar modelos usando XML e representa objetos conectados utilizando IDs de um mesmo arquivo ou arquivos diferentes (XMI).

O XMI contém elementos e atributos que definem como um objeto é associado com outro objeto, utilizando o padrão XML como base para construir o XMI.

O relacionamento das classes e todos seus atributos e métodos estão dentro do arquivo XMI, que por sua vez, é lido pelo ACCELEO e transformado por ele em classes. A Figura 11 ilustra um pequeno diagrama de classe UML e logo abaixo na mesma figura, o arquivo no formato XMI, correspondente ao arquivo de modelagem. Pode-se notar que o arquivo é no formato XML, mas com seus próprios elementos e

atributos. A partir das TAGs XML o ACCELEO lê e consegue verificar como a modelagem foi definida e cria as classes de acordo com os elementos e atributos do arquivo *.umlclass*, neste caso, example.umlclass.

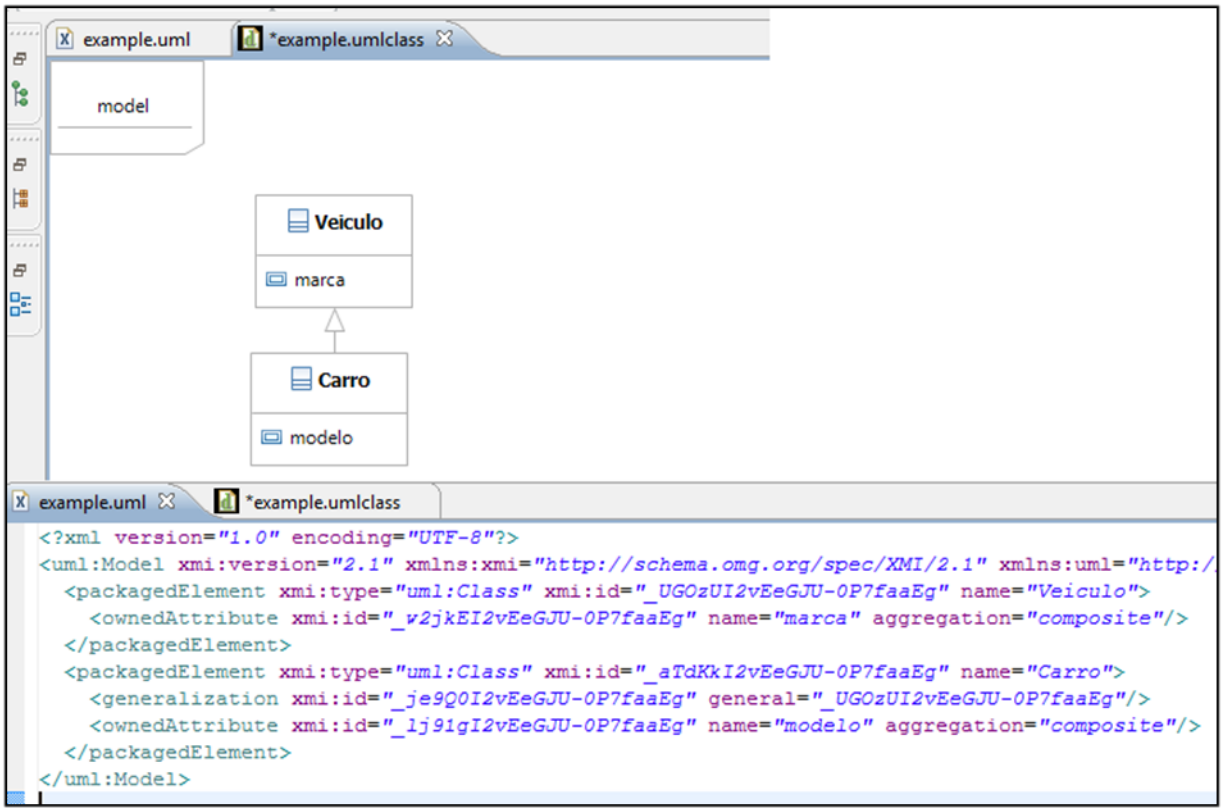


Figura 11: Linguagem XMI
Fonte: Autoria Própria

4.2.4 Papyrus

Papyrus é um ferramenta (*plugin)* para o eclipse, que provê a construção de modelos UML, ele é licenciado pela EPL (*Eclipse Public License*) e destina-se a todas as versões do eclipse lançadas até o momento (PAPYRUS, 2008).

Com o papyrus pode-se criar modelos de diferentes tipos como diagrama de classe, de estado, de uso, etc. A modelagem é de fácil criação, pois o fluxo para criar o modelo é arrastar e soltar a partir de uma paleta onde contém uma série de componentes para o diagrama que deseja.

Quando instalado no eclipse, o papyrus cria uma perspectiva para que todas as suas funções sejam mostradas para o usuário. Uma perspectiva é um ambiente

configurado no eclipse para o *plugin* instalado. A Figura 12 ilustra a perspectiva do papyrus iniciada no eclipse.

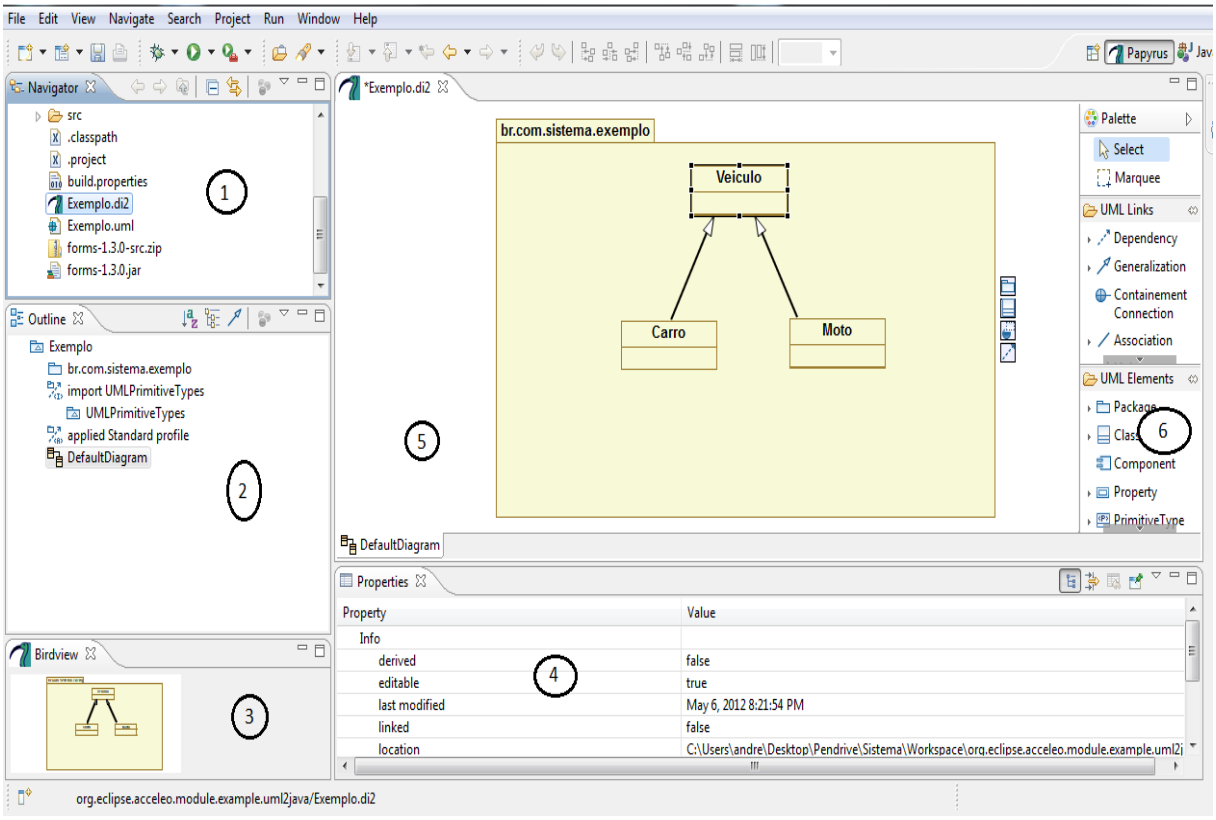


Figura 12: Perspectiva Papyrus
Fonte: Autoria Própria

Quando iniciada a perspectiva do papyrus no eclipse, ele abre todas as janelas mais utilizadas do *plugin.* No Figura 12 tem-se um exemplo. O número um (1) da Figura mostra o navegador, onde encontram-se os pacotes que foram criados para o sistema. O número dois (2), chamada de *outline* mostra uma visão de todos os componentes que já foram utilizados, como classes, relacionamentos, atributos, métodos etc.

Pelo *outline* consegue-se de maneira mais rápida, adicionar parâmetros de entrada e retorno de métodos. O *Birdview*, número três (3) mostra uma miniatura do modelo que está sendo criada na tela de modelo, número cinco (5). Na tela de modelo, são adicionados os componentes UML, o qual é arrastado da *palete*, número seis (6). A *palete* contém elementos, conexões para todos os diagramas, dependendo do diagrama criado a *palete* é alterado para a perspectiva do tipo do diagrama. E finalizando tem-se a janela *properties*, numero quatro (4), a qual é

possível adicionar parâmetro a métodos, alterar nomes das classes, alterar visibilidade de atributos, métodos e classes.

4.3 CONSTRUÇÃO DA ARQUITETURA

A construção do modelo foi baseada na arquitetura MVC, como citado no início do capítulo, onde se têm as camadas de visão, controle e modelo. A camada de modelo abrange também uma parte da persistência dos dados, o qual foi utilizado o padrão DAO (*Data Access Object*), e que permite separar as regras de negócio do banco de dados.

No eclipse, a IDE utilizada para o projeto, foi o editor UML Papyrus, citado anteriormente no capítulo 4.2.4.

4.3.1 Construção da camada de modelo

Para a construção da camada de modelo, foram criados dois pacotes contendo classes que remetem ao modelo do sistema, ou seja, onde contém as classes de entidades e também as classes que abstraem a persistência de dados chamada de DAO. O modelo foi separado em e pacotes:

- **br.com.sistemajuridico.modelo**: Este pacote contêm as classes de entidade do sistema jurídico
- **br.com.sistemajuridico.modelo.modeloDAO:** Este pacote contêm classes que são utilizadas para abstrair a persistência de dados no sistema jurídico

Para auxiliar com a conexão com o banco de dados, foi utilizado um *framework* para a persistência de dados chamada Hibernate. Com ele é possível mapear as classes de entidade, ou seja, as quais irão ter um identificador no banco de dados. Para isto foi criado uma classe fora do modelo UML chamada *HibernateUtils,* a qual é responsável por iniciar a aplicação com o banco.

O banco de dados que foi utilizado para este protótipo foi o MYSQL 5.5. Ele é o mais popular banco de dados, e esta disponível sobre a licença GPL, e é suportado por uma comunidade enorme e ativa de desenvolvedores de código aberto (MYSQL).

Para visualizar os dados do banco, foi utilizado o MYSQL *Workbench*, que é uma ferramenta a qual possui um ambiente que provê acesso ao banco de dados de modo gráfico, podendo criar *scripts* e consultas.

Para visualizar de forma clara como foi criada a arquitetura da camada de modelo a Figura 13 ilustra o diagrama de classe contendo o relacionamento entre as classes. No MDA, este processo é o chamado PIM, definindo assim um modelo independente de plataforma.

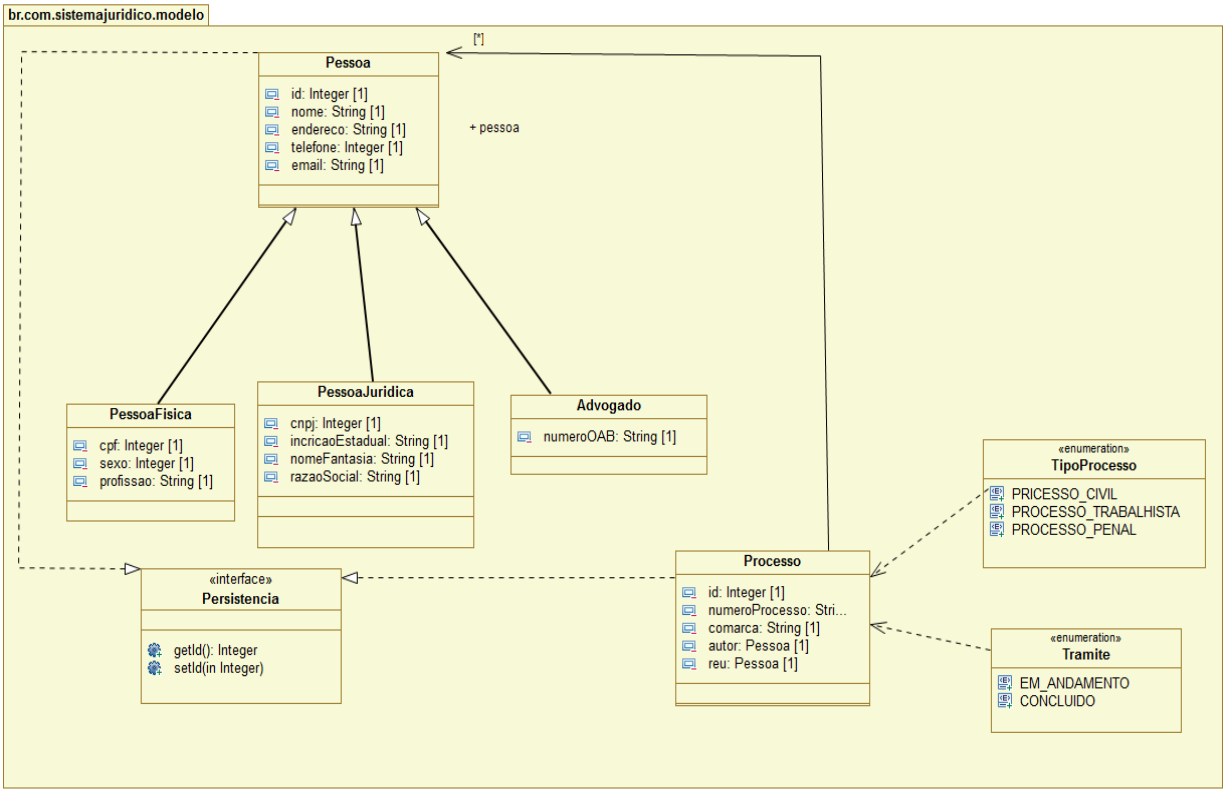


Figura 13: Diagrama de classe do modelo
Fonte: Autoria Própria

Na Figura 13, a classe Pessoa é um supertipo de classe. As classes PessoaFisica, PessoaJuridica e Advogado são subclasses de Pessoas. No modelo a classe Processo tem um relacionamento com a classe Pessoa, pois uma pessoa pode ter vários processos, mas um processo somente pode pertencer a uma única pessoa. Na classe Processo, foi criado um relacionamento para duas classes do tipo Enum, uma chamada TipoProcesso, a qual descreve que tipo o processo é, processo cível, processo trabalhista ou processo penal. Outra classe é o Enum Trâmite, o qual permite informar o *status* do processo, que pode ser em andamento ou concluído. Para finalizar foi criada uma interface Persistência, a qual contêm os

métodos *set* e get de id, para facilitar nas classes DAO para que a persistência de dados seja feita somente com classes que implementam a interface Persistência

O pacote DAO contém as classes que remetem a persistência de dados e também foi criado um modelo UML para descrever este pacote. A Figura 14 ilustra o modelo criado.

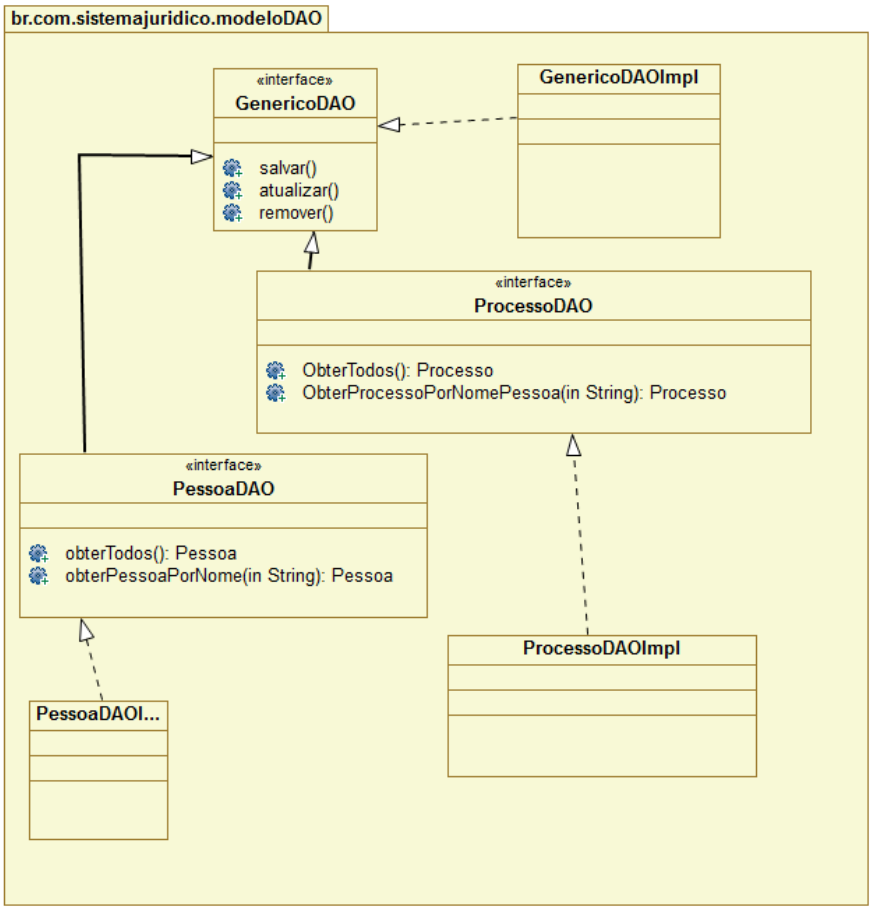


Figura 14: Diagrama de classe modelo DAO
Fonte: Autoria Própria

O relacionamento entre estes pacotes DAO e modelo não acontece, pois quem utiliza as classes do pacote DAO, são as classes do pacote controle. Por exemplo, no diagrama da Figura 13 mostra uma classe PessoaDAO, mas esta classe será chamada pela camada de controle, no caso do sistema proposto, será chamada pela classe ControlePessoa.java, a qual irá passar um objeto do tipo Pessoa para o DAO para que ele se encarregue da criação, remoção ou atualização de entidade Pessoa no banco de dados.

4.3.2 Construção da camada de controle

A camada de controle é responsável por intermediar a camada de visão e a camada de modelo. Todas as requisições que são destinadas ao uso do banco de dados, são intercaladas pela camada de controle. Para exemplificar pode-se ter o seguinte fluxo: É cadastro pela camada de visão uma pessoa em um sistema. Quando este formulário é enviado, os dados são passados pela camada de controle, para que talvez sejam validados, ou até mesmo aplicados alguma outra regra de negócio para esta nova pessoa que está sendo cadastrada. Seguindo o fluxo, a camada envia os dados para a camada de modelo, o qual irá gravar os dados no banco de dados.

Basicamente este fluxo ocorre no sistema proposto. A Figura 15 ilustra como foi modelada a camada de controle. Como citado anteriormente, o sistema é somente para demonstrar o uso do MDA, para controle apenas as duas classes foram definidas:

- **br.com.sistemajuridico.controle.PessoaControle**: Encarregada de intermediar os dados que chegam da camada de visão relacionada a pessoa, com a camada de modelo de pessoa.
- **br.com.sistemajuridico.controle.ProcessoControle**: Encarregada de intermediar os dados que chegam da camada de visão de processo para a camada de modelo de processo.

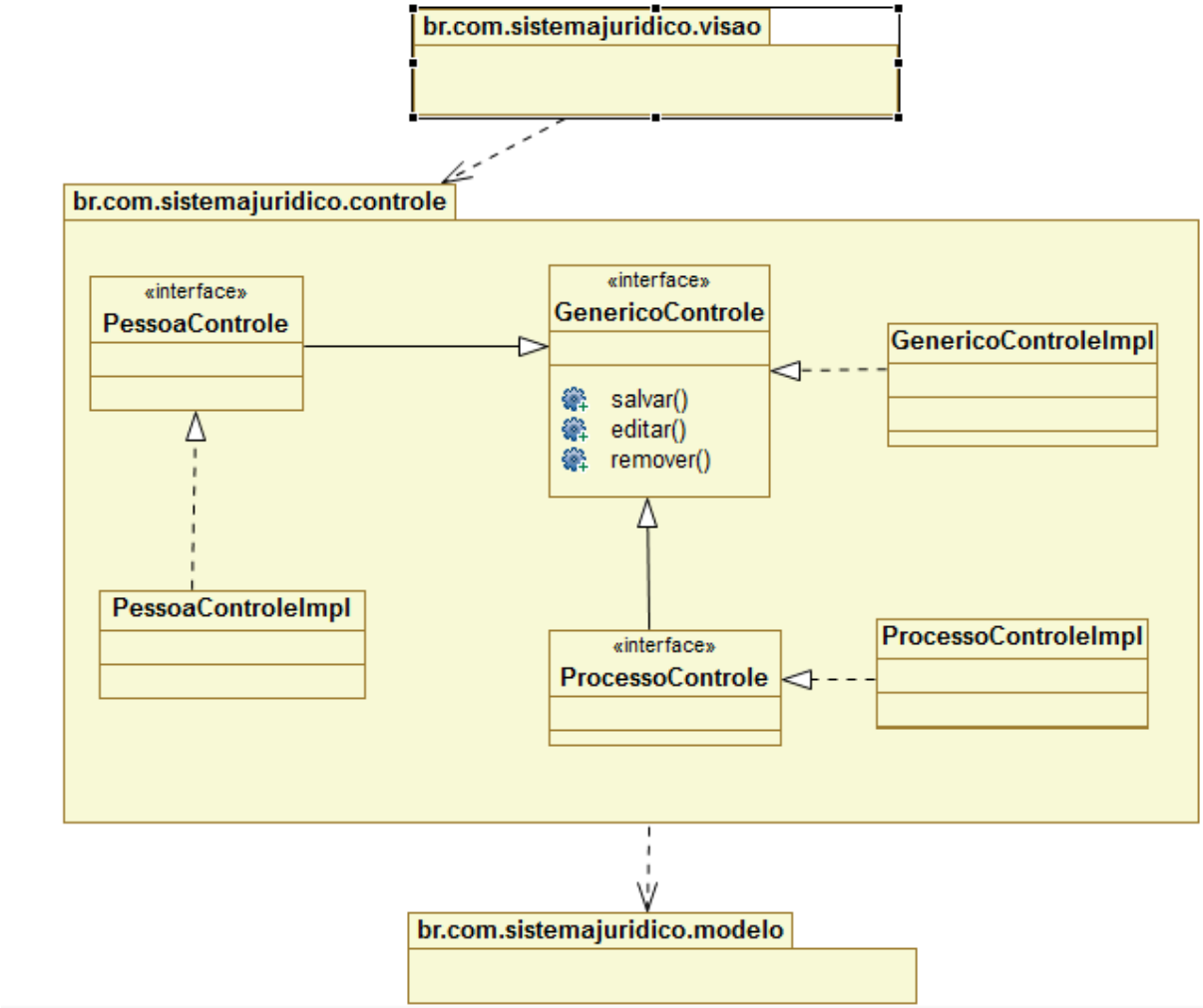


Figura 15: Diagrama de Classe do controle
Fonte: Autoria Própria

4.3.3 Construção da camada de visão

A camada de visão é a responsável pela interação do cliente com o sistema. É nela onde são adicionados, pesquisados, editados os dados. Para cada processo do sistema há uma janela a qual contém todos os campos para que usuário possa navegar entre eles e também entre os processos.

No sistema proposto foi criado um modelo UML somente para demonstrar a conexão entre cada janela (classe), e definido alguns métodos, pois o *framework* papyrus, o qual auxilia na criação da UML não tem componentes de tela. O ACCELEO criou as classes, e os componentes foram adicionados posteriormente juntamente com a sua regra de negócio. A figura 16 ilustra o modelo UML referente a camada de visão:

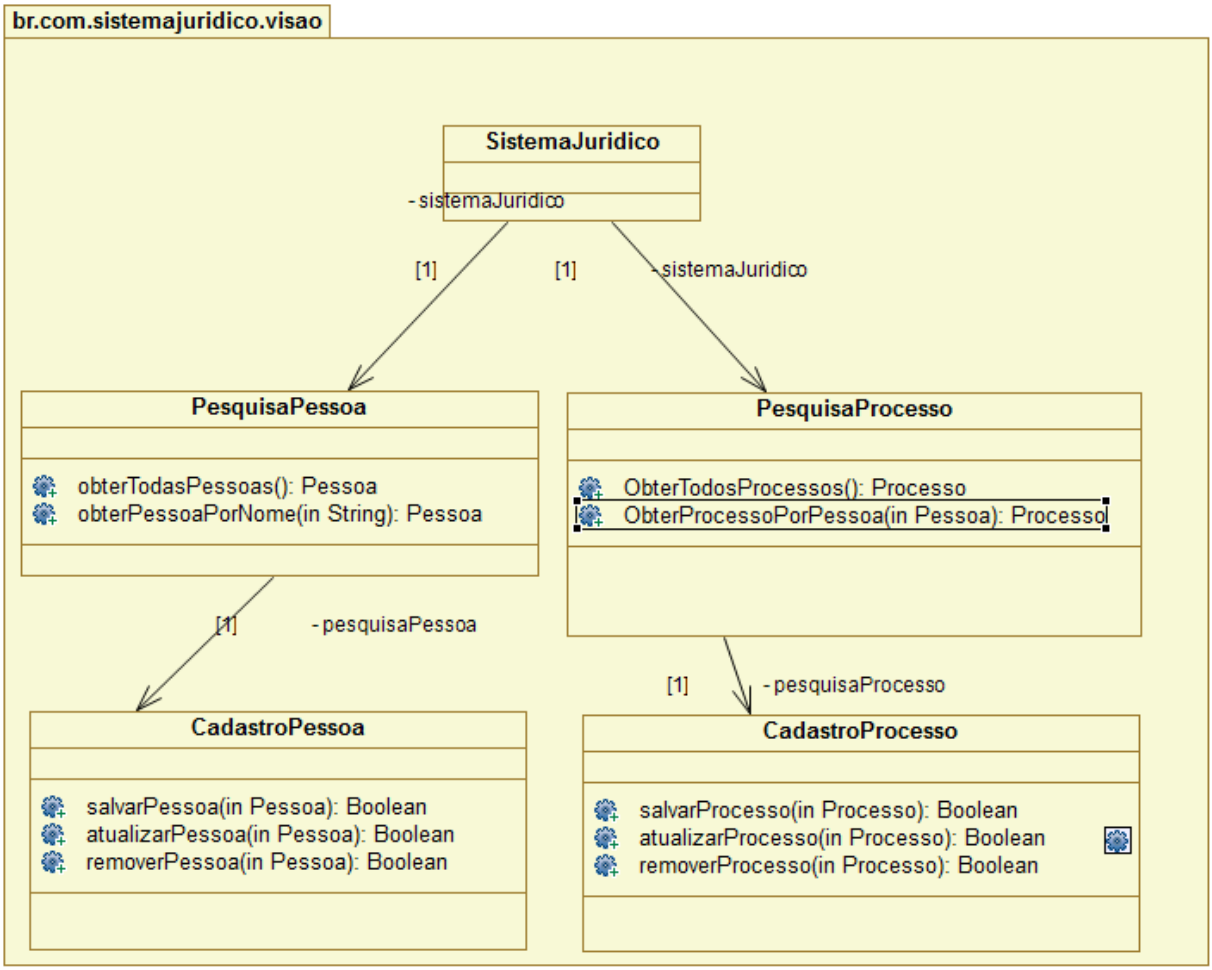


Figura 16: Diagrama de clsse do modelo de visão
Fonte: Autoria Própria

Com a modelagem criada acima no papyrus, consegue-se com o ACCELEO a geração das classes correspondente no modelo. Para demonstrar uma classe criada, a Figura 17 mostra a classe PesquisaPessoa.java.

```java
 * Generado com Acceleo
package br.com.sistemajuridico.visao;

// Start of user code for imports
import java.util.*;
// End of user code

/**
 * @author Noeli Rogalski
 */
public class PesquisaPessoa {
    private CadastroPessoa cadastroPesso;

    public CadastroPessoa getCadastroPesso() {
        return this.cadastroPesso;
    }

    public void setCadastroPesso(CadastroPessoa p_cadastroPesso) {
        this.cadastroPesso = p_cadastroPesso;
    }

    /**
     *
     * @param nome
     * @return
     */
    public Pessoa obterPessoaPorNome(String nome,) {
        // Start of user code for operation obterPessoaPorNome
        // TODO should be implemented
        return null;
        // End of user code
    }
    /**
     *
     * @return
     */
    public List<Pessoa> obterTodasPessoas() {
        // Start of user code for operation obterTodasPessoas
        // TODO should be implemented
        return null;
        // End of user code
    }
}
```

Figura 17: Exemplo de código gerado pelo ACCELEO
Fonte: Autoria Própria

A classe foi criada a partir do modelo da Figura 16 (Diagrama de classe do modelo de visão). A classe PesquisaPessoa, tem uma referência para a classe CadastroPessoa, pois somente por meio de PesquisaPessoa, é que se tem acesso ao CadastroPessoa.

Os métodos também foram adicionados a partir do modelo. Os dois métodos têm a mesma assinatura do modelo e retorna o que foi definido no modelo. Cabendo manualmente a implementação da regra para os métodos.

4.3.4 O Protótipo

Após a criação das classes e implementação das regras, foi criado o *layout* para cada classe e adicionado componentes visuais para que o protótipo seja navegável.

O sistema contém cinco janelas para navegação e foram definidos os nomes correspondentes para função para cada uma delas:

- **Tela Principal:** Corresponde ao Início da aplicação, Figura 18.

Figura 18: Tela Principal
Fonte: Autoria Própria

- **CRUD de Processo:** Janela responsável por cadastrar, ler, atualizar e remover processos no sistema. Figura 19

Figura 19:Tela Cadastro de Processo
Fonte: Autoria Própria

- **CRUD de Pessoa:** Janela responsável por cadastrar, ler, atualizar e remover pessoas no sistema. Figura 20.

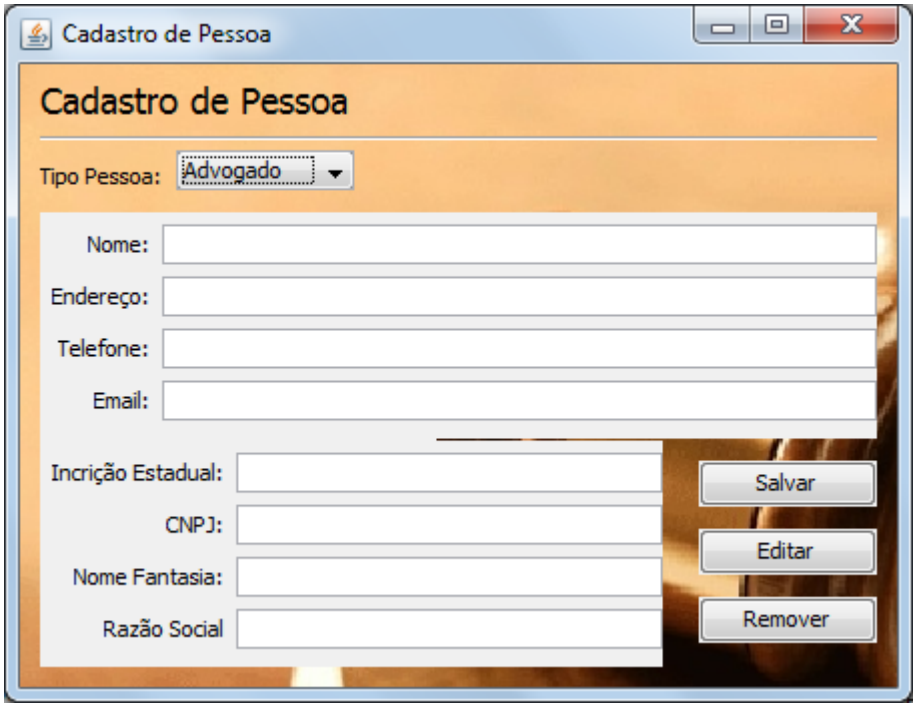


Figura 20: Tela Cadastro de Pessoa
Fonte: Autoria Própria

- **Pesquisa de Pessoas:** Janela responsável por mostrar para o usuário todas as pessoas cadastradas no sistema, podendo filtrar a pesquisa pelo nome da pessoa, onde irá mostrar todos os processos que esta pessoa tem.

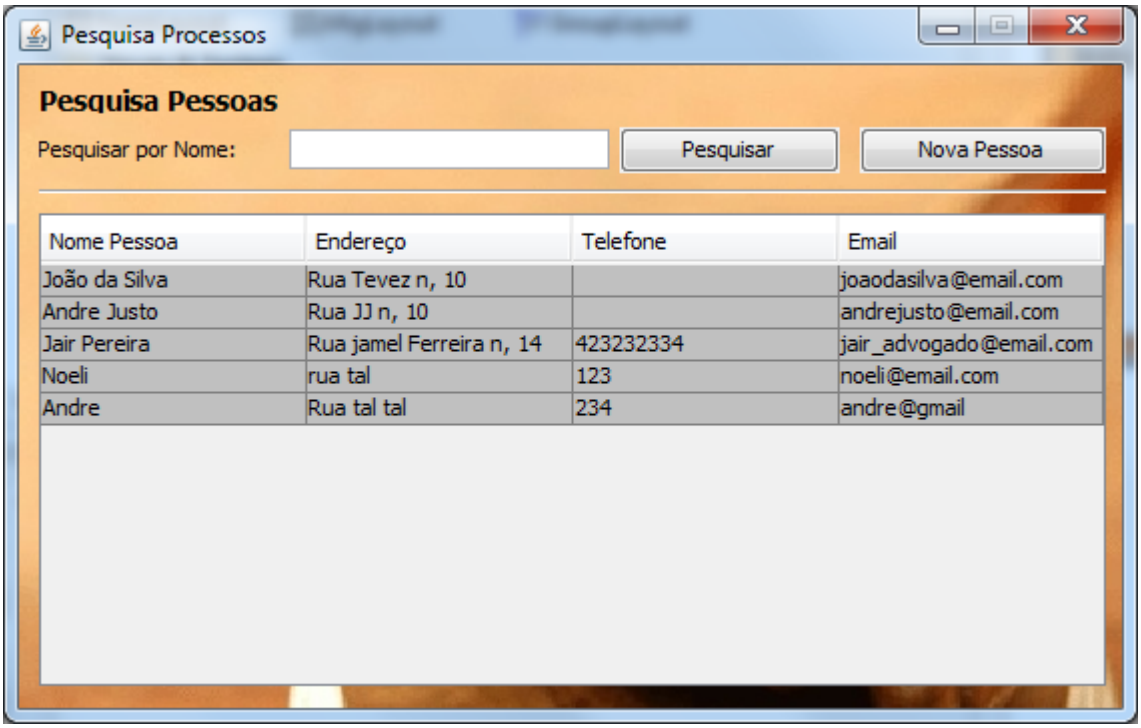


Figura 21: Tela de pesquisa de pessoas
Fonte: Autoria Própria

- **Pesquisa de Processo:** Tela responsável por mostrar para o usuário todos os processos cadastrados no sistema, podendo filtrar a pesquisa pelo nome da pessoa, onde irá trazer todos os processos envolvendo esta pessoa tanto como Réu como Autor do processo.

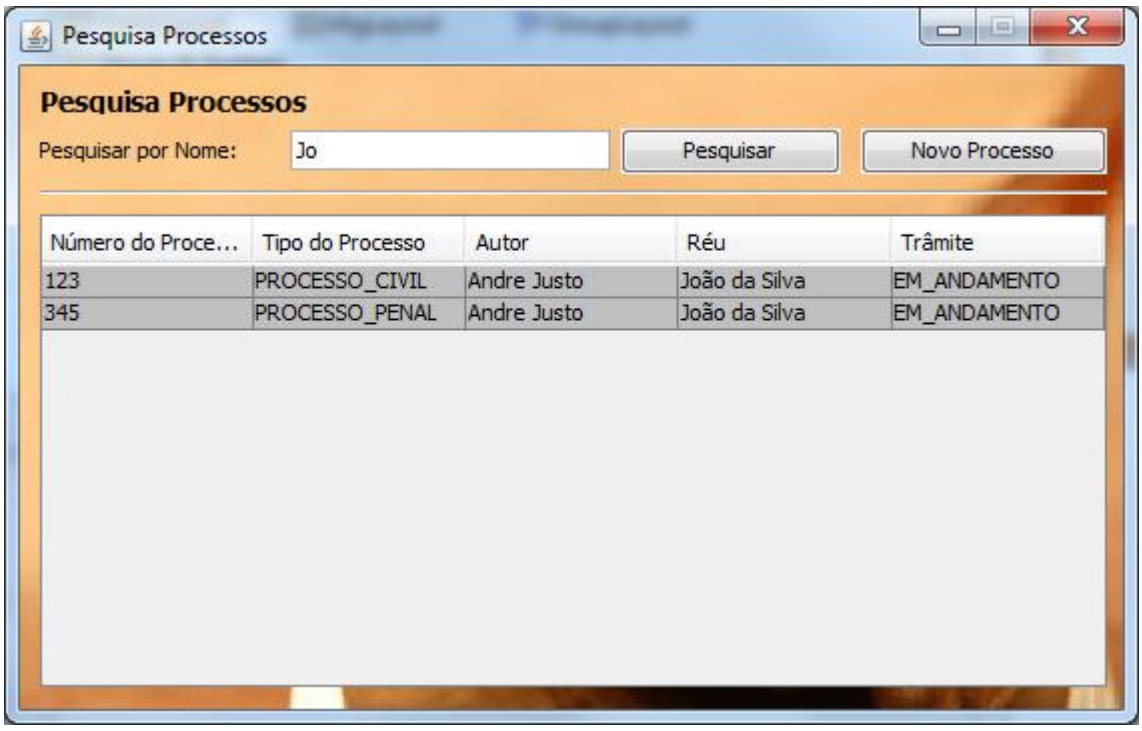


Figura 22: Tela pesquisa de processos
Fonte: Autoria Própria

4.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo demonstrou a transformação do PIM para o PSM na arquitetura MDA, utilizando para isso o *plugin* ACCELEO, desenvolvido para a IDE eclipse. Ele possibilita gerar a partir de modelo UML, classes fazendo com que a produtividade de implementação de código fonte seja mais eficiente.

Demonstrados também conceitos que auxiliam para entendimento do ACCELEO como seu funcionamento, *templates* e o metadado XMI, o qual é lido pelo ACCELEO para realizar a transformação do modelo para código fonte.

Por fim, foi criado um protótipo contendo um modelo UML básico de um sistema jurídico criado neste projeto. O modelo segue as regras do MVC, contendo para cada camada, seu pacote UML, que com o auxílio do ACCELEO transforma os modelos UML em classes. As telas que foram criadas para o sistema bem com algumas classes geradas pelo ACCELEO.

O *plugin* ACCELEO, definitivamente auxilia na produtividade de geração de código. Ele é uma ferramenta cujas funcionalidades são fáceis de utilizar. A linguagem criada pela OMG para criação de *templates,* no entanto, não é difícil de entender e criar novos *templates* para determinados tipos de classes.

Outra ferramenta que foi apresentada neste capítulo é o Papyrus, um excelente editor UML, podendo criar os modelos auxiliando na fase do modelo PIM da arquitetura MDA.

Com o protótipo final, pode-se verificar como a quantidade de código e tempo para desenvolvimento foi afetado com as ferramentas utilizadas. Com todas as classes prontas geradas pelo ACCELEO, se tratando de um protótipo, as regras básicas de negócio foram adicionar de forma rápida e eficiente, não sendo necessário se preocupar com outros passos do modelo, pois ele já tinha sido definido na fase do modelo PIM.

5 CONSIDERAÇÕES FINAIS

Este capítulo trata os resultados finais obtidos no trabalho nas seguintes seções: 5.1 Conclusão e o 5.2 Trabalhos Futuros.

5.1 CONCLUSÃO

A arquitetura MDA, é um auxílio na construção de *softwares*. Quando se trata da especificação de requisitos para a criação de um *software*, o processo ainda não é estruturado, podendo comprometer a qualidade. As metodologias tradicionais não abordam integralmente todos os processos que envolvem a construção de um software, e, portanto, não os organiza de uma maneira ideal.

O estudo sobre a arquitetura MDA permitiu identificar de maneira mais clara as fases de cada processo, desde a visão abstrata do modelo, quanto à modelagem de dados e a simplicidade para gerar arquivos de códigos fontes a partir de modelo UML.

A utilização da arquitetura MDA auxiliou o gerenciamento do processo de construção do *software* de forma mais eficaz, trazendo para o cliente final, um produto de qualidade com seus requisitos atendidos.

O processo de geração de código automatizada provida pela ferramenta ACCELEO proporciona um grau de produtividade maior com relação à implementação do sistema, auxiliando para que o prazo de entrega seja atendido garantindo a satisfação do cliente.

O processo de entendimento do sistema gerado na primeira fase da arquitetura (PIM) proporcionou a geração do modelo UML (PSM) com maior facilidade e uma ampla visão dos requisitos do sistema.

O protótipo criado forneceu uma aprendizagem com relação à geração automatizada de códigos fonte utilizando o *plugin* ACCELEO na última fase da arquitetura MDA. A produtividade na geração de código aumentou, pois o tempo gasto para criar cada classe e definir atributos e métodos diminuiu devido à modelagem UML. A vantagem ocorre quando há necessidade de uma mudança nas classes, pois somente alterando o modelo UML, podem-se gerar novamente as classes sem o comprometimento de alteração no próprio código fonte.

Apesar de existirem iniciativas bem sucedidas de geração de código, a MDA representa uma abordagem mais completa e consistente, até mesmo porque sua atuação não é restrita ao código fonte propriamente dito. De fato, com a MDA, praticamente todos os artefatos resultantes do processo de desenvolvimento se beneficiam em algum grau de sua capacidade de automatização.

A arquitetura aumenta a produtividade do desenvolvedor e permite que seja criado um projeto de desenvolvimento organizado e bem estruturado,

Adotar a UML e a MDA de forma consciente e madura significa dar um passo importante em direção à criação de aplicações mais completas, com mais qualidade.

5.2 TRABALHOS FUTUROS

A facilidade na geração de códigos automatizados demonstrado no protótipo proporciona uma visão interessante sobre ferramentas que geram código automaticamente.

No mercado existem outras ferramentas as quais geram códigos automatizados a partir de um modelo. O ACCELEO citado neste projeto é uma delas.

Como trabalhos futuros poderá ser analisado outras ferramentas as quais já citadas no capítulo 4 que auxiliam na geração de código fonte automatizados como também para outras linguagens de programação podendo assim obter uma análise mais completa sobre automatização de código fonte.

REFERÊNCIAS

ACCELEO. Disponível em: <http://www.ACCELEO.org>. Acesso em: 11 fev 2012.

ALMEIDA FILHO, J. C. A. **Processo e Teoria Geral do Processo Eletrônico**: A Informatização Judicial no Brasil. Rio de Janeiro: Editora Forense, 2007.

ECLIPSE. Disponível em:<http://www.eclipse.org/downloads/> Acesso em: 11 fev 2012.

FAVRE, Liliana. **Model Driven Architecture for Reverse Engineering Technologies: a Strategic Directions and System Evolution**. United States of America. Engineering Science Reference, 2010.

JONYSBERG. **Independência de plataforma para modelagem de software** HTTP://jonysberg.wordpress.com/category/mda. Acesso em: 11 fev 2012.

LIMA, Ricardo. **Nota de aula no curso de Planej. de Projetos. Processo de desenv. baseado em MDA**. Acesso em: 11 fev 2012.

LUCON, P. H. S. **Duração razoável e informatização do processo judicial**. Paranóptica, ano 1, n. 8, maio-junho 2007.

MYSQL. Disponível em: < http://mysql.com/products/community/ >. Acesso em: 05 mai 2012

MYSQL. Disponível em: <http://wb.mysql.com>. Acesso em: 11 fev 2012.

MDA - Desenvolvimento de Softwares baseado em Modelos, Hamden, Brasília, DF, 2010.

MDA. MDA Guide Version 1.0.1. In J. Miller & J. Mukerji (Eds.). Document omg/2003-06-01. 2003. Disponível em:< www.omg.org/mda > .Acesso em: 30 abr 2012

MOURÃO, Walter Itamar. **MDA: Fazendo a UML valer a pena.** Disponível em: <http://waltermourao.com.br/export/sites/waltermourao/download/MDA-Fazendo_a_UML_valer_a_pena.pdf>. Acesso em: 10 dez 2011.

OMG. OMG Model Driven Architecture. Disponível em: <http://www.omg.org/mda/>. Acesso em: 11 mar 2012.

PAPYRUS. Disponível em: < http://www.papyrusuml.org >. Acesso em: 05 mai 2012.

ROHRMANN. C. A. **A informatização do processo judicial segundo a lei n. 11.419, de 19 de dezembro de 2006**. Revista da Faculdade de Direito Milton Campos. Volume 16. Belo Horizonte: Del Rey, 2008.

SILVA, Thiago S**. Model Driven Architecture – Conceitos Fundamentais.** 2008. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=1953>Acesso em: 20 mai 2012.

Trinta, Fernando**. Model Driven Architecture – Centro de Informática/UFPE.** 2010. Disponível em: <http:// http://www.cin.ufpe.br/~cagf/sdgrad/aulas/MDA.pdf >Acesso em: 11 fev 2012.

TUTORIAL UFBA**. Gerando uma aplicação no Cordel**.Disponível em: <https://intranet.dcc.ufba.br/pastas/gaudi/cordel/www/tutorial.html> Acesso em: 11 fev 2012.

WEBER, Alison R.; JANUARIO, Cesar A. **Aplicação da arquitetura MDA no desenvolvimento de uma aplicação *HELP DESK*.** 2010. 64f Trabalho de conclusão de curso – Programa de graduação em Tecnologia, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2010.