UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS**

CIÊNCIA DA COMPUTAÇÃO

FELIPE IGAWA MOSKADO

UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS JPA BASEADO EM MDD E POA

Bandeirantes

2016

FELIPE IGAWA MOSKADO

UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS JPA BASEADO EM MDD E POA

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. André Luís Andrade Menolli.

Bandeirantes

2016

FELIPE IGAWA MOSKADO

UMA ABORDAGEM PARA GERAÇÃO DE CÓDIGO DE PERSISTÊNCIA DE DADOS JPA BASEADO EM MDD E POA

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

COMISSÃO EXAMINADORA

_____
Prof. Dr. André Luís Andrade Menolli
UENP –*Campus* Luiz Meneghel

_____
Prof. Me. Glauco Carlos Silva
UENP – *Campus* Luiz Meneghel

_____
Prof. Dr. Maurício Massaru Arimoto
UENP – *Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2016.

Dedico este trabalho aos meus pais, Alzira e Milton, sem os quais não teria tido forças de realizá-lo.

AGRADECIMENTOS

Agradeço primeiramente a Deus que permitiu que tudo isso acontecesse ao longo de minha vida e não somente nestes anos como universitário.

Aos meus pais Alzira e Milton que sempre me apoiaram nas minhas decisões e estavam comigo em todos os momentos que precisava de apoio e meus irmãos Gustavo e Henrique que sempre depositaram suas confianças em mim e me deram forças para continuar acreditando nos meus ideais.

Ao meu avô Robson e meus tios Márcia e Roberto, que no início da graduação me hospedaram em sua casa e me ajudando com o que eu necessitava para começar a minha jornada.

Ao meu orientador Menolli, que inicialmente me inseriu como bolsista de Iniciação Tecnológica e posteriormente na Iniciação Científica, me proporcionando novos conhecimentos além do que a graduação ensina. Agradecê-lo também pela oportunidade e apoio na elaboração deste trabalho.

A todos os professores que participaram da minha formação acadêmica durante esses quatro anos de curso, por me proporcionarem conhecimento e conselhos.

A todos os meus amigos que me acompanharam durante essa jornada, em especial Erick, Fernando, Luan, companheiros de estudos e de festas.

O impossível é uma palavra encontrada somente no dicionário dos tolos.
(Napoleão Bonaparte)

RESUMO

Com o passar dos anos, novas abordagens vêm sendo propostas para o desenvolvimento de software com o intuito de melhorar a qualidade do produto final e aumentar a produtividade no desenvolvimento. Dentre as abordagens propostas, destaca-se a programação orientada a aspectos que consiste de uma evolução oriunda da programação orientada a objetos e visa melhorar principalmente a separação de interesses, o que afeta diretamente a coesão e o acoplamento do software. Outra abordagem significativa é o desenvolvimento orientado a modelos, que possui como finalidade gerar códigos de forma automática por meio de modelos. Esta abordagem auxilia a manter a coerência entre modelo e código, além de agilizar o processo de desenvolvimento de software. O uso destas abordagens no domínio de persistência de dados pode trazer benefícios para o projeto de desenvolvimento de software. Diante disto, este trabalho propõe uma abordagem e um *framework* de persistência de dados JPA baseado no desenvolvimento dirigido a modelos e na programação orientada a aspectos. Utilizando este *framework*, espera-se que programadores inexperientes consigam criar aplicações de persistência de dados de maneira produtiva e com qualidade. Para isto, foi aplicado um estudo experimental, que procurou similar a utilização do *framework* para implementar uma parte do sistema de uma biblioteca, utilizando análise quantitativa dos dados. Os resultados obtidos através do experimento demonstram que a abordagem proposta é viável para a redução do tempo de desenvolvimento do software e para a melhoria da qualidade do código em comparação com a implementação tradicional, que usa a programação orientada a objetos.

Palavras-chave: Desenvolvimento dirigido a modelos, programação orientada a aspecto, persistência de dados, JPA.

ABSTRACT

Over the years, new approaches have been proposed for the development of software in order to improve the quality of the final product and increase productivity in development. Among the proposed approaches, emphasis is placed on aspect-oriented programming that consists of an evolution derived from object-oriented programming and aims to improve mainly the separation of interests, which directly affects the cohesion and coupling of software. Another significant approach is model-driven development, whose purpose is to generate codes automatically through models. This approach helps maintain consistency between model and code, as well as streamline the software development process. The use of these approaches in the field of data persistence can bring benefits to the software development project. Therefore, this work proposes a JPA data persistence approach and framework based on model driven development and aspect oriented programming. Using this framework, it is expected that inexperienced programmers will be able to create data persistence applications in a productive and quality way. For this, an experimental study was applied, which sought similar use of the framework to implement a part of the system of a library, using quantitative data analysis. The results obtained through the experiment demonstrate that the proposed approach is feasible for reducing software development time and improving code quality compared to traditional implementation, which uses object-oriented programming.

KeyWords: Model-Driven Development, Aspect Oriented Programming, Data Persistence, JPA

LISTA DE FIGURAS

Figura 2.1 - Separação de interesses com POA (FONTE: Chavez & Garcia, 2003). 26
Figura 2.2 - Situação problema em OO (FONTE: Chavez & Garcia, 2003) 27
Figura 2.3 - Solução do problema com POA (FONTE: Chavez& Garcia, 2003) 28
Figura 2.4 - Interesses Transversais na atualização da imagem. 30
Figura 2.5 - Interesses Transversais na atualização da imagem em um Aspecto. 30
Figura 2.6 - Ferramenta *Acceleo* (FONTE: ACCELEO, 2016). 34
Figura 3.1- Estrutura da Pesquisa 36
Figura 4.1 -- Abordagem Proposta 40
Figura 4.2 – Meta-modelo de persistência de dados JPA 42
Figura 4.3 - Exemplo de um *template* do *Acceleo* que gera uma classe Java 44
Figura 4.4 – Código-fonte gerado pelo *template* 45
Figura 4.5 - Etapas para geração de código com o *Acceleo* na abordagem (FONTE: ACCELEO, 2016) 45
Figura 4.6 - Diferentes papéis presentes no framework proposto. 47
Figura 6.1 - Tempo médio de implementação de cada grupo 57
Figura 6.2 - Gráficos de distribuição normal, tempoA x tempoS 58
Figura 6.3 – Separação de Interesses 60
Figura 6.4 - Acoplamento 61
Figura 6.5 - Falta de coesão das implementações 61
Figura 6.6 - Tamanho das implementações 62
Figura 6.7 - Tempo de implementação (JPA x JDBC) 63

LISTA DE TABELAS

Tabela 6.1 - Comparação das implementações com menor Tempo de implementação(Com abordagem x Sem abordagem) 56
Tabela 6.2 – Comparação da qualidade do código das implementações (Com abordagem x Sem abordagem) 59
Tabela 6.3 - Comparação das métricas de qualidade (JPA x JDBC) 64

LISTA DE SIGLAS

AJC	Compilador do *AspectJ*
ATL	*ATLAS Transformation Language*
C#	Linguagem de programação C#
C++	Linguagem de programação C++
CA	Acoplamento Eferente
CDC	Difusão do interesse sobre os components
CDLOC	Difusão do interesse sobre as linhas de códigos
CDO	Difusão do interesse sobre as operações
CE	Acoplamento Aferente
DIT	Profundidade da árvore de herança
DSOA	Desenvolvimento de Software Orientado a Aspecto
EJBQL	*Enterprise JavaBeans Query Language*
JAVA	Linguagem de programação JAVA
JDBC	*Java Database Connectivity*
JPA	*Java Persistence API*
JPQL	*Java Persistence Query Language*
JVM	*Java Virtual Machine*
LCOO	Falta de coesão nas operações
LOC	Linhas de código.
M2T	*Model-to-Text*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
MOF	*Meta-Object Facility*
NOA	Número de atributos
OMG	*Object Management Group*
OO	Orientação a Objeto
O-R	Objeto-Relacional
ORM	*Object-Relational Mapping*
PIMs	*Platform-Independet Models*
POA	Programação Orientado a Aspecto
POJO	*Plain Old Java Object*
POO	Programação Orientada a Objeto

QVTL	*Query-View Transformatio nLanguage*
SQL	*Structure Query Language*
SysML	*Systems Modeling Language*
UML	*Unified Modeling Language*
VD	Variável (eis) Dependente (s)
VI	Variável (eis) Independente (s)
WOC	Comprimento das operações por componentes
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformation*

SUMÁRIO

1. Introdução 15
1.1 Contextualização e Formulação do Problema 15
1.2 Objetivos 17
1.2.1 Objetivos Específicos 17
1.3 Justificativa 18
1.4 Organização do Trabalho 20
2. Fundamentação teórica 22
2.1 Desenvolvimento Dirigido a Modelo 22
2.2 Programação Orientada a Aspectos 25
2.2.1 AspectJ 28
2.3 Java Persistence API 31
2.4 Acceleo 32
3. Estruturação da pesquisa 35
3.1 Caracterização da Pesquisa 35
3.2 Estruturação da Pesquisa 35
3.2.1 Planejamento Inicial 36
3.2.2 Fase Exploratória 37
3.2.3 Desenvolvimento 37
3.2.4 Conclusão 38
4. Abordagem Proposta 39
4.1 Abordagem Proposta – Uma Visão Geral 39
4.2 Meta-Modelo 41
4.3 Gerador de Código 43
4.4 Os Papéis 46
5. Experimento 48
5.1 Método Experimental 48
5.2 Planejamento 49
5.2.1 Definição das variáveis 49
5.2.2 Definição das hipóteses 50
5.2.3 Seleção de Participantes 51

5.2.4 Descrição do Experimento 51
5.2.5 Descrição da Análise 52
5.2.6 Execução do Experimento 53
6. Análise dos Resultados 55
6.1 TEMPO DE DESENVOLVIMENTO 55
6.2 QUALIDADE DA IMPLEMENTAÇÃO 58
6.2.1 SEPARAÇÃO DE INTERESSES 60
6.2.2 ACOPLAMENTO, COESÃO E TAMANHO 60
6.2.3 DISCUSSÃO 62
6.3 JPAX JDBC 63
7. Considerações Finais 67
7.1 Relevância do estudo 67
7.2 Contribuições da Pesquisa 68
7.3 Limitações 68
7.4 Trabalhos futuros 68
REFERÊNCIAS 69
Apêndice A –Ecore/XML do Meta-Modelo 75
Apêndice B– *Template Acceleo* para Geração de Código 76
Apêndice C– Artefatos gerados pelo framework no experimento 82
Apêndice D– Descrição do Experimento 91
Apêndice E– Diagrama de Classe utilizado no experimento 94
Apêndice F– Diagrama de Sequência utilizado no experimento 95
Apêndice G– Script para geração da base de dados relacional do projeto em Mysql 96
Apêndice H– Termo de consentimento livre e esclarecimento 98

1. INTRODUÇÃO

Neste capítulo são apresentadas a contextualização e formulação do problema, os objetivos do trabalho, justificativa para a elaboração do trabalho, e por fim a organização do documento.

1.1 CONTEXTUALIZAÇÃO E FORMULAÇÃO DO PROBLEMA

Em muitas aplicações, sejam *Desktop*, móvel, *Web* ou qualquer outro tipo, são necessárias que suas informações sejam armazenadas de forma persistente. Atualmente, existe uma vasta quantidade de métodos e ferramentas, como por exemplo, *Persistence Data Collection* (MASSONI *et. al*., 2001), *Decoupling Patterns*, um conjunto de padrões de persistência de dados (NOCK, 2003), *Persistence Layer* (YODER *et. al,* 1998), *Data Access Object* (SUN MICROSYSTEMS, 2002), que permitem aos desenvolvedores implementarem essas funcionalidades de diversas maneiras.

No entanto, esta diversidade de opções não é necessariamente apropriada, pois muitas vezes, transfere a responsabilidade pela escolha do mecanismo de persistência de dados ao desenvolvedor e em muitos casos este não está apto para fazer a melhor escolha.

A escolha pelo método de persistência de dados deve ser realizada ao nível do projeto, pois o projetista de software deve ser responsável e ter conhecimento para diferenciar qual é o melhor método para a persistência de dados naquela ocasião. Porém, muitas equipes não possuem uma divisão de papel clara, com isto a escolha de qual persistência de dados utilizar acaba sendo de duas formas: Adota o padrão utilizado pela empresa ou uma escolha pessoal do responsável.

Notavelmente, escolhas feitas destas formas podem ser desvantajosas, pois pode ser que a escolha padrão da empresa não seja a mais apropriada para o projeto em específico, ou caso seja uma escolha individual, pode acontecer de que outros programadores não tenham domínio sobre a tecnologia, e consequentemente dificultará a manutenção e o reuso do software.

Contudo, existem diversos estudos relacionados a este assunto, e tentam elencar quais as técnicas e ferramentas são mais apropriadas para cada situação. Dentre estes estudos, destaca-se o trabalho de Pothu (2012) que faz uma análise

comparativa dos métodos de mapeamentos objeto relacional para Java. Outro trabalho é apresentado por Barcia (2008), que faz uma revisão sobre as principais tecnologias de persistência de dados.

Os estudos citados anteriormente descrevem em quais casos diferentes técnicas ou métodos são mais adequados. No entanto, escolher qual método ou técnica é mais adequado para um projeto é apenas um passo para a implementação da persistência de dados, pois ainda será necessário que os desenvolvedores aprendam a dominar estas diferentes técnicas e métodos para implementá-los.

Uma alternativa para tratar esta adversidade é utilizar o Desenvolvimento Dirigido a Modelos (*Model-Driven Development* - MDD). O MDD permite desenvolver códigos a partir de modelos de alto nível, sua função basicamente é transformar um modelo de software em código executável, seja parcialmente ou integralmente, variando de acordo com as necessidades.

A utilização do MDD pode auxiliar os programadores e todos os envolvidos no projeto de software, pois os códigos são gerados automaticamente sem que exista a necessidade de serem produzidos totalmente de forma manual. Além disso, o MDD auxilia na padronização de código e no reuso, uma vez que todas as implementações do código devem ser baseadas no modelo de alto nível. Consequentemente facilita a manutenção do software, os erros são minimizados e agiliza a entrega do produto final.

O MDD é uma evolução na Engenharia de Software, e contribui para sanar um grande problema no desenvolvimento de software, a incompatibilidade entre o modelo criado e o código desenvolvido (JIANG; ZHANG; MIYAKE, 2007).

Deste modo, uma ferramenta baseado em MDD que auxilie o desenvolvimento de código de persistência de dados, pode contribuir para evitar problemas recorrentes neste tipo de aplicação, além de mitigar contratempos comuns do desenvolvimento de software.

No entanto, outras abordagens da Engenharia de Software podem ser pertinentes no desenvolvimento de aplicações de persistência de dados, tais como a Programação Orientada a Aspecto (POA). A POA surgiu para solucionar a limitação do paradigma de orientação a objetos em capturar algumas decisões de projeto que um sistema deve implementar, causado assim a distribuição pelo código e espalhamento de código em diversas classes com diferentes propósitos (SOARES *et. al.*, 2002). Este entrelaçamento e espalhamento tornam difícil o desenvolvimento

e a manutenção destes sistemas. Desta forma, a POA aumenta a modularidade, pois separa o código que implementa funções específicas, afetando diferentes partes do sistema, chamadas de preocupações ortogonais (*Crosscutting concern*) (SOARES *et. al.*, 2002).

A POA é constituída de interesses que são requisitos, uma propriedade ou uma funcionalidade do sistema. Estes interesses podem ser implementados tanto por classes, que agrupam e modularizam uma funcionalidade, e por aspectos, que implementam códigos que estariam espalhados em um sistema Orientado a Objeto (OO) e afetariam o comportamento do sistema, os quais são denominados interesses transversais.

No domínio do desenvolvimento de aplicações de persistência de dados, o uso da POA é uma alternativa viável e interessante. Dessa forma, propõe-se utilizar a POA em conjunto com o MDD para criar uma aplicação de persistência de dados que seja condizente com as melhores práticas de Engenharia de Software, e ao mesmo tempo seja uma solução para cada situação particular.

1.2 OBJETIVOS

Este trabalho visa propor uma abordagem para ferramenta geradora de código para a linguagem *Java* integrado ao ambiente de desenvolvimento Eclipse que auxilie desenvolvedores a criar aplicações de persistência de dados JPA, por meio do desenvolvimento orientado a modelos e programação orientada a aspecto.

1.2.1 OBJETIVOS ESPECÍFICOS

- Desenvolver uma abordagem adequada para geração de código de persistência de dados JPA;
- Implementar a abordagem proposta;
- Elaborar experimento para testar a abordagem;
- Análise quantitativa dos dados por meio dos experimentos.

1.3 JUSTIFICATIVA

Desenvolvimento orientado a modelos ou MDD, é uma abordagem em Engenharia de Software que visa elevar o nível de abstração de aplicações e assim, simplificar e formalizar as tarefas e os estágios do ciclo de vida de um software, usando modelos e tecnologias de modelagem (HAILPERN, 2006). O MDD tem evoluído significantemente devido à sua flexibilidade e aplicabilidade (VIDYAPEETHAM, 2009). Convencionalmente, de acordo com Vidyapeetham (2009), desenvolvedores empregam duas abordagens diferentes em desenvolvimento de software:

- Projetar a solução de forma visual;
- Codificar diretamente a solução baseando-se em requisitos funcionais.

Utilizando o MDD, desenvolvedores são capazes de, ao mesmo tempo, projetar suas soluções e construir artefatos parciais que farão parte do produto final (VIDYAPEETHAM, 2009). Do mesmo modo, Selic (2008) afirma que o foco principal do MDD são os modelos ao invés de programas de computador. Além disso, Selic (2008) aponta a expressão de modelos usando conceitos que não são ligados a linguagens de baixo nível, mas muito embora próximo ao domínio real do problema, como principal vantagem da adoção do MDD.

O MDD já vem sendo utilizado nos mais diversos domínios para auxiliar programadores a criarem soluções de forma mais rápida, simples e confiável. Como exemplo, Heitkötter, Majchrzak e Kuchen (2013) apresentam o MD2, uma linguagem de domínio específico (DSLs) para desenvolvimento de aplicações móveis multi-plataformas, gerando aplicações nativas puras para os sistemas Android e iOS. Outro exemplo é o trabalho de Steiner *et al.* (2013), que apresenta uma DSL para desenvolvimento de aplicações integradas à nuvem.

Um tipo de aplicação o qual é possível propor soluções por meio de MDD é a persistência de dados. Implementações deste domínio são comuns em várias aplicações, porém em diversas ocasiões são implementadas de forma inapropriada, pois existem vários padrões, e muitas vezes o desenvolvedor não sabe como e qual utilizar. Na literatura podem-se encontrar vários padrões de persistência de dados, como exemplo, o trabalho de Nock (2003) descreve um conjunto de padrões de acesso a dados que apresentam estratégias para desacoplar os componentes de acesso a dados de outras partes da aplicação. A existência destes padrões indica a

possibilidade de criar meta-modelos de referências para facilitar uma implementação satisfatória dos mesmos.

Além dos inúmeros padrões existentes, os programadores podem utilizar em conjunto com estes padrões, diferentes tecnologias para desenvolver o código de persistência de dados, como o *Java Persistence API* (JPA), *Hibernate* e *Structured Query Language* (SQL).

Dentre os padrões de persistência existentes, um dos mais utilizados é o *Data Access Object* (DAO), que permite aos objetos de negócio utilizar a fonte de dados sem ter conhecimento dos detalhes específicos de sua implementação, fornecendo um acesso transparente a suas funcionalidades, uma vez que os detalhes de implementação de fonte de dados são encapsulados dentro do objeto de acesso a dados (SUN MICROSYSTEMS, 2002).

Entretanto, apesar de ser um padrão consolidado, o DAO apresenta algumas limitações, que podem ser mitigadas utilizando POA. No trabalho de Oliveira, Menolli e Coelho (2008) foi aplicado a POA no padrão DAO, o que proporcionou o encapsulamento das funções de acesso a dados da aplicação desenvolvida, uma vez que os objetos de negócio e outros componentes da aplicação não necessitam acessar funções de acesso a dados da aplicação.Todo o controle de chamadas a operações de acesso e persistência de dados é efetuado pelos aspectos. Sendo assim, os demais objetos da aplicação não necessitam mais conhecer os detalhes de implementação das classes e métodos responsáveis pelas funções de acesso a dados da aplicação, otimizando modularidade e a legibilidade da aplicação.

A aplicação de POA no padrão DAO é apenas um exemplo de como padrões de acesso a dados podem ter suas implementações aprimoradas, tratando o interesse transversal de persistência de dados. Estes interesses transversais de acordo com Soares *et. al.* (2002), consistem em funcionalidades responsáveis pelo armazenamento persistente de informações manipuladas pelo sistema. Essas funcionalidades correspondem a controle de conexões, gerenciamento de transações, carregamento parcial e cache de objeto para melhorar o desempenho e a sincronização de estados de objetos com as entidades do banco de dados correspondentes para assegurar a consistência. Além disso, verificou-se em Soares *et. al.* (2002), que o interesse de persistência pode ser dividido em vários sub-

interesses, como classes de acesso a dados, controle de conexões, gerenciamento de transações e customizações de classes de acesso a dados.

No entanto, o uso, e principalmente a aplicação de POA para refinar estes padrões não é uma tarefa simples. Sendo assim, uma ferramenta baseada em MDD que auxilie o desenvolvedor a selecionar e implementar o melhor padrão de persistência de dados para sua aplicação utilizando a tecnologia JPA, proporcionará um melhor desempenho e produtividade do programador. A tecnologia está em constante evolução e presente em muitas aplicações atuais. E utilizando conceitos adequados da Engenharia de Software, será aumentada a eficiência, a padronização, o reuso e a manutenção do software.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado da seguinte maneira:

No Capítulo 1 apresenta-se a introdução do trabalho que está subdividido em: contextualização e a formulação do problema, os objetivos e objetivos específicos, a justificativa e a estruturação da pesquisa.

No Capítulo 2 apresenta-se a fundamentação teórica que contém os seguintes conceitos: Desenvolvimento Dirigido a Modelos, Programação Orientada a Aspecto, *Java Persistence API*, *AspectJ* e Acceleo.

No Capítulo 3 apresenta-se a estruturação da pesquisa, na qual está subdivida respectivamente em: caracterização da pesquisa e estruturação da pesquisa que contém planejamento inicial, fase exploratória, desenvolvimento e conclusão.

No Capítulo 4 apresenta-se a abordagem proposta, apresentando uma abordagem para o framework, o meta-modelo e o gerador de código.

No Capítulo 5 apresenta-se o experimento que contém o método experimental e o planejamento que está subdividido em definição de hipóteses, seleção dos participantes, descrição do experimento, descrição da análise e execução do experimento.

No Capítulo 6 é apresentado à análise e a discussão dos dados, tempo de desenvolvimento, qualidade da implementação e JPA x JDBC, obtidos através do experimento.

No Capítulo 7 são apresentadas as considerações finais, descrevendo a relevância do estudo, as contribuições da pesquisa, as limitações e os trabalhos.

Por fim, os Apêndices contidos nestes trabalhos, apresentam:

- Os Apêndices A e B apresentam os artefatos utilizados para a criação do *Framework*, tais como: O XML do meta-modelo e o *template* para a geração de código, respectivamente.
- O Apêndice C apresenta os artefatos gerados pelo *Framework*, ou seja, o código de persistência de dados JPA.
- Os Apêndices D até o G detalham a documentação utilizada para realizar o experimento.
- O Apêndice H apresenta o termo de consentimento livro e esclarecimento do experimento realizado.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa apresentar os principais conceitos e ferramentas que embasam o desenvolvimento deste trabalho.

2.1 DESENVOLVIMENTO DIRIGIDO A MODELO

Desenvolvimento Dirigido a Modelo, ou MDD, é uma abordagem para projeto e desenvolvimento de software fortemente concentrada e baseada em modelos, através dos quais podem ser construídos modelos de software independentes de plataforma. O MDD implica no aumento da abstração de linguagens de programação de alto nível para linguagens de modelagem (FERNANDES; MACHADO; CARVALHO, 2008).

De acordo com Singh e Sood (2009), a principal diferença do MDD quando comparado ao desenvolvimento de software tradicional é a exploração de modelos como base para gerar código fonte de baixo nível. Selic (2008) menciona que os modelos acabam apenas como documentação, eles são de valor limitado, porque a documentação diverge muito facilmente da realidade. Consequentemente, uma premissa chave por trás do MDD é que os programas são gerados automaticamente a partir de seus modelos correspondentes.

Vidyapeetham (2009) aponta que o MDD compreende o uso de modelos no ciclo de vida do desenvolvimento de software e argumenta que o MDD automatiza o desenvolvimento de software via processamento de modelos, transformação de modelos, e técnicas de geração de código.

Como a ideia do MDD é representar um domínio ou problema por meio de modelos, Mizuno, Matsumoto e Mori (2010) apresentam o MDD como uma abordagem capaz de lidar satisfatoriamente com variações lógicas de negócios e tecnologias de baixo nível. Além disso, os autores também afirmam que MDD não só reduz o custo de desenvolvimento, mas também conflitos entre o projeto da aplicação e implementação por meio da adoção de modelos.

Além disso, Heitkötter, Majchrzak e Kuchen (2013), argumentam que o MDD pode refletir uma aplicação por meio de modelos, e sobre estes modelos, um é capaz de realizar transformações automáticas, a fim de gerar código-fonte de

baixo nível. Ou seja, os desenvolvedores podem criar aplicações que utilizam especificações de alto nível e dessas especificações, gerar um produto de trabalho final. Assim, o MDD pode não trazer uma implementação específica de uma aplicação, mas uma abstração de alto nível da aplicação, o que leva a ciclos rápidos de desenvolvimento (HEITKÖTTER *et. al.*, 2013).

Stahl e Voelter (2016) elegem os principais objetivos do MDD como:

1. O aumento da velocidade de desenvolvimento por meio da automação: O código-fonte de baixo nível pode ser gerado a partir de modelos através de transformações do modelo.
2. A melhoria da qualidade do software, especialmente porque uma arquitetura de software deverá repetir-se uniformemente uma vez que foi definido em um modelo e gerado por transformações.
3. A implementação de aspectos transversais podem ser especificados nas regras de transformação, o que também se aplica para corrigir bugs no código de saída. Separação de interesses não só conduz a uma melhor capacidade de manutenção de sistemas de software através da eliminação de redundância, mas também melhor capacidade de gerenciamento de mudanças de tecnologias.
4. Os modelos e transformações podem ser adotados como uma linha de produção de software para a construção de sistemas de software, levando a um maior nível de reutilização.
5. Através da abstração, melhorar a capacidade de gerenciamento da complexidade como modelos permite aos desenvolvedores programar em maior nível de abstração.

Além disso, Fernandes *et. al.* (2008) especificam as principais contribuições do uso do MDD:

1. **Ganho de produtividade**: Atkinson e Kuhne (2003) afirmam que a melhoria da produtividade dos esforços de desenvolvimento é a “motivação fundamental do MDD”. Além disso, consideram que a produtividade se deve a dois fatores: A produtividade de curto prazo que é obtida do quanto de funcionalidade um artefato de software

pode oferecer e a produtividade de longo prazo que é obtida pelo aumento da longevidade do artefato.

2. **Os conceitos de desenvolvimento mais próximo ao domínio:** Segundo Selic (2008), a grande vantagem de se utilizar o MDD é que se consegue expressar modelos usando conceitos que são muito menos ligada à tecnologia de implementação subjacente e são muito mais próximo do domínio do problema para a maioria das linguagens de programação populares. Além disso, o foco da modelagem permite mitigar e eliminar erros na passagem de um modelo abstrato para um produto final de implementação.
3. **Automação e menor sensibilidade às mudanças tecnológicas**: Mellor, Clark e Futagami (2003) ressaltam que as vantagens do MDD é a transformação automática dos modelos de projeto de alto nível em sistemas em execução, e a facilidade de uso e manutenção dos modelos, os quais são menos sensíveis à tecnologia escolhida.
4. **Captura de conhecimentos especializados e reutilização**: Além dos benefícios da utilização de conceitos de nível superior em uma linguagem de modelagem, MDD também permite a captura de conhecimento especializado. Isto é possível por meio de funções de mapeamento que transmitem informação para a transformação de um modelo para o outro, permitindo à reutilização quando uma aplicação ou as suas mudanças de implementação. Isto permite uma evolução independente dos modelos e conduz a uma maior longevidade dos modelos (FERNANDES *et. al.*, 2008).

A evolução tanto da modelagem de software quanto da geração automática de código, aprimorou a compreensão de como modelar um software. E Selic (2008) aponta que o fato de compreender melhor como modelar o software, o MDD se torna mais útil por causa de dois desenvolvimentos evolutivos fundamentais: O amadurecimento das tecnologias de automação necessárias e o surgimento de novos padrões.

No entanto, para alcançar os objetivos do MDD, existem técnicas e ferramentas que auxiliam neste processo. De acordo com Selic (2008), as técnicas

e ferramentas para fazer isso com sucesso atingiram um alto nível de maturidade, mesmo em aplicações industriais de grande escala. Os geradores de código modernos e tecnologias relacionadas podem produzir código cuja eficiência é comparável ao (e às vezes melhor do que) códigos feito à mão.

2.2 PROGRAMAÇÃO ORIENTADA A ASPECTOS

O uso da programação orientada a objetos (POO) no desenvolvimento de software teve um grande aumento devido à necessidade de se desenvolver software de qualidade, buscando maiores níveis de reuso e manutenibilidade, aumentando assim a produtividade no desenvolvimento e no suporte a mudanças de requisitos (MEYER, 1997). No entanto, apesar da orientação a objetos ser o paradigma de programação mais popular, as abstrações são reconhecidas como sendo incapazes de capturar todas as preocupações de interesse no sistema de software (KICZALES, 1997).

De acordo com Soares e Borba (2002), a programação orientada a aspectos (POA) surgiu para solucionar a limitação da POO em capturar algumas decisões de projeto que um sistema deve implementar, causando assim a distribuição pelo código e espalhamento de código em diversas classes com diferentes propósitos. Este entrelaçamento e espalhamento tornam difícil o desenvolvimento e a manutenção destes sistemas. Desta forma, a POA aumenta a modularidade separando código que implementa funções específicas, afetando diferentes partes do sistema, chamadas preocupações ortogonais (*crosscutting concern*) (SOARES; BORBA, 2002).

Sendo assim, a POA é constituída de interesses, que são requisitos, propriedades ou funcionalidades do sistema. Estes interesses podem ser implementados tanto por classes, o qual agrupam e modularizam uma funcionalidade, e por aspectos, que implementam códigos que estariam espalhados em um sistema OO e afetariam o comportamento do sistema, e estes são os denominados interesses transversais.

A POA ficou conhecida como uma meta-programação, por causa da característica do seu compilador, pois os compiladores destinados a POA não geram um produto final (programa compilável, executável ou interpretável), mas sim

um novo código. Nesta compilação são acrescentados novos elementos no código para dar suporte nas novas abstrações. Ainda assim, o código resultante deve ser novamente compilado, para aí sim gerar o produto final. E segundo Winck, Goetten Junior (2006), mais uma característica da meta-programação usada em POA é a reflexão computacional, o qual parte do código gerado é destinada a alterar características do programa.

Os princípios da POA são separar o código referente ao negócio do sistema dos interesses transversais, de forma bem definida e centralizada. De acordo com Chavez e Garcia (2003) os interesses são modularizados por meio de diferentes abstrações providas por linguagens, métodos e ferramentas, como mostra a Figura 2.1.

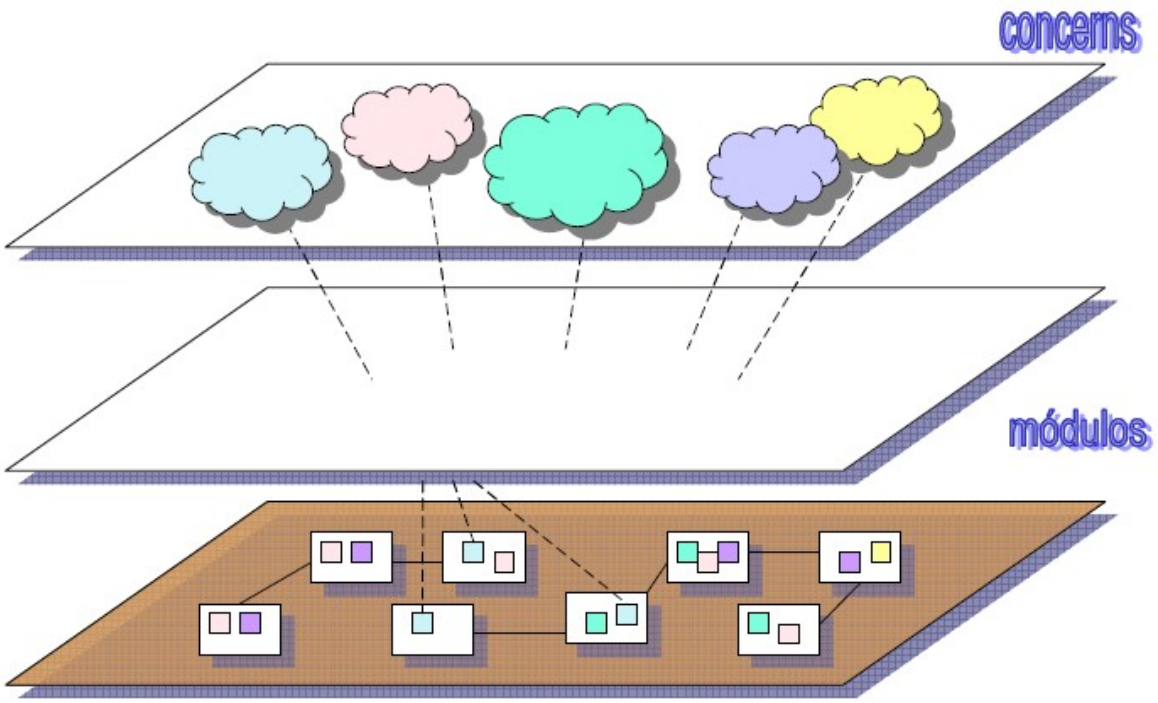


Figura 2.1 - Separação de interesses com POA (FONTE: Chavez & Garcia, 2003).

Observa-se que na Figura 2.1, cada nuvem representa um interesse sistêmico implementado no sistema, por estarem bem separados e definidos, os componentes podem ser melhor reutilizados e a sua manutenção e legibilidade torna-se mais agradável.

A POA foi proposta como uma técnica para melhorar a separação das preocupações na construção de software OO, melhorando a usabilidade e facilidade de evolução (KICZALES, 1997).

Na Figura 2.2, é apresentado o esquema de uma implementação OO. Neste exemplo, apesar dos problemas que o POO pode apresentar, ele satisfaz grande parte dos objetivos para os quais este paradigma foi criado, porém os códigos em vermelho, que representam algum interesse transversal, se espalham por várias classes, fazendo assim com que muitas classes tenham funcionalidades para os quais não foram projetadas.

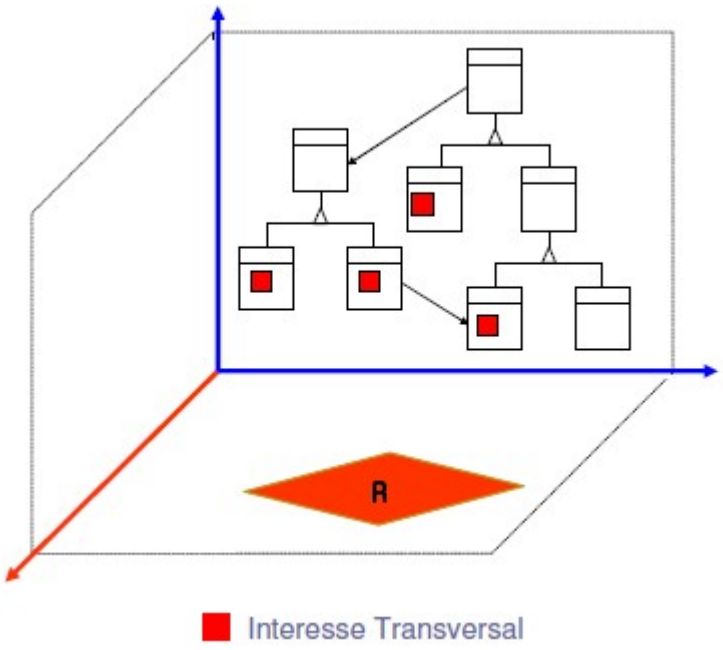


Figura 2.2 - Situação problema em OO (FONTE: Chavez & Garcia, 2003)

Analisando a Figura 2.2, uma solução apontada pela POA é isolar os requisitos transversais e fazer um novo tipo de módulo (R). Compor, combinar nos locais desejados e reutilizar, combinando em outros locais. Dessa forma, aplicando as modificações necessárias à solução em POA é apresentada na Figura 2.3, o qual os requisitos que estavam espalhados por diversas classes ficam isolados em R – no aspecto, que contém os interesses transversais.

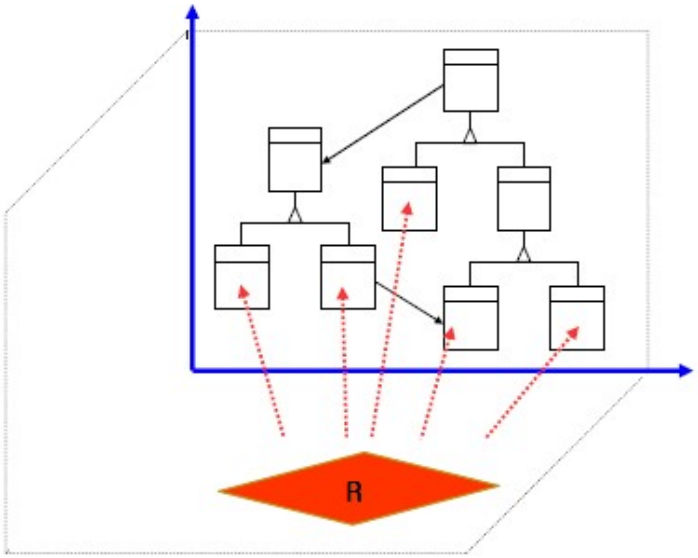


Figura 2.3 - Solução do problema com POA (FONTE: Chavez& Garcia, 2003)

2.2.1 ASPECTJ

A linguagem AspectJ, é uma extensão orientada a aspectos, de propósito geral, da linguagem Java (Gosgling *et al.*, 2000). Esta linguagem utiliza conceitos relacionados à POA. A seguir são mostrados alguns conceitos básicos da POA que o AspectJ implementa:

- **Pontos de junção** (*Join points*): um ponto de junção é qualquer ponto identificável pelo AspectJ durante a execução do programa. Segundo Chaves e Garcia (2003), "é aplicado em métodos (chamadas ou execução), construtor (chamadas ou execução de métodos), execução de tratamento de exceções e atributos (consultar, modificar valores)." E de acordo com Winck e Goetten Junior (2006), "a partir do ponto de junção, são criadas regras que dão origem aos pontos de atuação."
- **Pontos de corte** (*Pointcuts*): o objetivo dos pontos de corte é a criação de regras gerais para definição de eventos que serão os pontos de junção, os pontos de corte devem apresentar dados de execução relativos a cada ponto de junção, que serão utilizados pelo método acionado do ponto de junção mapeado no ponto de atuação. Os pontos de corte são formados pela composição de pontos de junção, através dos operadores && (e), || (ou), e ! (não). Utilizando pontos de corte podemos obter valores de argumentos de métodos, objetos em execução, atributos e exceções dos pontos de junção (SOARES;

BORBA, 2002). Nos pontos de corte podem-se utilizar expressões regulares para se especificar os pontos de corte, permitindo grande flexibilidade.

- **Advice:** de acordo com Chavez e Garcia (2003), "*advice* é um trecho de código a ser executado quando um ponto de corte for atingido." O advice, ou "adendo", pode ser executado antes (*before*), depois (*after*), ou "em volta" (*around*) do ponto de junção. O *before* é executado antes do *join point* ser executado e o *after* é executado depois do *join point*, enquanto no *around* o adendo escolhe quando vai executar, podendo ser antes ou depois.
- **Declarações intertipos (*inter-type declarations, introductions*):** aspectos podem declarar membros (campos, métodos e construtores) que pertencem a outros tipos. Estes são chamados de membros intertipos. Aspectos também podem declarar novas interfaces de outros tipos ou estender uma classe.
- **Aspectos**: a principal construção em AspectJ é um aspecto. Cada aspecto define uma função específica que pode afetar várias partes de um sistema como, por exemplo, distribuição. Um aspecto, como uma classe Java, pode definir membros (atributos e métodos) e uma hierarquia de aspectos, através da definição de aspectos especializados (SOARES; BORBA, 2002). E um aspecto agrupa ponte de corte, *advice* e intertipos.

Para entender a importância da POA nesta linguagem, a Figura 2.4 ilustra uma parte de um diagrama de classes de um editor de imagens. As classes *Line* e *Point* desse editor deveriam conter apenas código para implementação da linha e do ponto, mas como pode se observar, o interesse em destaque é a atualização da imagem (*Display Update*), ou seja, toda vez que uma linha ou ponto é criado ou modificado, é necessário que se atualize a imagem, e este interesse está espalhado pelas classes *Line* e *Point*. Na Figura 2.4, pode-se perceber que os conceitos de modularização e encapsulamento são quebrados, pois as classes *Line* e *Point* atendem interesses externos.

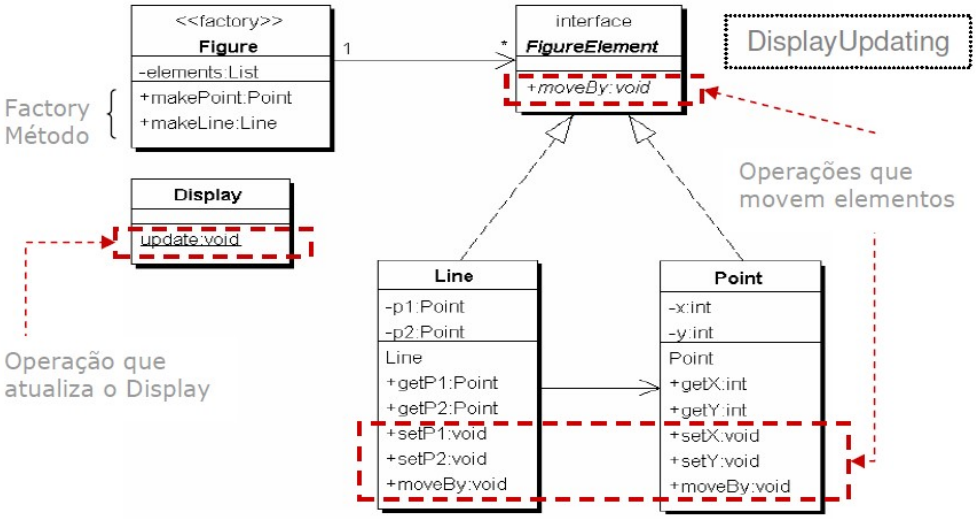


Figura 2.4 - Interesses Transversais na atualização da imagem.

Para resolver este problema, basta colocar este interesse transversal em um aspecto, como é ilustrado na Figura 2.5. Com esta solução, o interesse transversal ficará modularizado em um aspecto, assim as classes *Line* e *Point* só teriam código para implementação da Linha e do Ponto.

No entanto, programas orientados a aspectos precisam de um compilador específico. No caso da linguagem AspectJ, o compilador *ajc* transforma um programa escrito em AspectJ em um programa *Bytecode* Java que pode ser executado em qualquer *Java Virtual Machine* (JVM).

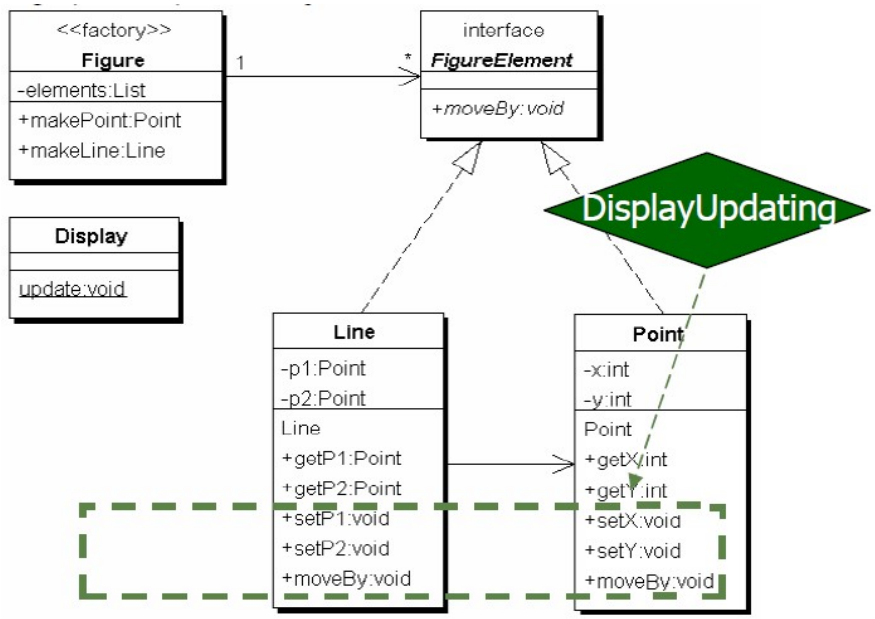


Figura 2.5 - Interesses Transversais na atualização da imagem em um Aspecto.

2.3 JAVA PERSISTENCE API

A plataforma Java por um longo tempo teve trabalho em lidar com sistemas de bancos de dados, e passou o mesmo problema que as outras linguagens de programação orientada a objetos, mover os dados entre um sistema de banco de dados relacional para um modelo de objeto de um aplicativo Java.

De acordo com Keith e Schincariol (2009), a técnica para fazer a ponte entre o modelo de objeto e o modelo relacional é conhecida como mapeamento objeto-relacional, muitas vezes referido como mapeamento de Objeto-Relacional (O-R) ou simplesmente *Object Relational Mapping* (ORM).

Contudo, uma solução padrão, o Java *Persistence* API (JPA), foi introduzido na plataforma para preencher a lacuna entre modelos de domínio de orientação a objetos e sistemas de banco de dados relacionais (KEITH; SCHINCARIOL, 2009).

O JPA é um *framework*, baseado em *Plain Old Java Object* (POJO) para persistência Java, ou seja, transforma as classes Java em entidades, e a partir dessa transformação consegue-se persistir os dados no banco de dados (KEITH; SCHINCARIOL, 2009).

A técnica de persistência de dados JPA, visa solucionar os problemas encontrados no *Java Database Connectivity* (JDBC), relacionados à codificação para o acesso aos dados e a portabilidade, pois as *queries* podem variar entre banco de dados e muitos programadores não sentem confortáveis para escrever código *Structure Query Language* (SQL). Portanto, o JPA fornece uma interface comum para *frameworks* de persistência de dados, ocultando o baixo nível de acesso de dados. Nessa forma, o desenvolvedor não precisa se preocupar com o banco de dados, podendo focar apenas no desenvolvimento da aplicação.

As principais características do JPA são:

- **POJO:** é a característica mais importante do JPA, pois todos os objetos são POJOs, isto significa que são objetos sem nada de especial, ou seja, possuem seus atributos, seus métodos e seus construtores normais. Porém, o mapeamento objeto-relacional com o JPA é por meio de meta-dados. Este mapeamento pode ser feito por meio de anotações ou utilizando *Extensible Markup Language* (XML).

- **Consultas de objeto:** segundo Keith e Schincariol (2009), as consultas podem ser expressas em *Java Persistence Query Language* (JPQL), uma linguagem de consulta que é derivado de *Enterprise JavaBeans Query Language* (EJBQL) e após modelado em SQL. As consultas usam um esquema de abstração que é baseada no modelo de entidade.
- **Configuração simples:** de acordo com Keith e Schincariol (2009), existe um grande número de características de persistência que a especificação oferece, todas são configuráveis por meio de anotações, XML ou uma combinação das duas.
- **Integração e Teste:** Keith e Schincariol (2009) afirmam que com o JPA é possível escrever código de persistência integrada no servidor e ser capaz de reutilizá-lo para testar fora do servidor.

2.4 ACCELEO

Atualmente, diferentes ferramentas para transformação de modelos orientadas a uma variedade de propósitos têm sido exploradas na literatura. Por exemplo, *Extensible Stylesheet Language Transformation* (XSLT), *Query-View Transformation Language* (QVTL), *ATLAS Transformation Language* (ATL) e *Acceleo* (KOCH; 2007).

O *Acceleo* é uma ferramenta que visa melhorar o desenvolvimento de software que enfatiza o conceito de *Model-Driven Architecture* (MDA). Esta ferramenta é uma implementação do padrão *Meta-Object Facility* (MOF) dirigido pela *Object Management Group* (OMG). Além disso, fornece um mecanismo simples e poderoso para desenvolvimento de geradores de código de execução da especificação *Model-to-text* (M2T) da OMG (ACCELEO, 2016). A ferramenta ajuda o desenvolvedor com a maioria das funcionalidades que podem ser esperado de uma *Integrated Development Environment* (IDE) geradora de código de ótima qualidade: Sintaxe simples, geração de código eficiente, ferramentas avançadas, funcionalidades a par com o JDT (ACCELEO, 2016). A Quadro 2.1 demonstra as funcionalidades do *Acceleo*, os quadrados que estão em cor verde são disponíveis e de cor vermelho não são disponíveis no *Acceleo*.

Quadro 2-1 - Funcionalidades do *Acceleo*.

destaque de sintaxe	realização	dobramento de código	Compilador	Refatoração (modelo extrato)
modelos de código personalizáveis	Abrir declaração	Profiler	Debugger	Refatoração (renomear)
mecanismo de geração	Esboço	Stand Aloneruntime	detecção de erros	Refatoração (consulta de extração)
suporte documentação	Flutuar	Correções rápidas	rastreabilidade	Refatoração (puxar para cima)
caracteres em branco visíveis	esboço rápido	pesquisa ocorrências	framework de testes	Geração incremental
Tipo de hierarquia rápida	Chamada vista hierarquia	avisos	suporteAnt	suporteMaven
Ver declaração	Integração Java Avançado	recipienteclasspath	suporteCheckstyle	suportecaseCamel
análise distribuída	integração BIRT	Folha de dicas	geração de documentação	Exportação deperfil de dados gprof

(FONTE: ACCELEO, 2016).

O *Acceleo* permite a importação de um modelo de diagrama Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML) ou *Systems Modeling Language* (SysML), e realiza a transformação para uma linguagem de programação como JAVA, C#, C++, PhP. Esta ferramenta não faz a modelagem de diagramas, exigindo que o usuário utilize outras ferramentas de modelagem para conseguir utilizar o *Acceleo*.

A Figura 2.6 apresenta a ferramenta *Acceleo* e para o seu funcionamento são necessárias três etapas que são descritas a seguir:

1. Inicialmente para utilizar o *Acceleo* é necessário que o usuário tenha o modelo do diagrama UML ou SysML definido.
2. Após a definição do modelo do diagrama, é preciso escolher a linguagem de programação (JAVA, C#, C++) que deseja transformar o modelo.
3. Por fim, é feita a importação do modelo do diagrama dentro do *Acceleo* e ao executá-lo, iniciará a transformação do modelo para a

linguagem de programação escolhida na etapa anterior e será gerado o código para o usuário.

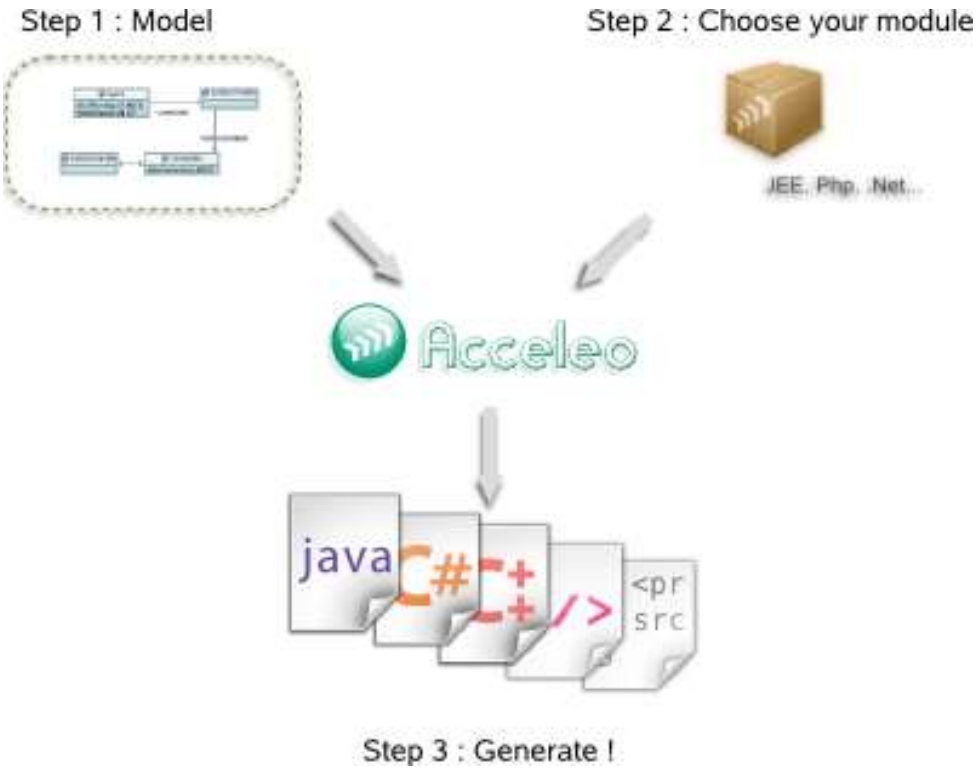


Figura 2.6 - Ferramenta *Acceleo* (FONTE: ACCELEO, 2016).

3. ESTRUTURAÇÃO DA PESQUISA

Este capítulo apresenta a metodologia e os métodos de pesquisa utilizados para alcançar o objetivo geral do trabalho. Ainda neste capítulo será abordada a caracterização e a estruturação da pesquisa.

3.1 CARACTERIZAÇÃO DA PESQUISA

De acordo com Gil (2002), há uma necessidade de se construir um modelo conceitual e operacional da pesquisa, com o intuito de comparar a visão teórica com os dados da realidade.

Sendo assim, esta pesquisa pode ser classificada como um estudo descritivo. Descritivo, uma vez que, baseado no conhecimento teórico e em análises realizadas durante o trabalho, busca a compreensão e a descrição de como esta proposta pode auxiliar os projetista de software a decidir qual a melhor forma de se implementar a persistência de dados para situações específicas.

Portanto, esta pesquisa é qualitativa, buscando verificar se a abordagem proposta é viável através da coleta de dados por meio do experimento realizado, após a aplicação da abordagem nos alunos do curso de Ciência da Computação da Universidade Estadual do Norte do Paraná.

3.2 ESTRUTURAÇÃO DA PESQUISA

A pesquisa pode ser considerada equivalente a um planejamento estratégico (MIGUEL, 2007), em que uma abordagem metodológica compreende diversos níveis de abrangência e profundidade. Algumas decisões são estratégicas, como a escolha da abordagem e do método, e outras são mais de ordem tática e operacional, tais como procedimentos de condução da pesquisa (REINEHR, 2008).

Desta maneira, foi definida uma estrutura de pesquisa para ser seguida, a fim de cumprir com os objetivos iniciais propostos na Seção 1.2, como pode ser visto na Figura 3.1. Esta estrutura é dividida em quatro fases (Planejamento Inicial, Fase Exploratória, Desenvolvimento e Conclusão), sendo que cada fase contém um ou mais etapas que estão ligadas entre si por meio de setas que demonstram a direção do fluxo e da ação decorrente.

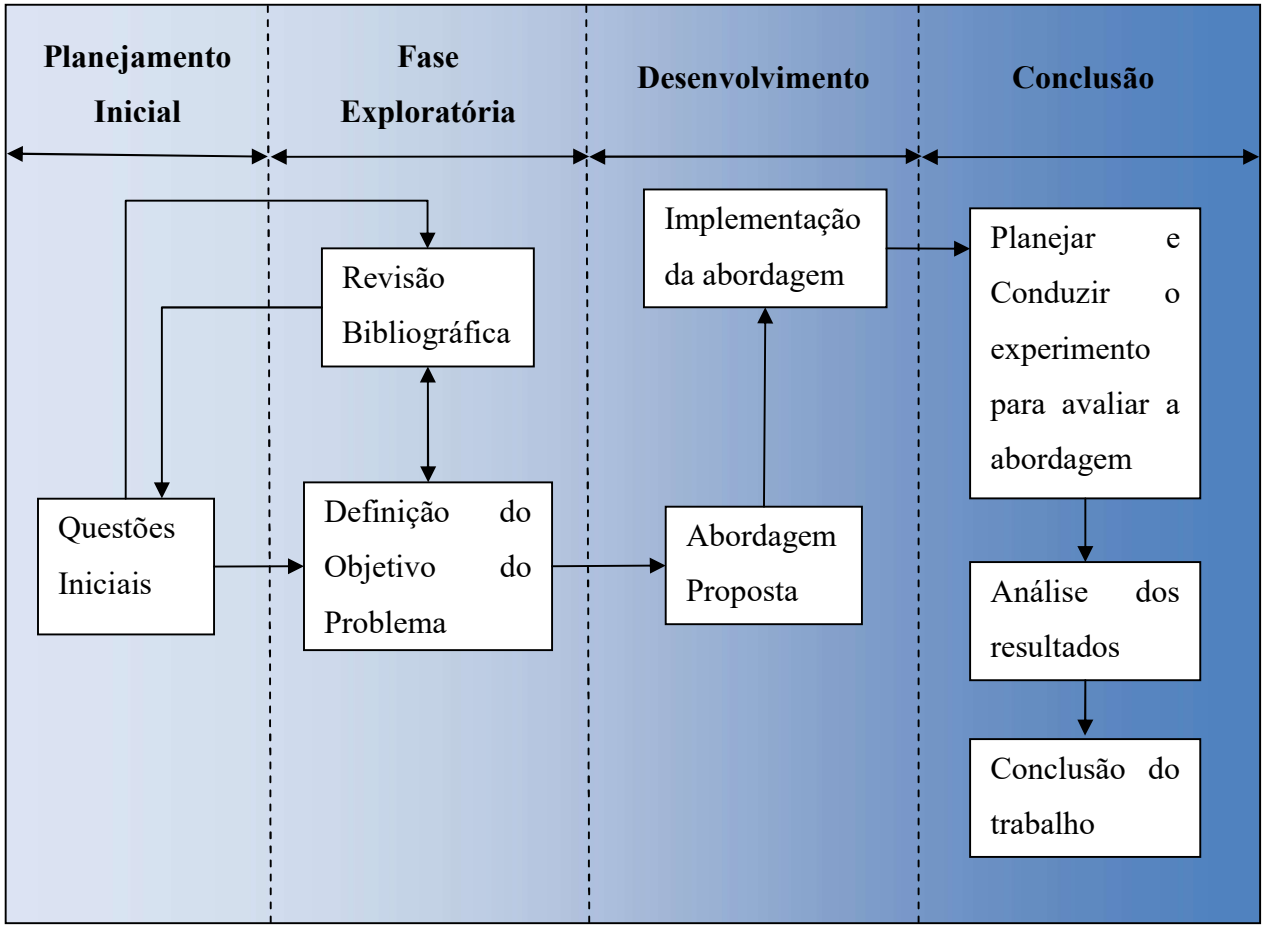


Figura 3.1- Estrutura da Pesquisa

3.2.1 PLANEJAMENTO INICIAL

Depois de definir a área de pesquisa, algumas questões são necessárias para dar início à pesquisa, tais como: "**Como o MDD e POA são utilizados na Engenharia de Software?**", "**Quais são os domínios que podem ser aplicado o conceito de MDD e POA?**", "**Qual as vantagens e desvantagens de utilizar MDD e POA?**", "**Quais ferramentas são necessárias para auxiliar neste processo?**".

Entretanto, essas questões iniciais têm o intuito de nortear a pesquisa, visto que existe um vasto domínio em que pode se aplicar os conceitos de MDD e POA. Após estabelecer as questões iniciais, pode-se determinar a área de estudo e os objetivos a serem alcançados, mas para definir e alcançar esses objetivos é necessário explorar o campo de estudo, seguindo para a próxima fase do trabalho, a Fase Exploratória.

3.2.2 FASE EXPLORATÓRIA

A fase exploratória tem como principal objetivo proporcionar uma maior familiaridade com o problema, ou seja, tornando-o explicito (GIL, 2002). Para isto é necessário realizar duas etapas: a revisão bibliográfica e a definição do objetivo do trabalho.

Na etapa da revisão bibliográfica, são pesquisados assuntos e trabalhos relacionados de forma aprofundada e bem detalhada. Mafra e Travassos (2006) afirmam que, o levantamento bibliográfico se faz necessário uma vez que a revisão de literatura é o meio pelo qual o pesquisador pode identificar o conhecimento científico existente em uma determinada área, de forma a planejar sua pesquisa, evitando a duplicação de esforços e a repetição de erros passados.

A etapa da definição do objetivo do trabalho é definida todos os objetivos do trabalho que ainda estavam vagos na fase anterior, deixando-os compreensíveis, determinando de fato o que será realizado durante todo o trabalho (GIL, 2002). Esta definição se baseia nas questões iniciais, pois as mesmas delimitam o escopo de trabalho, auxiliando a identificar os problemas a serem solucionados.

3.2.3 DESENVOLVIMENTO

O desenvolvimento é a fase que utiliza todas as questões iniciais levantadas na fase do planejamento inicial. Além disso, todas as ideias e conhecimento adquirido na fase exploratória são colocados em prática a fim de responder as questões propostas.

Sendo assim, para alcançar os objetivos propostos pelo trabalho, esta etapa é divida em duas etapas subsequentes: abordagem proposta e implementação da abordagem.

A abordagem proposta tem como intuito apresentar uma abordagem, que será necessário propor uma arquitetura para implementá-la. Esta arquitetura define e organiza todos os componentes necessários para desenvolver uma ferramenta para geração de código de persistência de dados JPA.

A implementação da abordagem permite alcançar o objetivo principal deste trabalho, a implementação de uma versão inicial do *framework* baseada na arquitetura e abordagem proposta foi realizada. Os componentes definidos na arquitetura serão implementados na linguagem de programação Java.

3.2.4 CONCLUSÃO

A conclusão é a última fase da estrutura da pesquisa apresentada na Figura 3.1, na qual é dividida em três etapas: Planejar e conduzir o experimento para avaliar a abordagem, discussão dos resultados e conclusão do trabalho.

O planejamento e a condução do experimento para avaliação da abordagem se deram inicialmente escolhendo a melhor forma de ser realizada. Portanto, foi definida a utilização de métricas, que tem como intuito analisar os dados obtidos em um ambiente controlado para confirmar ou negar as hipóteses que estão definidas na Seção 5.2.1. Os participantes que realizaram o experimento são os alunos do curso de Ciência da Computação da Universidade Estadual do Norte do Paraná.

A análise dos dados é relevante, pois como se trata de persistência de dados JPA, existem diferentes tecnologias e padrões para a sua persistência, sendo assim é preciso analisar e fazer comparação com outra abordagem, a que utiliza o JDBC, com o intuito de verificar qual é a mais viável.

Por fim a conclusão do trabalho, após o levantamento geral da pesquisa, com todas as informações pertinentes, a implementação da abordagem e a discussão dos resultados. Nesta etapa, os pontos positivos e negativos são destacados, sendo assim é possível concluir o trabalho.

4. ABORDAGEM PROPOSTA

Neste capítulo é apresentada uma visão geral da abordagem proposta, o meta-modelo desenvolvido, que reflete o domínio de aplicações de persistência de dados JPA, o gerador de código utilizado e os papéis de quem usa a abordagem.

4.1 ABORDAGEM PROPOSTA – UMA VISÃO GERAL

Na área de desenvolvimento de software, novas técnicas e métodos estão sempre sendo propostos, a fim de reduzir a complexidade do desenvolvimento de software. No entanto, aplicar novos métodos e técnicas, muitas vezes não é uma tarefa trivial, visto que os programadores necessitam aprender essas técnicas e saibam onde aplicá-las. Além disso, é sempre desejável que as empresas apliquem padrões de programação, ou seja, todos os programadores tenham o mesmo entendimento de como aplicar algum conceito. Por exemplo, se uma empresa quer adotar a POA, sua equipe precisa saber em quais situações o uso dessa abordagem é vantajosa, além de saber como implementá-la.

Portanto, uma implementação correta da POA não é simples. Muitos estudos indicam em que situações esta abordagem é viável, assim, uma boa estratégia é um especialista definir quando ela pode ser usada. Dessa maneira, a abordagem proposta para a ferramenta visa auxiliar o desenvolvedor a definir como e onde POA pode ser aplicada, utilizando uma abordagem orientada a modelos. A Figura 4.1 demonstra como a abordagem funciona.

Na abordagem proposta há dois papéis:

- **O Desenvolvedor da Ferramenta:** Responsável por criar o meta-modelo e as transformações;
- **O Desenvolvedor da Aplicação:** Responsável por criar um modelo e posteriormente uma aplicação baseada no modelo, meta-modelo e nas transformações existentes.

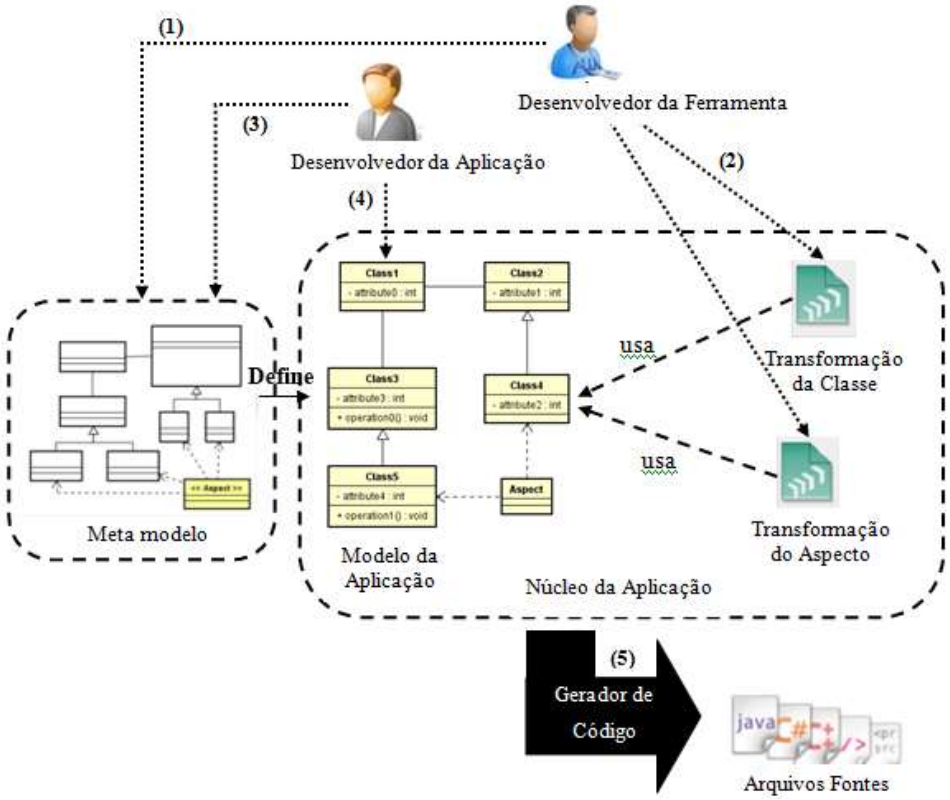


Figura 4.1 -- Abordagem Proposta

O primeiro passo (1) desta abordagem é a criação do meta-modelo. O meta-modelo (veja na Seção 4.2) descreve os conceitos essenciais que uma aplicação deve ter para um domínio específico, portanto este meta-modelo é um guia para criar aplicações deste domínio. Dessa forma, qualquer aplicação desenvolvida pelo Desenvolvedor de Aplicação deve ser criado a partir do meta-modelo.

Uma diferença na abordagem proposta, é que, no meta-modelo devem ser especificados os aspectos e em quais classes devem agir. Desta forma, o Desenvolvedor da Ferramenta é responsável por definir quais aspectos o modelo deve conter, ajudando a criar um padrão organizacional, uma vez que qualquer aplicação terá os mesmos aspectos e a mesma implementação. Assim, a ferramenta possui um meta-modelo para persistência de dados utilizando a tecnologia JPA.

Após a construção do meta-modelo, o segundo passo (2) é criar as transformações do modelo. Para tais, é proposta a utilização do *Acceleo* para realizar as transformações necessárias (veja na seção 4.3).Nesta etapa é projetado o gerador de códigos através de um projeto *Acceleo* e por meio dos *template*, que são criados pelo Desenvolvedor da Ferramenta, é criado o gerador de código.

Uma vez construído o meta-modelo e as transformações, as aplicações podem ser desenvolvidas. No terceiro passo (3), o desenvolvedor da aplicação seleciona o meta-modelo para implementação da persistência de dados JPA, e a partir disso é iniciado o desenvolvimento da aplicação.

Após isto, o quarto passo (4) é constituído pela criação do modelo da aplicação. Nesta etapa, o Desenvolvedor da Aplicação cria o modelo da aplicação, que é o modelo criado e configurado com base no meta-modelo. O modelo criado em conjunto com as transformações constitui o núcleo da aplicação, a partir destes elementos, pode ser gerado o código fonte.

Por fim, no quinto passo (5), o gerador de código exige como entrada um modelo de aplicação e as transformações de classes e aspectos para a geração de código fonte de baixo nível como saída.

4.2 META-MODELO

O desenvolvimento dirigido a modelos pode ser em muitos casos iniciado a partir de um meta-modelo que contém regras, especificações e informações relevantes que construirão os modelos. De certa forma, é comum a existência de um meta-modelo por trás das cenas que decidirá o que é possível de ser realizado por meio de determinado modelo. De acordo com Guedes (2012), o meta-modelo define uma linguagem para expressar modelos, sendo mais compacto que um modelo que ele descreve.

Segundo Koch (2012), o Eclipse é uma plataforma de desenvolvimento baseada em *plugins* que suporta diversas linguagens de programação, por exemplo, Ada, ABAP, C, C++, Clojure, COBOL, Fortran, Groovy, Haskell, JavaScript, Lasso, Natural, Perl, PHP, Prolog, Python, R, Ruby, Scala, Scheme, entre outras ferramentas. Koch (2012) cita as principais características do Eclipse:

a) Livre integração com outras tecnologias;
b) Vasto suporte;
c) Comunidade altamente ativa;
d) Disponibilidade de *plugins*;
e) Suporte para múltiplcas linguagens de programação e modelagem; e
f) Ampla adoção e uso por desenvolvedores em serviço.

Além disto, o Eclipse e o *Eclipse Modeling Framework* (EMF), tem se tornado os padrões na comunidade de engenharia orientada a modelos (*Model Driven Engineering – MDE*), como também na maioria das ferramentas MDE presentes no mercado (KOLOVOS *et al.*, 2010).

EMF é um *framework* de modelagem e mecanismo de geração de código para desenvolvimento de aplicações baseadas em um modelo de dados estruturados (NELLAIAPPAN, 2009). Portanto, devido a sua vasta popularidade e as ferramentas disponíveis, o Eclipse e o EMF são as alternativas pertinentes para o desenvolvimento do meta-modelo para persistência de dados utilizando a tecnologia JPA.

Para começar a criação do meta-modelo foi necessário utilizar o *Ecore Diagram*, um *plugin* que proporciona a criação de um modelo *Ecore* de maneira amigável, pois o *Ecore* corresponde a uma extensão XML que utiliza *namespaces* para definir novas entidades e atributos, seu XML pode ser encontrado no Apêndice A. Por meio do *Ecore Diagram*, é possível efetuar uma representação visual, estruturada e simplificada de um determinado conceito.

Pode-se observar na Figura 4.2, através do *plugin Ecore Diagram* foi possível elaborar a modelagem desse meta-modelo, que servirá para qualquer aplicação de persistência de dados que utilize a tecnologia JPA.

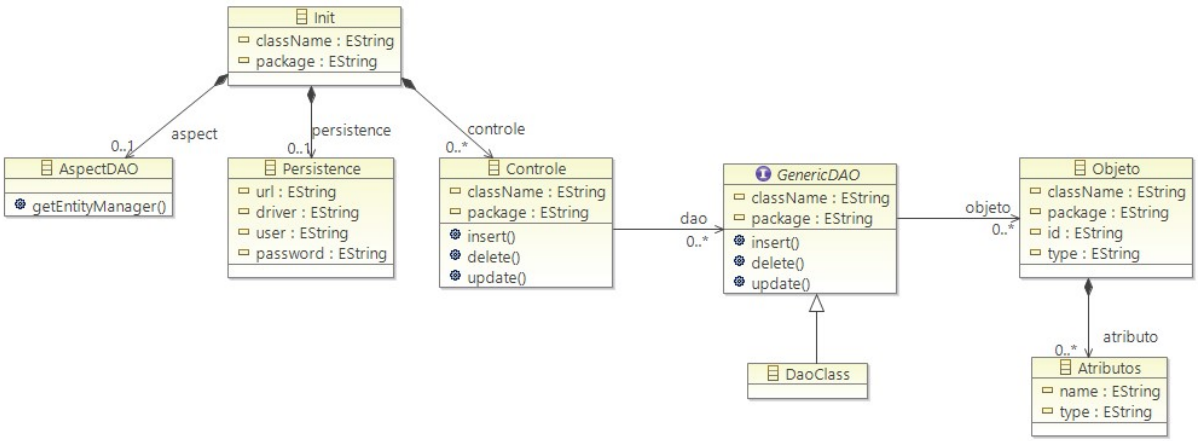


Figura 4.2 – Meta-modelo de persistência de dados JPA

A Figura 4.2 apresenta um diagrama que contém as entidades do meta-modelo e as suas relações. O meta-modelo é composto pelas seguintes entidades:

- **Init:** Tem como função ser o ponto de partida na construção do modelo. Sendo assim, é possível criar as entidades *AspectDAO*, *Persistence* e *Controle*.

- **Controle:** Associa o *Objeto* ao *DaoClass*. Além disso, a *Controle* contém um conjunto de interfaces equivalentes ao conjunto de interfaces trazido pelo *DaoClass*, que pode ser utilizado pelo *Init* para acionar a funcionalidade de persistência de dados.
- **GenericDAO:** Um esqueleto para o *DaoClass*. O *GenericDAO* é uma entidade "interface" que contém as declarações dos métodos (*insert*, *delete*, *update*) necessários de um *DaoClass*. Além disso, contém a funcionalidade que o *AspectDAO* usará como ponto de corte.
- **DaoClass:** Uma implementação da classe *GenericDAO*. Como o *DaoClass* está associado a um *Objeto*, assim, ela pode conter consultas específicas realizadas nele.
- **Objeto:** Um POJO que determina na entidade de domínio os seus atributos e comportamentos. Em outras palavras, no domínio de persistência de dados, o *Objeto* também poderia ser descrito como um elemento da camada de modelo. Nesta entidade precisa ser definido qual é o seu *id* e *type*, ressaltando que, como se utiliza a tecnologia JPA, entende-se que cada *Objeto* é uma entidade no banco de dados.
- **Atributos:** Responsável por definir quais são os atributos que a classe *Objeto* possui.
- **Persistence:** Responsável por realizar a conexão com o banco de dados através do XML, para isto, é necessário informar a *url*, o *driver*, o *user* e o *password*.
- **AspectDAO:** O *AspectDAO* detém as informações da conexão para o banco de dados que foram definidas na entidade *Persistence*, e define um ponto de corte que é acionado assim que o método *insert*, *delete* ou *update* for executado, assim realizando uma consulta ou persistindo o dado.

4.3 GERADOR DE CÓDIGO

O gerador de código utilizado na abordagem proposta é construído baseado no *Acceleo* e requer uma instância do meta-modelo de persistência de dados JPA como entrada para a geração do código. De acordo com Koch (2012) diferentes

ferramentas têm sido exploradas para transformação de modelos orientadas a uma variedade de propósito, por exemplo, XSLT, QVT, ATL e o *Acceleo*.

Para cumprir os requisitos da abordagem proposta, o *Acceleo* demonstra ser a alternativa mais adequada para a geração de código, não só devido à simplicidade e facilidade de uso da ferramenta, mas também devido à possibilidade de transformar automaticamente modelos independentes de plataforma (*Platform-Independent Models* - PIMs) para código fonte de baixo nível. (MTSWENI, 2012). Além disso, dentre essas ferramentas citadas anteriormente, o *Acceleo* se destaca, pois, fornece um mecanismo simples e poderoso para desenvolvimento de geradores de código por meio de *template*.

Os *template* são os motores para geração de código no *Acceleo*. Como apresentado na Figura 4.3, os *template* são constituídos por dois tipos de sintaxes: A sintaxe literal e sintaxe dinâmica. A sintaxe literal (em fonte de cor preta na Figura 4.3) diz respeito a tudo que será literalmente gerado no arquivo final, enquanto a sintaxe dinâmica (ou programada), escrita entre colchetes, refere-se ao aspecto dinâmico do gerador, isto é, processamento de utilização das entidades e atributos instanciados no modelo. O *template* apresentado na Figura 4.3 gera uma classe simples em Java, constituída por método principal *main*, como demonstrado na Figura 4.4.

```
[comment encoding = UTF-8 /]
[module generate('http://pretcc.com')]


[template public generateElement(aMyClass : MyClass)]
[comment @main/]
[file (aMyClass.att1.concat('.java'), false, 'UTF-8')]
public class [aMyClass.att1/] {
    public static void main(String['['/']'/] args) {
        System.out.println("Hello [aMyClass.att2/]!");
    }
}
[/file]
[/template]
```

Figura 4.3 - Exemplo de um *template* do *Acceleo* que gera uma classe Java

O conteúdo da classe apresentada na Figura 4.4, é gerado em um arquivo denominado *MyClass.java* onde o nome do arquivo, como também o nome da classe, é um atributo dinâmico, como também a palavra “Hello World!” que está armazenada dentro de um atributo dinâmico chamado *att2* em *aMyClass*.

```
public class MyClass {
        public static void main(String[] args) {
                System.out.println("Hello World!");
        }
}
```

Figura 4.4 – Código-fonte gerado pelo *template*

Na abordagem proposta, o meta-modelo possui dois tipos de transformações baseadas em *template Acceleo*, que serão utilizadas para gerar o código fonte:

- **Transformação da Classe:** Utiliza o meta-modelo e o modelo para criar código fonte de classe Java.
- **Transformação do Aspecto:** Utiliza o meta-modelo e o modelo para criar código fonte dos aspectos.

Neste capítulo é utilizado um exemplo simples de como é criado o gerador de código, a fim de demonstrar o seu funcionamento. A versão completa do mesmo é anexado no Apêndice B enquanto a versão completa do código gerado está no Apêndice C.

Para projetar um modelo do *framework*, um *plugin* do Eclipse é construído por meio de um meta-modelo contido no EMF. Portanto, uma vez que o modelo é criado, o gerador de código gera o código fonte de acordo com as informações utilizadas no modelo usado como entrada. A Figura 4.5, ilustra as etapas da ferramenta *Acceleo* para a geração de código.

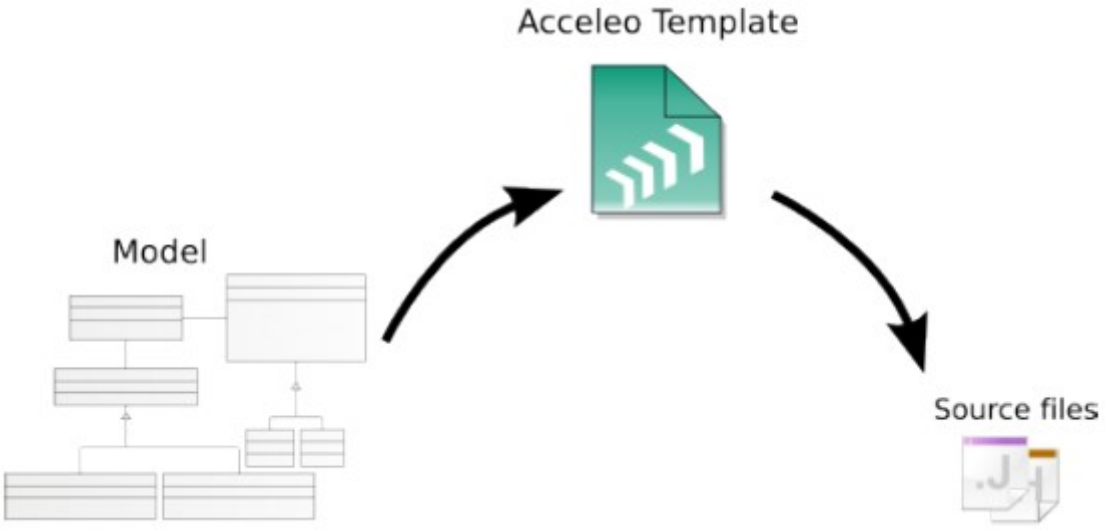


Figura 4.5 - Etapas para geração de código com o *Acceleo* na abordagem (FONTE: ACCELEO, 2016)

4.4 OS PAPÉIS

Como apresentado na Figura 4.2, a abordagem proposta envolve partes distintas, ou seja, o desenvolvedor do *framework* (1) cria o meta-modelo e (2) as transformações necessárias, o desenvolvedor da aplicação (3) seleciona o meta-modelo, (4) cria o modelo da aplicação e (5) gera o código de persistência de dados JPA, que juntos resultam no nosso produto final.

Entretanto, diferentes partes podem envolver diferentes papéis. Em outras palavras, existem dois papéis diferentes presentes na abordagem. A Figura 4.6 ilustra a expansão da Figura 4.2, e atribui o papel presente em cada elemento do framework. Os papéis presentes em nossa abordagem são:

a) Desenvolvedor do *Framework*.
b) Desenvolvedor da Aplicação.

O papel referente à área azul da Figura 4.6, o desenvolvedor do *framework* está presente nas etapas de criação do meta-modelo, do *plugin* e da construção do gerador de código. Este papel compreende o mais baixo nível na hierarquia em termos de complexidade de suas ações e é responsável por futuras extensões do framework. Exemplificando, ele é responsável por criar um novo meta-modelo utilizando as tecnologias de persistência de dados *Hibernate* com JPA ou até podendo acrescentar novas entidades e atributos.

O papel referente à área laranja da Figura 4.6, compreende o desenvolvedor usuário do Framework, compreendendo o público alvo. Este papel representa o desenvolvedor que utiliza o framework para desenvolvimento, ou seja, para quem o código de persistência de dados JPA é gerado. Por sua vez, está presente somente na etapa de instância do modelo, que ocorre sob o *plugin* previamente construído, em outras palavras, o usuário do framework apenas cria o modelo da aplicação e o executa, o que resulta na geração do código da aplicação, representado pela área branca da Figura 4.6.

É importante ressaltar que o papel do usuário do framework não está vinculado na criação ou na expansão do meta-modelo, e não está envolvido no projeto do gerador de códigos, tampouco requer o conhecimento necessário para tais. Também, é importante notar que o papel do desenvolvedor do framework está presente neste passo, porém, somente como criador do *plugin* e não como usuário do framework.

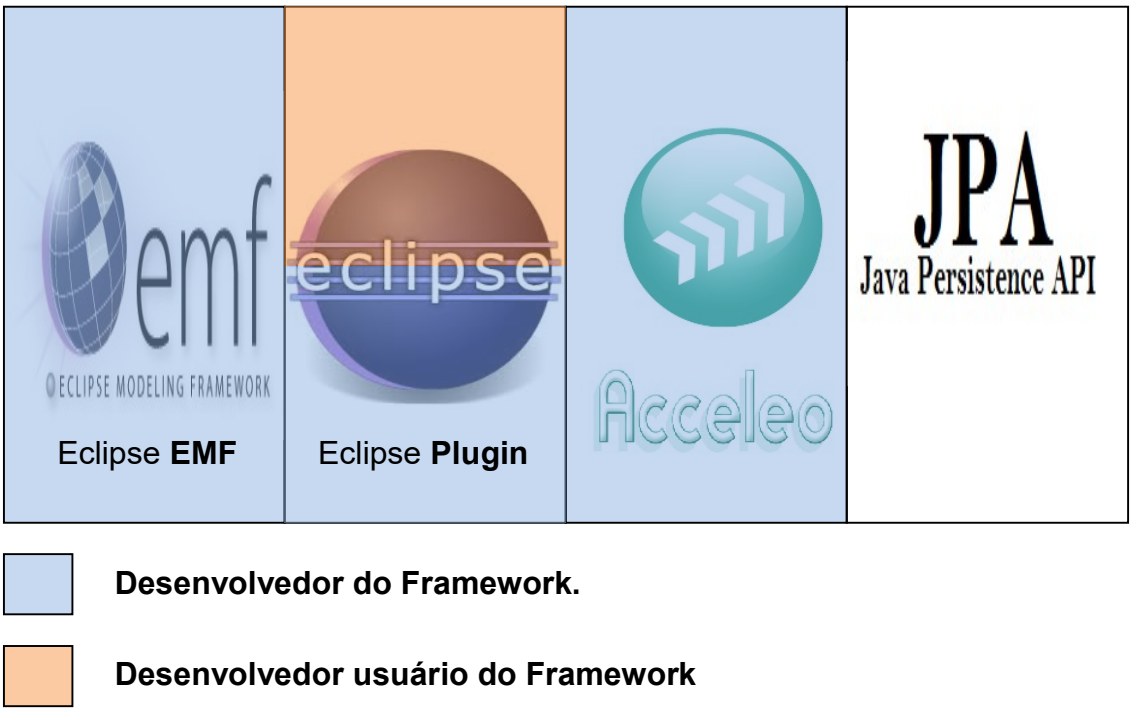


Figura 4.6 - Diferentes papéis presentes no framework proposto.

5. EXPERIMENTO

Neste capítulo é apresentado o método experimental é constituído pelo objetivo do estudo, propósito, com foco na qualidade, perspectiva e contexto. Ainda neste capítulo apresenta-se o planejamento que contém definição das hipóteses, seleção dos participantes, descrição do experimento, descrição da análise e execução do experimento.

5.1 MÉTODO EXPERIMENTAL

O presente experimento é caracterizado como um estudo experimental, que visa coletar dados em um ambiente controlado, para confirmar ou negar uma hipótese. O método experimental considera a proposta e avaliação de um modelo com os estudos experimentais (BASILI, 1996). Experimentação provê uma maneira sistemática, disciplinada, quantificável e controlada para avaliar atividades baseada em humanos. Esta é uma das muitas razões pelo o qual pesquisas empíricas são comuns nas ciências sociais e ciências comportamentais (WOHLIN, 2012).

No entanto, é importante ressaltar que os experimentos não provam nada. Nenhum experimento fornece uma prova com certeza absoluta. Os experimentos verificam uma previsão teórica contra a realidade (BASILI, 1996). Neste experimento, foi utilizado um experimento controlado em um pequeno objeto de estudo. A maior vantagem do experimento é o controle total sobre o processo, além da possibilidade de repeti-lo.

- **Objetivo do Estudo:** Avaliar se a abordagem para persistência de dados JPA baseada em MDD e POA é melhor à implementação de persistência de dados da forma tradicional de desenvolvimento.
- **Propósito:** Propor a implementação de problemas reais e comparar o desenvolvimento utilizando a abordagem baseada em MDD e POA com implementação sem o uso da abordagem.
- **Com foco na qualidade:** Para medir a qualidade do código foram utilizadas as métricas de separação de interesses (Difusão do interesse sobre os componentes – CDC, Difusão do interesse sobre as operações – CDO, Difusão do interesse sobre as linhas de código – CDLOC), as métricas de acoplamento (Acoplamento Eferente – CE, acoplamento Aferente – CA, profundidade da árvore de herança –

DIT), a métrica de coesão (Falta de coesão nas operações – LCOO) e as métricas de tamanho (Linha de código – LOC, Número de atributos – NOA, comprimento das operações por componentes – WOC), tempo.

- **Perspectivas:** A perspectiva é a partir do ponto de vista dos pesquisadores, ou seja, o pesquisador quer identificar se há alguma diferença na produtividade dos programadores que usam as duas formas distintas de implementação e se há diferença na qualidade do código gerado pelas duas formas de implementação.
- **Contexto:** Avaliar as implementações de contratos em um sistema, implementando-os utilizando diferentes técnicas de persistência de dados. Os programadores serão alunos do curso de Ciência da Computação que tenham bom conhecimento de programação orientada a objetos, conheçam as tecnologias de persistência de dados JPA e JDBC. Além disso, devem também conhecer o padrão DAO e saberem utilizar a abordagem baseada em MDD e POA.

5.2 Planejamento

O planejamento do experimento foi dividido em seis etapas. Cada etapa é descrita nas seções seguintes.

5.2.1 Definição das variáveis

Visto que a característica desse experimento é um estudo experimental, de acordo com Rauen (2012), o pesquisador assume hipoteticamente a possibilidade de uma variável ter relação assimétrica com outra variável, ou seja, um fato ou fenômeno é fator de causa para outro fenômeno. Diante disso, foi necessário definir as variáveis dependentes e independentes antes de testar um modelo específico contra as observações dos fenômenos (Pinsonneault; Kraemer, 1993).

5.2.1.1 Variáveis Independentes

Variável independente, X ou VI, é a variável que exerce influência sobre outra variável, ou seja, são os fatores determinantes para que ocorra um determinado resultado, efeito ou consequência (MARCONI; LAKATOS, 2000).

Geralmente, são controladas pelo pesquisador em seus experimentos a fim de procurar estabelecer sua relação e influência sobre o resultado de um fenômeno.

As variáveis independentes definidas neste experimento foram os participantes, as tecnologias de persistência de dados JPA e JDBC, o MDD e POA.

5.2.1.2 Variáveis Dependentes

Variável dependente, Y ou VD, de acordo com Marconi e Lakatos (2000), é a variável a ser explicada ou descoberta, em virtude de ser influenciada, determinada ou afetada pela variável independente. O resultado que a variável apresenta pode variar de acordo com o pesquisador, pois, o mesmo pode colocar, tirar ou modificar a variável independente. Segundo Jung (2009) as variáveis dependentes são aquelas cujo comportamento se quer verificar em função das oscilações das variáveis independentes, ou seja, correspondem àquilo que se deseja prever e/ou obter como resultado.

As variáveis dependentes definidas neste experimento foi o tempo de desenvolvimento de desenvolvimento e a qualidade do código das implementações.

5.2.2 Definição das hipóteses

Para orientar a pesquisa, duas hipóteses foram estabelecidas com base no objetivo principal deste experimento:

Ha0 $\mu_{tradicional} = \mu_{MDD}$, em que não existe diferença de produtividade em relação ao uso da abordagem baseada em MDD com aspecto, em relação à implementação tradicional que utiliza a POO.

Ha1 $\mu_{tradicional} < \mu_{MDD}$, em que, a abordagem baseada em MDD e Aspecto é superior na produtividade em relação ao desenvolvimento tradicional de software.

Hb0 $\mu_{tradicional} = \mu_{MDD}$, em que não existe diferença de qualidade em relação ao uso da abordagem baseada em MDD com aspecto, em relação a implementação tradicional.

Hb1 $\mu_{tradicional} < \mu_{MDD}$, em que, a abordagem baseada em MDD e Aspecto gera um código superior em qualidade em relação ao desenvolvimento tradicional de software.

5.2.3 Seleção de Participantes

Os participantes que colaboraram neste experimento foram estudantes de graduação do curso de Ciência da Computação. Os participantes são indivíduos especialmente selecionados da população de interesse para a realização do experimento, uma vez que o experimento quer simular o comportamento de indivíduos dentro de uma empresa de desenvolvimento de software. Assim, o grupo de participantes tem de ser homogêneo, de modo que todos sejam capazes de realizar as funções propostas em um tempo semelhante.

Portanto, na seleção dos participantes, utilizou-se uma amostragem não probabilística, considerando uma amostragem por conveniência, o qual as pessoas mais próximas e mais convenientes são selecionadas (WOHLIN, 2012).

5.2.4 Descrição do Experimento

Neste experimento, o principal objetivo é avaliar se a abordagem proposta, de gerar códigos de persistência de dados JPA, pode auxiliar o programador a ter melhor desempenho no momento do desenvolvimento do software, por meio de medições realizadas a partir de implementação desenvolvidas pelos participantes.

Os participantes foram solicitados a realizar duas implementações de um software. O software é um sistema para gerenciamento de empréstimos de livros em uma biblioteca, desenvolvido na plataforma Java.

As duas implementações propostas foram aplicações de padrão e tecnologia de persistência de dados. Na primeira, os participantes foram solicitados a implementar uma parte do sistema, a funcionalidade de empréstimo de livros, aplicando o padrão DAO utilizando o *Generic*DAO e a tecnologia JPA. No entanto, esta implementação foi dividida em duas tarefas:

- Implementar a funcionalidade do sistema utilizando a abordagem proposta neste trabalho.
- Implementar a funcionalidade do sistema sem utilizar a abordagem proposta neste trabalho.

Na segunda implementação, os participantes foram solicitados a implementar a mesma funcionalidade e aplicando o mesmo padrão, porém utilizando a tecnologia JDBC. Assim, a segunda implementação foi dividida em duas tarefas:

- Implementar a funcionalidade do sistema utilizando a abordagem de geração de código de persistência de dados JDBC.

- Implementar a funcionalidade do sistema sem utilizar a abordagem de geração de código de persistência de dados JDBC.

5.2.5 Descrição da Análise

Para analisar este experimento, foi utilizada análise quantitativa. Experimentos são quase puramente quantitativos, uma vez que têm foco na contagem de variáveis diferentes, alterá-los e medi-los novamente. Durante estas investigações, os dados quantitativos são recolhidos e, em seguida, são aplicados os seguintes métodos estatísticos: a média, a mediana, o desvio padrão, a distribuição normal e o teste binomial. No entanto, dados qualitativos podem ser coletados para auxiliar a interpretação dos dados (KARAHASANOVI´C, *et. al.*, 2005).

A análise foi realizada através de um conjunto de métricas de separação de interesses, de acoplamento, de coesão e de tamanho (SANT'ANNA *et. al.,* 2003) para avaliar as implementações utilizando abordagem proposta neste trabalho e as que não utilizaram a abordagem proposta. Essas métricas já foram utilizadas em três estudos diferentes (GARCIA *et. al.,* 2005; SOARES, 2004; OLIVEIRA *et. al.,* 2008)e foram aplicadas neste trabalho por considerar as abstrações e o mecanismo da POA. Essas métricas foram definidas com base no reuso e refinamento de algumas métricas clássicas OO (CHIDAMBER; KEMERER, 1994; FENTON;PFLEEGER, 1997), que foram estendidas para o paradigma orientado a aspecto (OLIVEIRA *et. al.,* 2008).

Para realizar a análise, a fim de responder às hipóteses propostas na seção 5.2.1, é apresentado no Quadro 5.1 às métricas utilizadas para avaliar a implementação e na mesma é apresentada uma breve definição de cada métrica aplicada, e as associa com os atributos medidos por cada uma. Neste Quadro, observa-se que as métricas de acoplamento possuem alterações em relação à de Sant'Anna *et. al.* (2003)

Quadro 5-1- Métricas

ATRIBUTOS	MÉTRICAS	DEFINIÇÕES
Separação De Interesses	Difusão do interesse sobre os componentes (CDC)	Conta o número de classes e aspectos que a principal proposta é contribuir com a implementação de um interesse e o número de outras classes e aspectos que os acessam.
	Difusão do interesse sobre as operações (CDO)	Conta o número de métodos e advices que a principal proposta é contribuir com a implementação de um interesse e o número de outros métodos e advices que os acessam.
	Difusão do interesse sobre as linhas de códigos (CDLOC)	Conta o número de pontos de transições para cada interesse por meio de linha de código. Ponto de transições são pontos no código onde há uma troca de interesses.
Acoplamento	Acoplamento Eferente (CE)	Representa a contagem de quantas classes diferentes referem-se à classe atual, por meio de campos ou parâmetros.
	Acoplamento Aferente (CA)	Representa a contagem de quantas classes diferentes a classe atual faz referência, por meio de campos ou parâmetros
	Profundidade da árvore de herança (DIT)	Conta a profundidade da hierarquia de herança em que uma classe ou aspecto é declarado.
Coesão	Falta de coesão nas operações (LCOO)	Mede a falta de coesão de uma classe ou um aspecto em termos da quantidade de pares de métodos e advices que não acessam a mesma variável de instância.
Tamanho	Linhas de código (LOC)	Conta o número de linhas de código.
	Número de atributos (NOA)	Conta o número de atributos de cada classe ou aspecto.
	Comprimento das operações por componentes (WOC)	Conta o número de métodos e advices de cada classe ou aspecto e o número de seus parâmetros.

FONTE: OLIVEIRA *et. al.*, 2008 (Adaptado pelo autor)

5.2.6 Execução do Experimento

Como citado na Seção 5.2.3, cada implementação foi dividida em duas tarefas, resultando em duas implementações e quatro tarefas, denominadas 1a, 1b, 2a e 2b. Para realização de todas as tarefas, os participantes do grupo 1 continham mais conhecimento sobre a abordagem proposta enquanto o grupo 2 estavam em nível de treinamento. Entretanto, ambos tinham a mesma experiência sobre o padrão DAO e as técnicas de persistência de dados utilizadas, JPA e JDBC.

Os quatros participantes selecionados foram divididos em dois grupos, G1 e G2, de modo que foi concebido um experimento equilibrado, com duas equipes de desenvolvimento homogêneas. Portanto, o método experimental utilizado foi um estudo replicado (experimento controlado), com dois tratamentos: participantes

usando a abordagem proposta e participantes não usando a abordagem proposta, todos desenvolvendo a mesma tarefa.

O experimento foi executado duas vezes, cada execução com um grupo que deveria realizar a implementação proposta. No entanto, um integrante do grupo começava a implementar utilizando a abordagem proposta enquanto o outro começava a implementar sem utilizar a abordagem. Durante a execução das tarefas, os participantes tiveram o acesso à Internet para pesquisar materiais que pudesse ajudá-los a resolver as tarefas.

O Quadro 5.2 mostra a ordem de execução das implementações de cada grupo durante o experimento, nas quais as implementações 1a e 2a se refere às implementações utilizando a tecnologia de persistência de dados JPA e JDBC respectivamente usando o meta-modelo enquanto as implementações 1b e 2b utilizam as mesmas tecnologias, porém sem utilizar o meta-modelo.

Quadro 5.2 - Ordem de implementação

GRUPO	INTEGRANTES	ORDEM DE IMPLEMENTAÇÃO			
GRUPO 1	ALUNO 1	1b	1a	2a	2b
	ALUNO 2	1a	1b	2b	2a
GRUPO 2	ALUNO 3	1a	1b	2b	2a
	ALUNO 4	1b	1a	2a	2b

Após a conclusão de cada tarefa, os participantes anotaram em uma tabela o horário de início e do fim da implementação, por fim foi enviado o código-fonte de cada implementação.

De forma a evitar problemas relacionados a confidencialidade do trabalho proposto, foi solicitado aos participantes que assinassem um termo de consentimento e de esclarecimento, encontrado no Apêndice H, no qual consta que os participantes não poderiam repassar os dados a terceiros, e que não seriam divulgados os nomes dos participantes neste trabalho.

No Capítulo 6, são apresentados os resultados obtidos das análises realizadas baseando-se nas métricas definidas no Quadro 5.2.

6. ANÁLISE DOS RESULTADOS

Nesta fase do trabalho, os dados são trabalhados e exibidos graficamente para facilitar a análise dos resultados, a fim de se obter um entendimento melhor sobre a amostra. Sendo assim, os dados obtidos foram submetidos a operações estatísticas, tais como: média, mediana, teste binomial, desvio padrão e distribuição normal.

Juntamente com o trabalho desenvolvido, outro foi realizado para avaliar a abordagem proposta, no qual foi utilizada a tecnologia de persistência de dados *Hibernate*. Dessa maneira, para avaliar se a abordagem proposta, em que soluções que utilizam MDD, podem ser realizadas em um tempo menor que soluções que não utilizam a abordagem, foi-se utilizado os dados gerados neste trabalho. Assim como no presente trabalho, o projeto que utilizou *Hibernate* possui o mesmo experimento apresentado aqui, para confrontar o uso da abordagem em conjunto com a tecnologia *Hibernate* em detrimento da tecnologia JDBC.

Portanto, considerando que os dois trabalhos utilizaram a mesma abordagem para comparar o seu desempenho com aplicações desenvolvidas sem o uso da abordagem, diferenciando apenas a tecnologia utilizada, e os dois foram realizados nas mesmas condições, foi possível utilizar os dados destes projetos para analisar o tempo e qualidade de desenvolvimento com ou sem o uso da abordagem proposta.

Sendo assim, nesta seção, baseada nas categorias e questões definidas na Seção 5.2.4, foram extraídos os resultados, analisando os dados obtidos através da realização do experimento. Abaixo é apresentada a análise e o resultado de cada uma das categorias.

6.1 TEMPO DE DESENVOLVIMENTO

Nesta categoria, apenas uma questão foi elaborada. “Desenvolvedores que utilizam uma abordagem MDD, solucionam as tarefas mais rapidamente do que os que não utilizam?”.

Resultados

Para começar a analisar os dados, é necessário negar a hipótese nula, Ha0 (definida na Seção 5.2.1), isto é, a intenção do experimento é rejeitar a hipótese. (WOHLIN *et. al.*, 2012). De acordo com Wohlin *et. al.* (2012), se a hipótese nula não

for rejeitada, nada pode ser dito sobre o resultado alcançado, no entanto, se a hipótese for rejeitada, pode-se afirmar que a hipótese é falsa.

Sendo assim, foi utilizado o método estatístico não-paramétrico, este método consiste em uma análise de forma mais geral dos dados enquanto o método estatístico paramétrico faz suposições sobre a distribuição dos dados (WHOLIN *et. al*, 2012). A utilização desse método se deu devido ao fato de que foi contado quantas implementações com a utilização da abordagem foram mais rápidas comparado com as implementações sem a abordagem, tratando assim os dados de maneira mais geral.

Para isto, foi realizado o teste binomial para medir o tempo de desenvolvimento de cada implementação e foram classificadas da seguinte maneira: Tradicional (sem utilizar a abordagem) e MDD (utilizando a abordagem). Entretanto, é necessário realizar um teste para verificar se há diferença no tempo de implementação, assim verificar se a hipótese Ha0 é falsa.

Portanto, a hipótese nula pode ser formulada como:

Ha0 : P(tradicional) = P(MDD) = 1/2

Segundo Wholin *et. al.* (2012), é decidido que o *α* deve ser menor que 0,10. A Tabela 4 apresenta os dados obtidos do experimento:

Tabela 6.1 - Comparação das implementações com menor Tempo de implementação (Com abordagem x Sem abordagem)

IMPLEMENTAÇÕES	Número de implementações mais rápidas
Com a abordagem	11
Sem a abordagem	1

Dessa forma, para verificar se a hipótese Ha0 é falsa, é utilizada a seguinte fórmula:

$$\textbf{P(0-1 sem abordagem)} = \sum_{i=0}^{1}\left(\frac{12}{i}\right)\left(\frac{1}{2}\right)^{i}\left(\frac{1}{2}\right)^{12-i} = \frac{1}{2^{12}}\sum_{0}^{1}\left(\frac{12}{i}\right) = 0.003$$

Para fazer o teste binomial foi utilizada a linguagem de programação R, pois, a mesma é software voltado para análise estatística (AQUINO, 2014). Ao realizar o teste binomial, o resultado do α é de 0.003, com isso, pode-se concluir que a hipótese Ha0 é falsa, pois, o $\alpha < 0.10$, indicando o grau de significância da hipótese ser nula. A partir do momento que a hipótese nula é negada, consegue-se analisar o tempo de desenvolvimento das implementações.

O tempo de desenvolvimento para os dois grupos, em média sempre foi menor para as implementações que utilizaram o MDD, como pode ser observado na Figura 6.1 à mediana foi de 67 minutos para as implementações utilizando o MDD, enquanto para as implementações que não utilizaram o MDD, o tempo mediano foi de 93 minutos. As implementações em que não foi utilizado o MDD, o tempo total de desenvolvimento dos dois grupos foi em média de 28% superior em relação às implementações que se utilizou o MDD.

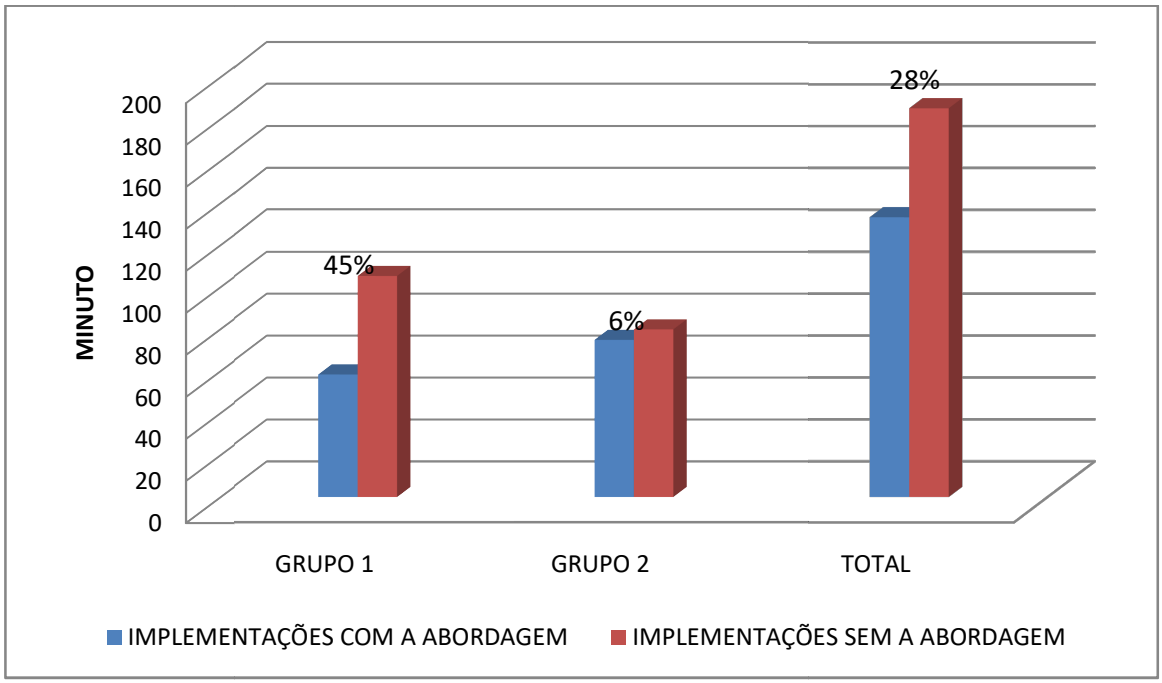


Figura 6.1 - Tempo médio de implementação de cada grupo

Entretanto, verificar somente a média do tempo de implementação não se tem uma precisão dos dados, pois podem existir dados muitos distintos. Portanto, foi utilizado também o desvio padrão como forma de aumentar a precisão da análise Assim, o tempo de desenvolvimento dos dois grupos, para implementações que não utilizam o MDD, de acordo com o desvio padrão, foi 14% superior que a abordagem proposta.

Em conjunto com desvio padrão foi utilizada a distribuição normal, com o objetivo de verificar a dispersão dos dados, assim mostrar o quão os dados estão distribuídos e precisos.

Na Figura 6.2 existem dois gráficos: tempoA (implementações com o MDD) e tempoS (implementações sem o MDD). Nota-se no gráfico tempoA possui uma curva mais fechada, ou seja, os dados estão mais concentrados em relação ao gráfico de tempoS que apresenta uma curva mais aberta.

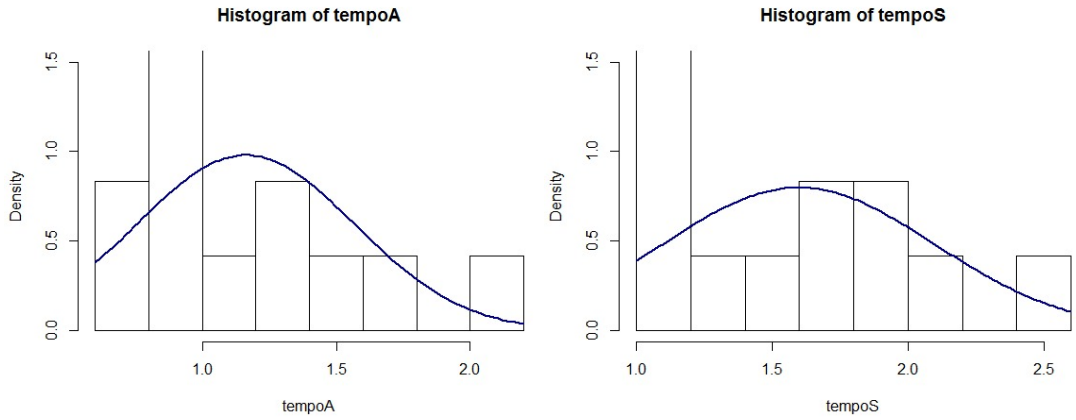


Figura 6.2 - Gráficos de distribuição normal, tempoA x tempoS

Após o tratamento dos dados obtidos através do experimento aplicado, pode-se verificar que o tempo de implementação de persistência de dados se reduz ao utilizar a abordagem proposta, ou seja, o tempo é reduzido em 14%, como apresentado anteriormente. Assim, observando a Figura 6.2, nota-se que em uma implementação sem a utilização de MDD levaria aproximadamente de 80 minutos à 140 minutos, enquanto que com a abordagem o tempo seria de 30 minutos à 110 minutos.

6.2 QUALIDADE DA IMPLEMENTAÇÃO

Nesta categoria, apenas uma questão foi elaborada: “A utilização da abordagem proposta, auxilia a melhorar a qualidade das implementações nas tarefas propostas?”.

Resultados

Para começar a analisar os dados, é necessário negar a hipótese nula, Hb0 (definida na Seção 5.2.1), isto é, a intenção do experimento é rejeitar a hipótese. (WOHLIN *et. al.*, 2012). De acordo com Wohlin *et. al.* (2012), se a hipótese nula não for rejeitada, nada pode ser dito sobre o resultado alcançado, no entanto, se a hipótese for rejeitada, pode-se afirmar que a hipótese é falsa.

Sendo assim, foi utilizado o método estatístico não-paramétrico, este método consiste em uma análise de forma mais geral dos dados enquanto o método estatístico paramétrico faz suposições sobre a distribuição dos dados (WHOLIN *et. al.*, 2012). A utilização desse método se deu devido ao fato de que foi contado

quantas implementações que utilizaram a abordagem tiveram melhor qualidade em seu código comparado com as implementações sem a abordagem, tratando assim os dados de maneira mais geral, pois, a quantidade de participantes.

Para isto, é realizado o teste binomial para medir a qualidade de cada implementação e foram classificadas da seguinte maneira: Tradicional (sem utilizar a abordagem) e MDD (utilizando a abordagem). Entretanto, é necessário realizar um teste para verificar se há diferença na qualidade da implementação, assim verificar se a hipótese Hb0 é falsa.

Portanto, a hipótese nula pode ser formulada como:

Hb0 : P(tradicional) = P(MDD) = ½

Segundo Wholin *et. al.* (2012), é decidido que o *α* deve ser menor que 0,10. A Tabela 4 apresenta os dados obtidos do experimento

Tabela 6.2 – Comparação da qualidade do código das implementações (Com abordagem x Sem abordagem)

IMPLEMENTAÇÕES	Número de implementações mais rápidas
Com a abordagem	70
Sem a abordagem	50

Dessa forma, para verificar se a hipótese Hb0 é falsa, é utilizada a seguinte fórmula:

$$\textbf{P(0-1 sem abordagem)} = \sum_{i=0}^{50}\left(\frac{120}{i}\right)\left(\frac{1}{2}\right)^{i}\left(\frac{1}{2}\right)^{120-i} = \frac{1}{2^{120}}\sum_{0}^{50}\left(\frac{120}{i}\right) = 0.041$$

Para fazer o teste binomial foi utilizada a linguagem de programação R, pois, a mesma é software voltado para análise estatística (AQUINO, 2014). Ao realizar o teste binomial, o resultado do α é de 0.041, com isso, pode-se concluir que a hipótese Hb0 é falsa, pois, o $\alpha < 0.10$, indicando o grau de significância da hipótese ser nula. A partir do momento que a hipótese nula é negada, consegue-se analisar a qualidade das implementações.

A comparação foi dividida em duas partes. A Seção 6.2.1 apresenta a análise da separação de interesses, da persistência de dados, relacionadas com as implementações com a abordagem e sem a abordagem. Enquanto a Seção 6.2.2 apresenta os resultados das métricas de acoplamento, coesão e tamanho

O eixo Y do gráfico apresenta os valores absolutos obtidos pela aplicação das métricas. A cada par de barras, corresponde os valores obtidos de cada

implementação, junto a cada barra é apresentado à diferença, em porcentagem, dos resultados obtidos das implementações com e sem o MDD.

6.2.1 SEPARAÇÃO DE INTERESSES

Observa-se na Figura 6.3, que todas as implementações utilizando a abordagem obteve uma melhor qualidade em relação à separação de interesses das classes que implementam a persistência de dados.

Sendo assim, isto aconteceu devido ao aspecto separar os interesses do sistema. Portanto, como apresentado na Figura 6.3, o CDC (difusão do interesse sobre os componentes) teve uma melhora de 83%. Enquanto que no CDO (difusão de interesse sobre as operações) teve uma melhora de 86%. E na última métrica, a difusão de interesse sobre as linhas de códigos (CDLOC), obteve-se uma melhora de 91%.

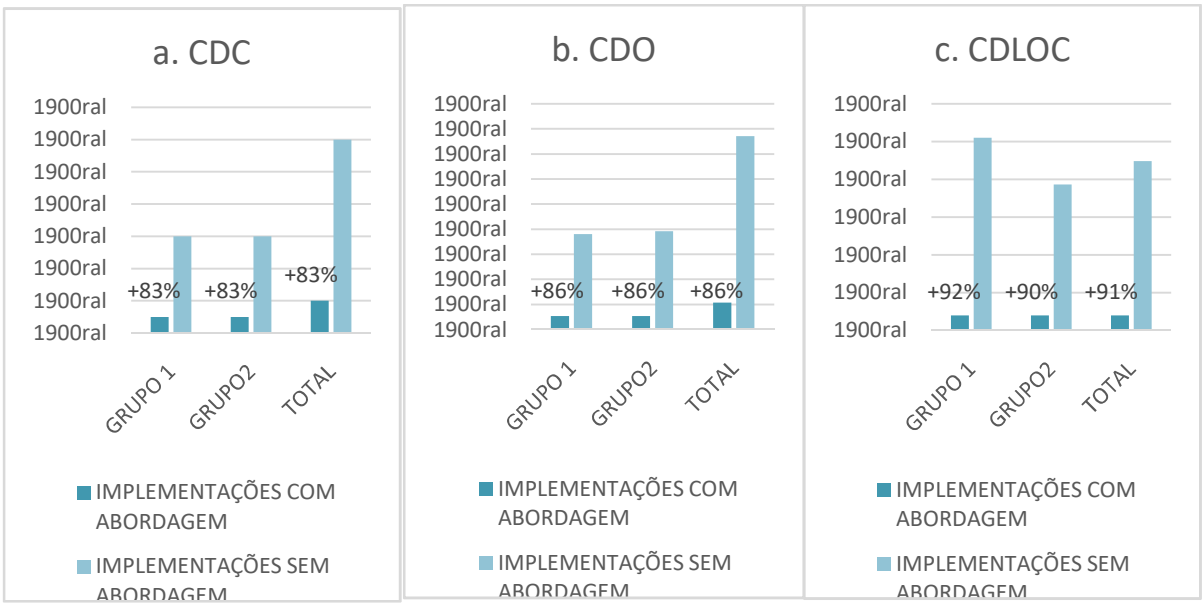


Figura 6.3 – **Separação de Interesses**

6.2.2 ACOPLAMENTO, COESÃO E TAMANHO

Na Figura 6.4, são apresentadas três métricas, a profundidade da árvore de herança (DIT), o acoplamento eferente (CE) e o acoplamento aferente (CA). Nota-se que no DIT, não houve diferença entre as duas abordagens.

No entanto, os resultados obtidos nas métricas de CE e CA, mostram que as implementações utilizando a abordagem, tiveram uma redução de acoplamento de 9% e 3% respectivamente.

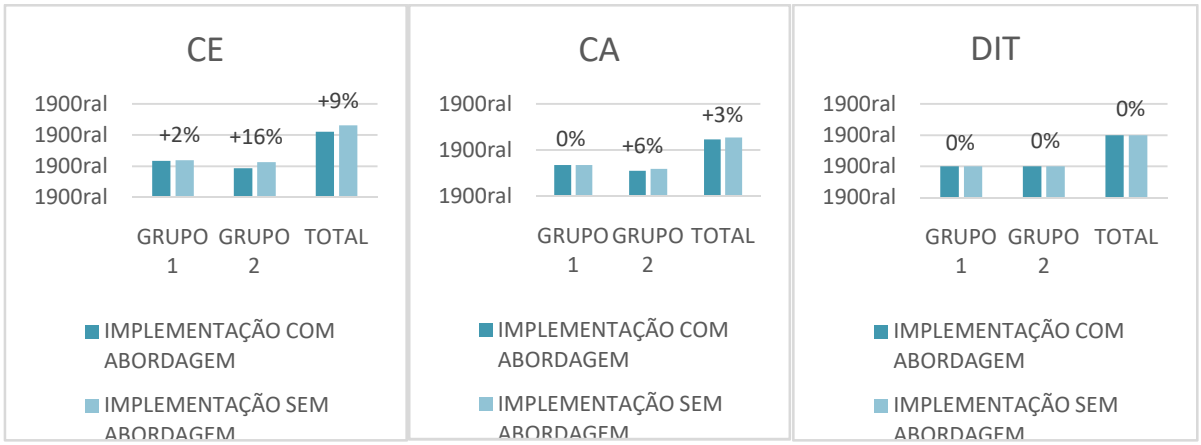


Figura 6.4 - Acoplamento

Em relação à métrica de falta de coesão nas operações (LCOO), as implementações, com a abordagem proposta, de ambos os grupos, teve-se uma pequena diferença de 4%, como é ilustrado na Figura 6.5.

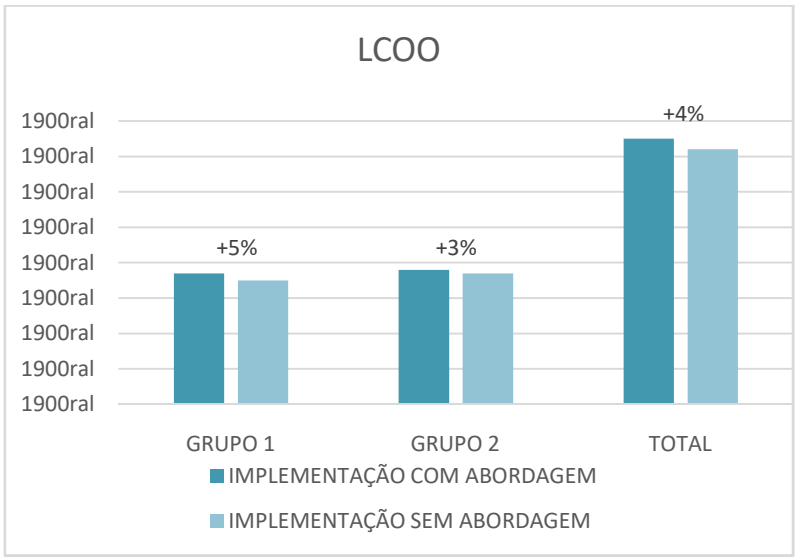


Figura 6.5 - Falta de coesão das implementações

Nas últimas métricas, referente ao tamanho do código é analisado os seguintes dados: linha de códigos (LOC), número de atributos (NOA) e comprimento das operações por componente (WOC).

Como ilustrado na Figura 6.6, a diferença de LOC foi de 4%, onde favoreceu a abordagem proposta. Em relação à métrica NOA, a implementação com abordagem foi 2% melhor que sem a abordagem. Enquanto na WOC, observa-se que houve uma variação de 3%.

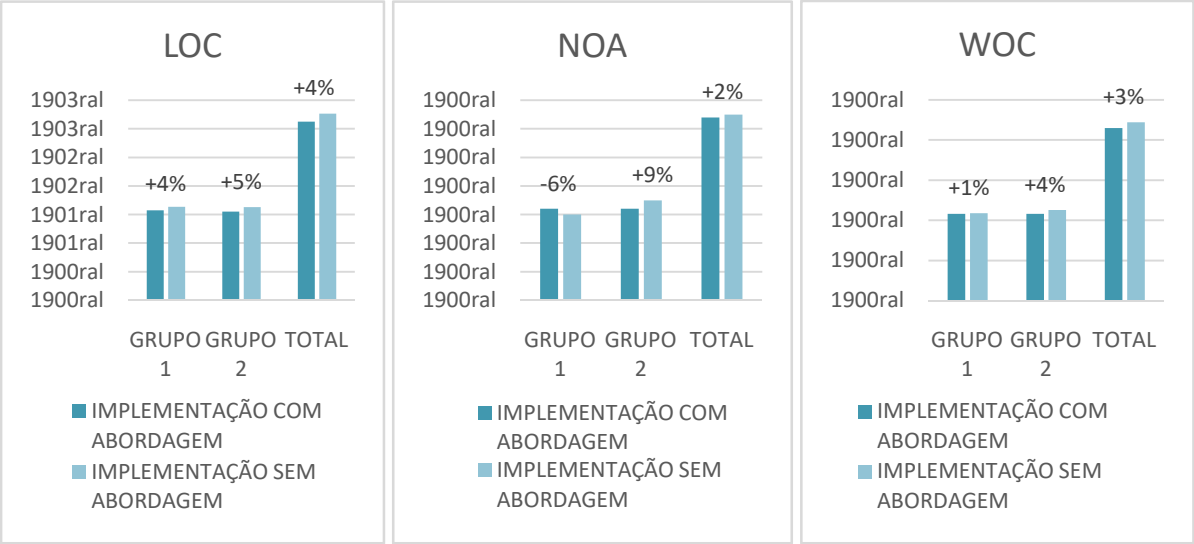


Figura 6.6 - Tamanho das implementações

6.2.3 DISCUSSÃO

A análise dos resultados obtidos indica que a qualidade na separação de interesse de cada implementação por grupo que utilizaram a abordagem, foi melhor do que as implementações que não utilizaram. Assim, a abordagem se mostra eficiente, principalmente quando se trabalha com o padrão DAO e as tecnologias de persistência de dados, ou seja, diminuindo os interesses transversais de uma classe.

Consequentemente, os resultados das métricas de acoplamento, CE e CA, reforçam a qualidade na separação de interesse, pois, como seus interesses estão em um aspecto, às classes que implementam DAO não precisam persistir os dados como é feito na OO, deixando que o aspecto realize a persistência baseado na sua tecnologia. Entretanto, a métrica DIT, não apresenta nenhuma alteração, visto que, nas implementações não houve heranças.

Além disso, a métrica de coesão também demonstra um resultado positivo, devido ao fato dos resultados anteriores serem satisfatório, isso faz com que a manutenção, reuso e o entendimento do código de persistência de dados fiquem mais fácil e claro para os programadores.

Da mesma forma, observa-se que as métricas de tamanho, LOC, NOA e WOC, mostram resultados positivos. Com a utilização do aspecto, o tamanho do projeto consegue ser reduzido consideravelmente, pois, não há necessidade de repetir o código de persistência de dados, uma vez que, já estão implementados no aspecto.

Com isso, os resultados obtidos na Seção 6.2.2 indicam que a qualidade do código das implementações que utilizaram a abordagem foram melhores do que as que não utilizaram, visto que, todas as métricas analisadas obtiveram resultados positivos. Assim, a hipótese Hb1 é verdadeira.

6.3 JPAX JDBC

Para avaliar a diferença entre as tecnologias JPA e JDBC não foi proposta uma hipótese, mas uma questão de pesquisa: “Existe diferença no tempo de implementação e na qualidade de código de persistência de dados JPA em relação à tecnologia JDBC, com ou sem a abordagem?”

Resultados

Na Figura 6.7, apresenta-se um gráfico com a comparação entre o tempo de implementação JPA e JDBC realizado no experimento. Nota-se que o tempo de desenvolvimento das implementações JPA, utilizando ou não a abordagem proposta no trabalho, foi mais rápido. O JPA foi 37% mais rápido que o JDBC referente ao tempo de desenvolvimento.

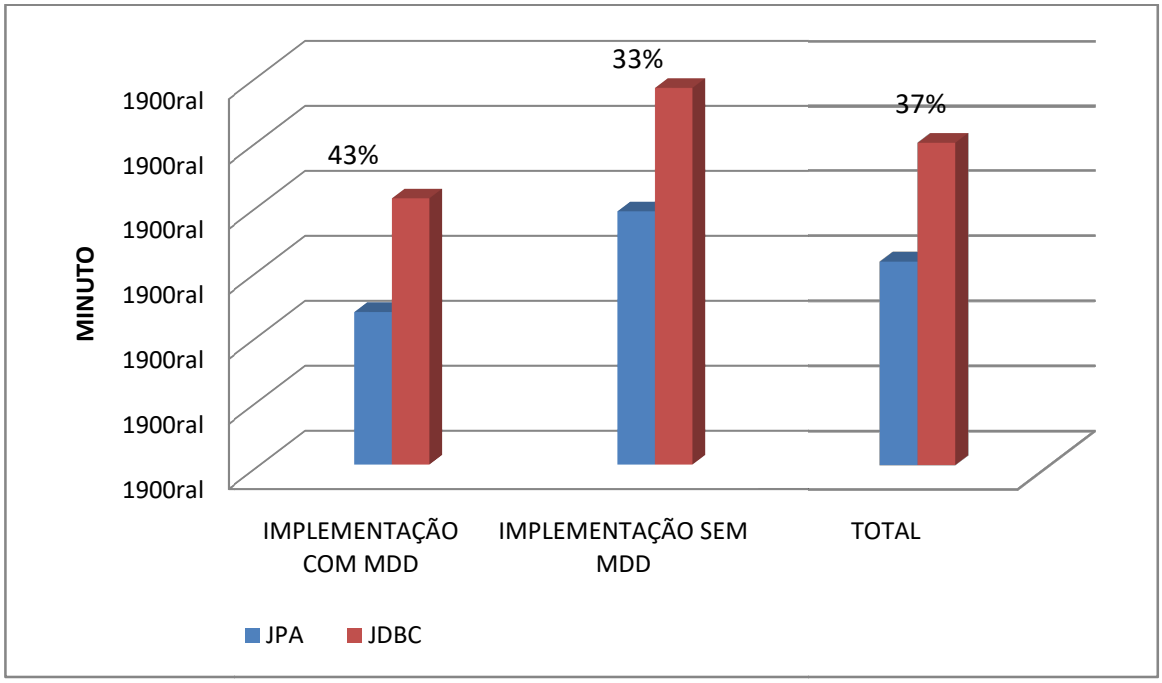


Figura 6.7 - Tempo de implementação (JPA x JDBC)

A Tabela 6.8 apresenta a comparação entre JPA e JDBC, onde as métricas foram separadas por implementações com a abordagem e sem a abordagem proposta. Portanto, para analisar os dados de forma mais simples, é utilizado

técnicas de estatísticas, média, mediana e desvio padrão, para unir os dados dos quatro participantes.

Tabela 6.3 - Comparação das métricas de qualidade (JPA x JDBC)

		COM A ABORDAGEM		SEM A ABORDAGEM	
		JPA	JDBC	JPA	JDBC
CDC	Média	1	1	6	6
	Mediana	1	1	6	6
	Desvio Padrão	0	0	0	0
CDO	Média	6	4	38,25	38,5
	Mediana	6	4	38	39
	Desvio Padrão	0	0	2,061553	1
CDLOC	Média	2	2	21,5	22,75
	Mediana	2	2	22,5	25
	Desvio Padrão	0	0	2,380476	6,130525
CE	Média	1,9375	2,1125	2,25	2,25
	Mediana	1,875	2,175	2,25	2,25
	Desvio Padrão	0,349702	0,572094	0,645497	0,73598
CA	Média	3,0625	3,0625	3,0625	3,375
	Mediana	3,125	3,125	3	3,125
	Desvio Padrão	0,349702	0,505594	0,314576	0,595119
DIT	Média	1	1	1	1
	Mediana	1	1	1	1
	Desvio Padrão	0	0	0	0
LCOO	Média	0,38	0,39	0,37375	0,35725
	Mediana	0,3725	0,3915	0,371	0,3405
	Desvio Padrão	0,017645	0,024993	0,027548	0,035929
LOC	Média	584,75	716,75	634,25	681,75
	Mediana	581	686	643	647,5
	Desvio Padrão	59,35416	135,6156	68,68467	120,9197
NOA	Média	31	35,75	31	32,25
	Mediana	30	35,5	30	32
	Desvio Padrão	2,708013	0,957427	2,708013	3,304038
WOC	Média	95,25	124,5	102,5	116,75
	Mediana	95,5	118,5	102,5	112,5
	Desvio Padrão	4,924429	17,59735	13,27906	18,06239

Após separar os dados de forma a torná-los mais compreensíveis, é verificado através das métricas de separação de interesse:

- **CDC:** Referente à métrica de separação de interesse, não houve diferença entre JPA e JDBC, tanto nas implementações com a abordagem quanto as implementações sem a abordagem.

- **CDO:** O JPA utilizando a abordagem é pior que o JDBC utilizando a abordagem, porém, em comparação sem utilizar a abordagem, o JPA é melhor do que o JDBC.
- **CDLOC:** Tanto nas implementações JPA e JDBC que utilizaram a abordagem, não houve diferença. Contudo, o JPA é superior ao JDBC sem utilizar a abordagem.

Em relação às métricas de acoplamento e coesão, nota-se que:

- **CE:** As implementações JPA com a abordagem é superior ao JDBC com a abordagem, no entanto, sem a abordagem não houve diferença.
- **CA:** As implementações JPA sem a abordagem é superior ao JDBC, porém, com a abordagem não houve diferença.
- **DIT:** Não houve diferença entre JPA e JDBC, tanto nas implementações com a abordagem quanto as implementações sem a abordagem.
- **LCOO:** Referente à métrica de coesão, o JPA utilizando a abordagem é inferior ao JDBC utilizando a abordagem, entretanto, na sem a abordagem o JPA é superior.

Nas métricas de tamanho, observa-se que:

- **LOC:** Referente à métrica de tamanho, o JPA com e sem a abordagem é superior ao JDBC.
- **NOA:** As implementações JPA com e sem a abordagem são superiores ao JDBC.
- **WOC:** As implementações JPA com e sem a abordagem são superiores ao JDBC.

Analisando os dados apresentados, nota-se que os resultados obtidos das implementações JPA utilizando a abordagem proposta é superior na maioria das métricas analisadas em relação ao JDBC, contudo, na métrica LCOO foi inferior. Enquanto as implementações JPA sem utilizar a abordagem foi superior em todas as métricas.

Assim, ao concluir a analise dos dados referente à qualidade do código e do tempo de desenvolvimento, o JPA é a melhor solução para se implementar a persistência de dados em comparação a tecnologia JDBC, utilizando ou não a

abordagem proposta Contudo, utilizando a abordagem proposta os programadores conseguiram implementar de maneira mais eficiente e rápida o seu código em relação a implementação tradicional.

7. CONSIDERAÇÕES FINAIS

Após apresentar os resultados obtidos com esta pesquisa, este capitulo posiciona a sua relevância, as contribuições, limitações e perspectivas de trabalhos futuros.

7.1 Relevância do estudo

No desenvolvimento de software, aplicações necessitam que as informações sejam armazenadas de forma persistente. Todavia, como há um grande número de métodos e ferramentas para este fim, a diversidade de opções não é apropriada, pois, na maioria das vezes transfere a responsabilidade pela escolha do método de persistência de dados aos desenvolvedores e em muitos casos este não está apto para fazer a melhor escolha.

Na literatura existem diversos estudos relacionados a este tema, em que tentam definir quais técnicas e ferramentas são apropriadas para cada tipo de situação. Os estudos existentes apenas descrevem em quais casos diferentes técnicas ou métodos são mais adequados. Entretanto, a escolha da técnica ou métodos é apenas um passo para implementação da persistência de dados. Sendo assim, uma alternativa pare resolver esta adversidade é a utilização do MDD.

O MDD auxilia a padronização e a utilização de códigos, pois a implementação dos códigos devem adotar um modelo de alto nível, assim, facilitando a manutenção de software, minimização de erros, além de agilizar o processo de entrega do produto final.

Sendo assim, o presente estudo visa propor uma abordagem que utiliza o MDD, pois, com sua utilização consegue-se evitar problemas recorrentes às aplicações de persistência de dados, além de mitigar contrapontos comuns no desenvolvimento de software.

Outra abordagem da engenharia de software, que pode estar correlacionada no desenvolvimento de aplicações de persistência de dados é a POA. A POA aumenta a modularidade, pois separa o código que implementam funções específicas, afetando diferentes partes do sistema (SOARES *et. al.*, 2002).

Com o exposto, é demonstrado que o estudo corrente contribui no assunto em questão, de forma que ao utilizar a abordagem proposta a abordagem proposta

aumenta a produtividade dos programadores e qualidade dos códigos gerados por eles.

7.2 Contribuições da Pesquisa

A principal contribuição desta pesquisa é o próprio objetivo estabelecido no começo do trabalho, desenvolver uma abordagem para geração de códigos de persistência de dados JPA baseado em MDD e POA. Para tanto, foi definido uma arquitetura para esta abordagem, detalhando suas camadas e seus componentes. Posteriormente, foi desenvolvida a abordagem de acordo com a sua arquitetura. Considerando o experimento realizado, verificou-se a eficácia desta abordagem baseado no tempo de desenvolvimento e na qualidade da codificação. Por fim, pôde-se observar que a abordagem proposta mitiga problemas encontrados na qualidade do código e o tempo de desenvolvimento de aplicações de persistência de dados JPA.

7.3 Limitações

A principal limitação da pesquisa está centrada no uso do framework proposto, onde os participantes tiveram dificuldades em realizar os procedimentos necessários para gerar o modelo do sistema e quando conseguiam gerar, encontravam dificuldades em utilizar o *Acceleo* para gerar os códigos. Com isto, os participantes perderam muito tempo para se adequar ao uso do framework.

7.4 Trabalhos futuros

Por fim, são identificadas e expostas lacunas que o presente trabalho não conseguiu suprir, de forma que possam ser concernidas em trabalhos futuros.

O primeiro trabalho identificado é a implementação de uma ferramenta que auxilie no processo de criação do modelo e geração dos códigos de persistência. Além do programador poder escolher a tecnologia e o padrão de persistência que ele pretende usar. Desta forma, a ferramenta terá já armazenado os meta-modelos com seus respectivos *templates* de geração de códigos de persistência.

O segundo trabalho identificado é a criação de um framework juntando as tecnologias JPA e *Hibernate*, visto que ambas conseguem trabalhar em conjunto da maneira tradicional, ou seja, sem utilizar uma abordagem semelhante a este trabalho.

REFERÊNCIAS

ACCELEO 2016a. *The Eclipse Foundation, 2016.* Disponível em: <http://www.acceleo.org/pages/first-generator-module/e>. Acesso em: 04/07/2016.

ACCELEO 2016b. *The Eclipse Foundation*, 2016. Disponível em: <https://www.eclipse.org/acceleo/>. Acesso em 04/07/2016.

AQUINO, J. A., "*R para cientistas sociais*", 2014.

ATKINSON, C.; KÜHNE, T. *"Model-Driven Development: A Metamodeling Foundation*", In: IEEE Software, vol. 20, ed. 5, 2003.

BARCIA, R.; HAMBRICK, G.; BROWN, K.; PETERSON, R.; BHOGAL, K. S. "*Persistence in the Enterprise: A Guide to Persistence Technologies*", 1ª Edição, IBM Press, 2008.

BARDIN, L. Análise de Conteúdo. Lisboa, Portugal; Edições 70, LDA, 2009.

BASILI V. *"The Role of Experimentation in Software Engineering: Past, Present, Future***"**. IN : PROC. OF THE 18TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1(2), 1996, p. 133-164

CHAVEZ, C.; GARCIA A.; KULESZA, U., "*Desenvolvimento Orientado a Aspectos*". Anais do XVII Simpósio Brasileiro de Engenharia de Software, Manaus: Universidade Federal do Amazonas, 2003.

CHIDAMBER, S.; KEMERER, C.,"*A Metrics Suite for OO Design*". *IEEE Trans. on Soft.Eng.,20-6,* June1994, 476-493. edition, 1997.

FENTON, N.; PFLEEGER, S.,"*Software Metrics: A Rigorous Practical Approach*." London: PWS, 1997.

FERNANDES, J. E.; MACHADO, R. J.; CARVALHO, J. A. "*Model-Driven Software Development for Pervasive Information Systems Implementation*", In: Proceedings of

the 6th International Conference on the Quality of Information and Communications Technology (QUATIC), p. 218-222, 2007.

GARCIA, A.; SANT'ANNA, C.; FIGUEIREDO, E.; KULESZA, U. "*Modularizing Design Patterns with Aspects: A Quantitative Study"*. International Conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA, 14-18 March 2005. ACM Press. P. 3-14.

GUEDES, G. T. A., "*Um metamodelo UML para modelagem de requisitos em projetos de sistemas multiagentes".* 2012. 229 f. Tese (Doutorado em Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

HAILPERN, B; TARR, P. "*Model-driven development: The good, the bad, and the ugly*". IBM Systems Journal.Riverton, NJ, USA, v. 45, n. 3, p. 451-461, 2006.

HEITKÖTTER, H.; MAJCHRZAK, T. A.; KUCHEN, H. "*Cross-Platform Model- Driven Development of Mobile Applications with MD2*". In: 28th Annual ACM Symposium on Applied Computing. 2013. New York.

JIANG, K.; ZHANG, L.; MIYAKE, S. "*Using OCL in executable UML*".Easst, Beijing, v. 9, p. 1-12, 2007.

JUNG, C. F., "*Metodologia científica e tecnológica: módulo 3 – variáveis e constants*". Campinas, 2009. Disponível em: <www.dsce.fee.unicamp.br/~antenor/mod3.pdf>. Acesso em: 18 de dezembro de 2016.

KARAHASANOVI´C, A., ANDA, B., ARISHOLM, E., HOVE, S.E., JØRGENSEN, M., SJØBERG, D., WELLAND, R.: "*Collecting feedback during software engineering experiments.*"Empir.Softw.Eng. 10(2), 113–147 (2005). doi: 10.1007/s10664-004-6189-4. URL http://www.springerlink.com/ index/10.1007/s10664-004-6189-4

KEITH, M.; SCHINCARIOL, M. "*Pro JPA 2: Mastering the Java™ Persistence API*", 2009

KICZALES, G. et al. “*Aspect -Oriented Programming. European Conference on Object - Oriented Programming (ECOOP)*”, LNCS (1241), Springer-Verlag, Finland, 1997.

KOCH, N. “*Classification of model transformation techniques used in UML based Web engineering.*”Software, IET. v. 1, n. 3, p. 98-111, 2012.

KOLOVOS, D. S.; ROSE, L. M.; ABID, S. B.; PAIGE, R. F.; POLACK, F. A. C.; BOTTERWECK, G. 2010, “*Taming EMF and GMF Using Model Transformation*”, in DC Petriu, N Rouquette, Ø Haugen (eds.), Model Driven Engineering Lan- guages and Systems, 1st edn, Springer Berlin Heidelberg, Oslo, Østlandet, Norway.

MAFRA, S. N; TRAVASSOS, G. H. “*Estudos Primários e Secundários apoiando a busca por Evidência em Engenharia de Software*”. Programa de Engenharia de Sistemas e Computação COPPE/UFRJ, 2006.

MARCONI, M. A., LAKATOS, E. M., “*Metodologia científica: ciência e conhecimento científico; métodos científicos; teoria, hipóteses e variáveis; metodologia jurídica*”. 3. ed. rev. ampl. São Paulo: Atlas, 2000.

MASSONI, T.; ALVES, V. and SOARES, S. PDC: “*Persistent Data Collections pattern. First Latin American.*”Conference on Pattern Languages of Programming*,* Rio de Janeiro, Brazil, 3th-5th October 2001.

MELLOR, S.; CLARK, A.; FUTAGAMI, T. “*Model-driven development – Guest editor's introduction*”, In: IEEE Software. vol. 20, n. 5, 2003.

MEYER, B., “*Object–Oriented Software Construction*”. Prentice–Hall, ed. 2, 1997.

MIZUNO, T.; MATSUMOTO, K.; MORI, N., “*A Model-Driven Development Method for Management Information Systems*”, *Eletronics and Communications in Japan*, 96, 2, 245-252. 2010.

MTSWENI, J. “*Exploiting UML and Acceleo for Developing Semantic Web Services*.” In: The 7th International Conference for Internet Technology and Secured Transactions (ICITST-2012). Pretoria, South Africa, 2012.

NELLAIAPPAN, R., “*An Eclipse-based tool for modeling service based systems*”. 2009. 49 f. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – Amrita School of Engineering, Coimbatore, Tamil Nadu, India, 2009.

NOCK, C. “*Data Access Patterns: Database Interactions in Object-Oriented Applications*”. Addison-Wesley, 2003.

OLIVEIRA, A. L.; MENOLLI, A.; COELHO, R. G. “*Separating Data Access Crosscuting Concerns Using AspectJ, A Quantitative Assessment*.” In: IASTED International Conference on Software Engineering, 2008, Innsbruck. Proceedings of International Conference on Software Engineering, 2008

OLIVEIRA, A. L; MENOLI, A. L. A; COELHO, R. G.; CAMARGO, V. V.; RAMOS, R. A.; PENTEADO, R. A. D., “*Um Estudo Quantitativo das Implementações Orientadas a Aspectos das Estratégias do Padrão Data Access Object*”. II Latin American Workshop on Aspect-Oriented Software Development, 2008.

PINSONNEAULT A., KRAEMER, K., “*Survey research methodology in management information systems*”. An assessment. J Manag Inf Syst 10(2):75–105, 1993.

POTHU, S. “*A Comparative Analysis of Object-relational mappings for Java*”, MSc. Thesis, University of Applied Sciences at Braunschweig/Wolfenbuettel, 2008.

RAUEN, F. J. “Pesquisa científica: discutindo a questão das variáveis”, 2012.

SANT’ANNA, C.; GARCIA, A.; CHAVEZ, C.; LUCENA, C.; STAA, A. von. *On the Reuse and Maintenance of Aspect-Oriented Software*: An Assessment Framework. Proc. of Brazilian Symposium on Software Engineering (SBES’03), Manaus, Brazil, Oct 2003, 19-34.

SELIC, B., "*The pragmatics of model-driven development.IBM Rational Software*". Vol. 20, n. 5, p. 19-25, 2008.

SINGH, Y.; SOOD, M. Model-driven Architecture: A perspective in *2009 IEEE International Advance Computing Conferencce*, Patiala, India, 6-7. 2009.

SOARES, S. "*An Aspect-Oriented Implementation Method*". Doctoral Thesis, Federal Univ. of Pernambuco, Oct 2004.

SOARES, S. *et. al.* "*Implementing distribution and persistence aspects with AspectJ*".ACM.In Proceedings of the OOPSLA' 2002, pp. 174-190, Seattle, USA, 2002.

SOARES, S.; BORBA, P. "*AspectJ — Programação orientada a aspecto sem Java*", Brazilian Syposium of Language Programming, SBLP, Rio de Janeiro, 2002.

STAHL, T.; VOELTER, M. "*Model-Driven Software Development: Tecnology, Engineering, Management*", Wiley, West Sussex. 2016.

STEINER, D.; TURLEA, C.; CULEA, C.; SELINGER, S. "*Model-Driven Development of Cloud-Connected Mobile Applications Using DSLs with Xtext*". In: Computer Aided Systems Theory - EUROCAST 2013:14th International Conference, Las Palmas de Gran Canaria, Spain, February10-15, 2013. Revised Selected Papers, Part II.Springer Berlin Heidelberg, v. 8112, p. 409-416, 2013.

SUN MICROSYSTEMS. Core J2EE Patterns: Data Access Object. 2002. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

VIDYAPEETHAM, A. V. "*An eclipse-based tool for modeling service-based systems*".2009.49 f. Trabalho de Conclusão de Curso (Graduação em Tecnologia da Informação) – *Amrita School of Engineering*, Coimbatore, Tamil Nadu, India, 2009.

WINCK, D. V.; GOTTEN JUNIOR, V. "*AspectJ: Programação Orientada a Aspectos em Java*". 1ª São Paulo: Novatec, 2006.

WOHLIN C, RUNESON P, HÖST M, OHLSSON MC, REGNELL B, WESSLÉN A., "*Experimentation in software engineering*". Springer, ISBN 978-3-642-29043-5, 2012.

YODER, J. W.; JOHNSON, R. E.; WILSON, Q. D. “*Connecting Business Objects to Relational Databases.*”Conference on the Pattern Languages of Programs, 1998.

APÊNDICE A –ECORE/XML DO META-MODELO

```
<?xml version="1.0" encoding="UTF-8"?>
<genmodel:GenModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/tcc.dao.jpa.poa.metamodel/src" modelPluginID="tcc.dao.jpa.poa.metamodel"
    modelName="GenericJpaDaoPoa" rootExtendsClass="org.eclipse.emf.ecore.impl.MinimalEObjectImpl$Container"
    importerID="org.eclipse.emf.importer.ecore" complianceLevel="7.0" copyrightFields="false"
    operationReflection="true" importOrganizing="true">
  <foreignModel>genericJpaDaoPoa.ecore</foreignModel>
  <genPackages prefix="Genericjpadaopoa" disposableProviderFactory="true" ecorePackage="genericJpaDaoPoa.ecore#/">
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//Init">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Init/className"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Init/package"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//Init/aspect"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//Init/persistence"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//Init/controle"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//Controle">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Controle/className"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Controle/package"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//Controle/controleGenericoObjeto"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//Controle/insert"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//Controle/delete"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//Controle/update"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//AspectDAO">
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//AspectDAO/getEntityManager"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//Persistence">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Persistence/url"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Persistence/driver"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Persistence/user"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Persistence/password"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//Objeto">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Objeto/className"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Objeto/package"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Objeto/id"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Objeto/type"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//Objeto/atributo"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//Atributos">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Atributos/name"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//Atributos/type"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//GenericDAO">
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//GenericDAO/className"/>
      <genFeatures createChild="false" ecoreFeature="ecore:EAttribute genericJpaDaoPoa.ecore#//GenericDAO/package"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//GenericDAO/insert"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//GenericDAO/delete"/>
      <genOperations ecoreOperation="genericJpaDaoPoa.ecore#//GenericDAO/update"/>
    </genClasses>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//DaoClass"/>
    <genClasses ecoreClass="genericJpaDaoPoa.ecore#//ControleGenericObjeto">
      <genFeatures children="true" createChild="true" propertyDescription="" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//ControleGenericObjeto/GenericDAO"/>
      <genFeatures children="true" createChild="true" ecoreFeature="ecore:EReference genericJpaDaoPoa.ecore#//ControleGenericObjeto/Objeto"/>
    </genClasses>
  </genPackages>
</genmodel:GenModel>
```

APÊNDICE B– *TEMPLATE ACCELEO* PARA GERAÇÃO DE CÓDIGO

Generate.mtl

```
1  [comment encoding = UTF-8/]
2  [module generate('http://genericjpadaopoa/1.0')]
3
4
5  [template public generateElement(anInit : Init)]
6  [comment @main/]
7  [file (anInit._package.replaceAll('\\.', '/').concat('/').
8          concat(anInit.className).concat('.java'), false, 'UTF-8')]
9  package [anInit._package/];
10
11 import [anInit.controle._package/].[anInit.controle.className/];
12
13 public class [anInit.className /] {
14     private static [anInit.controle.className /] control;
15
16     public static void main(String['['/]']'/] args) throws Exception {
17         // performs anything beautifully
18     }
19 }
20 [/file]
21
22 [comment Controles /]
23 [for (controle : Controle | anInit.controle)]
24 [file (controle._package.replaceAll('\\.', '/').concat('/').
25         concat(controle.className.toUpperFirst()).concat('.java'), false, 'UTF-8')]
26 package [controle._package /];
27
28 import java.sql.SQLException;
29 [for (controleGenericObjeto : ControleGenericObjeto | controle.controleGenericoObjeto)]
30 import [controleGenericObjeto.Objeto._package.concat('.').
31         concat(controleGenericObjeto.Objeto.className)/];
32 [/for]
33 public class [controle.className /] {
34 }
35 [/file]
36
37 [comment modelo /]
38 [for (objeto : ControleGenericObjeto | controle.controleGenericoObjeto)]
39 [file (objeto.Objeto._package.replaceAll('\\.', '/').concat('/').
40         concat(objeto.Objeto.className.toUpperFirst()).concat('.java'), false, 'UTF-8')
```

```
41 package [objeto.Objeto._package /];
42
43 import java.io.Serializable;
44 import javax.persistence.Entity;
45 import javax.persistence.GeneratedValue;
46 import javax.persistence.GenerationType;
47 import javax.persistence.Id;
48
49 @Entity
50 public class [objeto.Objeto.className/] implements Serializable {
51
52 private static final long serialVersionUID = 1L;
53
54     @Id
55     @GeneratedValue(strategy = GenerationType.IDENTITY)
56     private [objeto.Objeto.type.concat(' ').concat(objeto.Objeto.id)/];
57     [for (atributo : Atributos | objeto.Objeto.atributo)]
58     private [atributo.type.concat(' ').concat(atributo.name)/];
59     [/for]
60
61     public [objeto.Objeto.type/] get[objeto.Objeto.id.toUpperFirst()/](){
62         return [objeto.Objeto.id/];
63     }
64
65     public [objeto.Objeto.type/] getId(){
66         return [objeto.Objeto.id/];
67     }
68
69     public void set[objeto.Objeto.id.toUpperFirst()/]([objeto.Objeto.type.concat(' ').
70                                                         concat(objeto.Objeto.id)/]){
71         this.[objeto.Objeto.id/] = [objeto.Objeto.id/];
72     }
73
74     [for (atributo : Atributos | objeto.Objeto.atributo)]
75     public [atributo.type/] get[atributo.name.toUpperFirst()/](){
76         return [atributo.name/];
77     }
78
79     public void set[atributo.name.toUpperFirst()/]([atributo.type.concat(' ').
80                                                     concat(atributo.name)/]){
```

```
81          this.[atributo.name/] = [atributo.name/];
82      }
83  [/for]
84  }
85  [/file]
86
87  [comment dao /]
88  [file (objeto.GenericDAO._package.replaceAll('\\.', '/').concat('/').
89          concat(objeto.GenericDAO.className.toUpperFirst()).concat('.java'), false, 'UTF-8')]
90  package [objeto.GenericDAO._package/];
91
92  import util.GenericDAO;
93  import java.util.ArrayList;
94  import [objeto.Objeto._package.concat('.').concat(objeto.Objeto.className)/];
95
96  public class [objeto.GenericDAO.className.toUpperFirst()/]
97              implements GenericDAO<[objeto.Objeto.className/]> {
98
99      public void salvar([objeto.Objeto.className/] o){
100         // Sets the query for inserting a 'o'.
101     }
102
103     public void deletar([objeto.Objeto.className/] o) {
104        // Sets the query for removing 'o'.
105     }
106
107     public void alterar([objeto.Objeto.className/] o) {
108             // Sets the query for updating 'o'.
109     }
110
111     public [objeto.Objeto.className/] buscarPorId([objeto.Objeto.className/] o){
112         return o;
113     }
114
115     public [objeto.Objeto.className/] buscarPorIdEstrangeiro
116                                     ([objeto.Objeto.className/] o, String sql, int i){
117         return null;
118     }
119
120 }
```

```
[/file]
[/for]
[/for]

[comment GenericDAO/]
[file ('util/GenericDAO.java', false, 'UTF-8')]
package util;

import java.util.ArrayList;

public interface GenericDAO<T> {

    public void salvar(T t);
    public void deletar(T t);
    public void alterar(T t);
    public T buscarPorId(T t);
    public T buscarPorIdEstrangeiro(T t, String sql, int id);
}
[/file]
[comment Aspecto /]
[file ('util/Aspect.aj', false, 'UTF-8')]
package util;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import util.GenericDAO;

public aspect Aspect {

    public EntityManager getEntityManager(){
        EntityManagerFactory emf = Persistence.
                                        createEntityManagerFactory("[anInit.className/]");
        EntityManager em = emf.createEntityManager();
        return em;
    }

    pointcut persist(GenericDAO x, Object o) :
            target(x) && args(o) && execution(public void GenericDAO.salvar(Object));
```

```
161     after(GenericDAO x, Object o) returning() : persist(x, o) {
162         EntityManager em = getEntityManager();
163         try {
164             em.getTransaction().begin();
165             em.persist(o);
166             em.getTransaction().commit();
167         } catch (Exception e) {
168             System.out.println("error: " + e);
169         } finally{
170             em.close();
171         }
172 
173     }
174 
175     pointcut merge(GenericDAO x, Object o):
176                 target(x) && args(o) && execution(public void GenericDAO.alterar(Object));
177     after(GenericDAO x, Object o) returning() : merge(x, o) {
178         EntityManager em = getEntityManager();
179         try {
180             em.getTransaction().begin();
181             em.merge(o);
182             em.getTransaction().commit();
183         } catch (Exception e) {
184             System.out.println("error: " + e);
185         } finally{
186             em.close();
187         }
188     }
189 
190     pointcut remove(GenericDAO x, Object o):
191                 target(x) && args(o) && execution(public void GenericDAO.deletar(Object));
192     after(GenericDAO x, Object o) returning() : remove(x, o) {
193         EntityManager em = getEntityManager();
194         try {
195             em.getTransaction().begin();
196             em.remove(o);
197             em.getTransaction().commit();
198         } catch (Exception e) {
199             System.out.println("error: " + e);
200         } finally{
```

```
            em.close();
        }
    }

    pointcut buscarPorId(GenericDAO x, Object o, Object i) :
                target(x) && args(o, i) &&
                execution(public Object GenericDAO.buscarPorId(Object, Object));
    Object around(GenericDAO x, Object o, Object i) : buscarPorId(x, o, i) {
    EntityManager em = getEntityManager();
        try {
            o = em.find(Object.class, i);
        } catch (Exception e) {
            System.out.println("error: " + e);
        } finally{
            em.close();
        }
        return o;
    }

    pointcut buscarPorIdEstrangeiro(GenericDAO x, Object o, String sql, int i) :
                target(x) && args(o, sql, i) &&
                execution(public Object GenericDAO.buscarPorId(Object, String, Object));
    Object around(GenericDAO x, Object o, String sql, int i) :
                buscarPorIdEstrangeiro(x, o, sql , i) {
        EntityManager em = getEntityManager();
        Query query = em.createQuery(sql, Object.class);
        query.setParameter(1, i);
        return (Object) query.getSingleResult();

    }

}
[/file]

[comment persistence.xml/]
[file ('META-INF/persistence.xml', false, 'UTF-8')]
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="[anInit.className/]" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    [for (controle : Controle | anInit.controle)]
    [for (objeto : ControleGenericObjeto | controle.controleGenericoObjeto)]
        <class>[objeto.Objeto._package.concat('.').concat(objeto.Objeto.className)/]</class>
    [/for]
    [/for]

        <properties>
            <property name="javax.persistence.jdbc.url" value="[anInit.persistence.url/]"/>
            <property name="javax.persistence.jdbc.user" value="[anInit.persistence.user/]"/>
            <property name="javax.persistence.jdbc.password" value="[anInit.persistence.password/]"/>
            <property name="javax.persistence.jdbc.driver" value="[anInit.persistence.driver/]"/>
        <property name="javax.persistence.schema-generation.database.action" value="create"/>
        </properties>
    </persistence-unit>
</persistence>
[/file]

[/template]
```

APÊNDICE C– ARTEFATOS GERADOS PELO FRAMEWORK NO EXPERIMENTO

Aluno.class

```
1  package modelo;
2
3  import java.io.Serializable;
4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.GenerationType;
7  import javax.persistence.Id;
8
9  @Entity
10 public class Aluno implements Serializable {
11
12 private static final long serialVersionUID = 1L;
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private int matricula;
17     private String nome;
18     private String cpf;
19     private String endereco;
20
21     public int getMatricula(){
22         return matricula;
23     }
24
25     public int getId(){
26         return matricula;
27     }
28
29     public void setMatricula(int matricula){
30         this.matricula = matricula;
31     }
32
33     public String getNome(){
34         return nome;
35     }
36
37     public void setNome(String nome){
38         this.nome = nome;
39     }
40     public String getCpf(){
41         return cpf;
42     }
43
44     public void setCpf(String cpf){
45         this.cpf = cpf;
46     }
47     public String getEndereco(){
48         return endereco;
49     }
50
51     public void setEndereco(String endereco){
52         this.endereco = endereco;
53     }
54 }
55
```

Debito.class

```
1  package modelo;
2
3  import java.io.Serializable;
4  import java.util.Date;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9
10 @Entity
11 public class Debito implements Serializable {
12
13 private static final long serialVersionUID = 1L;
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private int idDebito;
18     private double valor;
19     private Date data;
20
21     public int getIdDebito(){
22         return idDebito;
23     }
24
25     public int getId(){
26         return idDebito;
27     }
28
29     public void setIdDebito(int idDebito){
30         this.idDebito = idDebito;
31     }
32
33     public double getValor(){
34         return valor;
35     }
36
37     public void setValor(double valor){
38         this.valor = valor;
39     }
40     public Date getData(){
41         return data;
42     }
43
44     public void setData(Date data){
45         this.data = data;
46     }
47 }
48
```

Livro.class

```
1  package modelo;
2
3  import java.io.Serializable;
4  import java.util.Date;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9
10 @Entity
11 public class Livro implements Serializable {
12
13 private static final long serialVersionUID = 1L;
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private int idLivro;
18     private int prazo;
19     private String isbn;
20     private int edicao;
21     private Date ano;
22     private String titulo;
23     private int emprestavel;
24
25     public int getIdLivro(){
26         return idLivro;
27     }
28
29     public int getId(){
30         return idLivro;
31     }
32
33     public void setIdLivro(int idLivro){
34         this.idLivro = idLivro;
35     }
36
37     public int getPrazo(){
38         return prazo;
39     }
40
```

```
41      public void setPrazo(int prazo){
42          this.prazo = prazo;
43      }
44      public String getIsbn(){
45          return isbn;
46      }
47
48      public void setIsbn(String isbn){
49          this.isbn = isbn;
50      }
51      public int getEdicao(){
52          return edicao;
53      }
54
55      public void setEdicao(int edicao){
56          this.edicao = edicao;
57      }
58      public Date getAno(){
59          return ano;
60      }
61
62      public void setAno(Date ano){
63          this.ano = ano;
64      }
65      public String getTitulo(){
66          return titulo;
67      }
68
69      public void setTitulo(String titulo){
70          this.titulo = titulo;
71      }
72      public int getEmprestavel(){
73          return emprestavel;
74      }
75
76      public void setEmprestavel(int emprestavel){
77          this.emprestavel = emprestavel;
78      }
79  }
80
```

ItemEmprestimo.class

```
1  package modelo;
2
3  import java.io.Serializable;
4  import java.util.Date;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9
10 @Entity
11 public class ItemEmprestimo implements Serializable {
12
13 private static final long serialVersionUID = 1L;
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private int idItemEmprestimo;
18     private Date dataDevolucao;
19     private Date dataPrevista;
20     private Emprestimo emprestimo;
21     private Livro livro;
22
23     public int getIdItemEmprestimo(){
24         return idItemEmprestimo;
25     }
26
27     public int getId(){
28         return idItemEmprestimo;
29     }
30
31     public void setIdItemEmprestimo(int idItemEmprestimo){
32         this.idItemEmprestimo = idItemEmprestimo;
33     }
34
35     public Date getDataDevolucao(){
36         return dataDevolucao;
37     }
38
39     public void setDataDevolucao(Date dataDevolucao){
40         this.dataDevolucao = dataDevolucao;
41     }
42     public Date getDataPrevista(){
43         return dataPrevista;
44     }
45
46     public void setDataPrevista(Date dataPrevista){
47         this.dataPrevista = dataPrevista;
48     }
49     public Emprestimo getEmprestimo(){
50         return emprestimo;
51     }
52
53     public void setEmprestimo(Emprestimo emprestimo){
54         this.emprestimo = emprestimo;
55     }
56     public Livro getLivro(){
57         return livro;
58     }
59
60     public void setLivro(Livro livro){
61         this.livro = livro;
62     }
63 }
64
```

Emprestimo.class

```
1  package modelo;
2
3  import java.io.Serializable;
4  import java.util.Date;
5  import javax.persistence.Entity;
6  import javax.persistence.GeneratedValue;
7  import javax.persistence.GenerationType;
8  import javax.persistence.Id;
9
10 @Entity
11 public class Emprestimo implements Serializable {
12
13 private static final long serialVersionUID = 1L;
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private int idEmprestimo;
18     private Date dataEmprestimo;
19     private Date dataPrevista;
20     private Date dataDevolucao;
21     private double multa;
22     private boolean atraso;
23     private double valor;
24     private Aluno aluno;
25
26     public int getIdEmprestimo(){
27         return idEmprestimo;
28     }
29
30     public int getId(){
31         return idEmprestimo;
32     }
33
34     public void setIdEmprestimo(int idEmprestimo){
35         this.idEmprestimo = idEmprestimo;
36     }
37
38     public Date getDataEmprestimo(){
39         return dataEmprestimo;
40     }
```

```
41
42      public void setDataEmprestimo(Date dataEmprestimo){
43          this.dataEmprestimo = dataEmprestimo;
44      }
45      public Date getDataPrevista(){
46          return dataPrevista;
47      }
48
49      public void setDataPrevista(Date dataPrevista){
50          this.dataPrevista = dataPrevista;
51      }
52      public Date getDataDevolucao(){
53          return dataDevolucao;
54      }
55
56      public void setDataDevolucao(Date dataDevolucao){
57          this.dataDevolucao = dataDevolucao;
58      }
59      public double getMulta(){
60          return multa;
61      }
62
63      public void setMulta(double multa){
64          this.multa = multa;
65      }
66      public boolean getAtraso(){
67          return atraso;
68      }
69
70      public void setAtraso(boolean atraso){
71          this.atraso = atraso;
72      }
73      public double getValor(){
74          return valor;
75      }
76
77      public void setValor(double valor){
78          this.valor = valor;
79      }
80      public Aluno getAluno(){
81          return aluno;
82      }
83
84      public void setAluno(Aluno aluno){
85          this.aluno = aluno;
86      }
87  }
88
```

GenericDAO

```
1  package util;
2
3  import java.util.ArrayList;
4
5  public interface GenericDAO<T> {
6
7      public void salvar(T t);
8      public void deletar(T t);
9      public void alterar(T t);
10     public T buscarPorId(T t);
11     public T buscarPorIdEstrangeiro(T t, String sql, int id);
12
13 }
```

AspectDAO

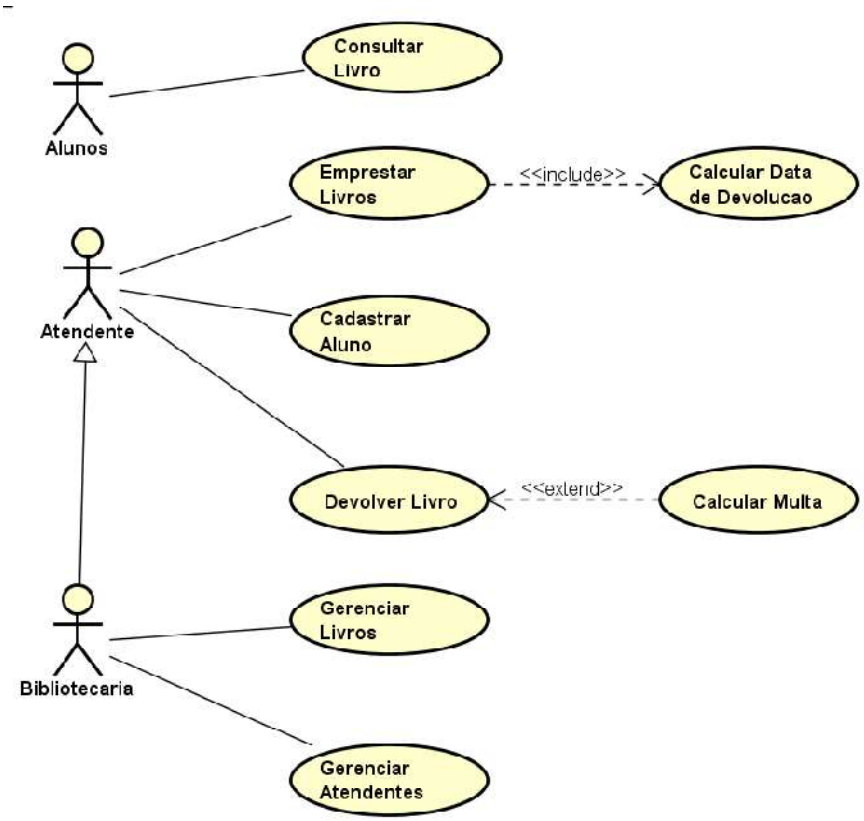
```
1  package util;
2  import javax.persistence.EntityManager;
8
9  public aspect Aspect {
10
11     public EntityManager getEntityManager(){
12         EntityManagerFactory emf = Persistence.createEntityManagerFactory("ExperimentoPU");
13         EntityManager em = emf.createEntityManager();
14         return em;
15     }
16
17     pointcut persist(GenericDAO x, Object o): target(x) && args(o) && execution(public void GenericDAO.salvar(Object));
18     after(GenericDAO x, Object o) returning() : persist(x, o) {
19         EntityManager em = getEntityManager();
20         try {
21             em.getTransaction().begin();
22             em.persist(o);
23             em.getTransaction().commit();
24         } catch (Exception e) {
25             System.out.println("error: " + e);
26         } finally{
27             em.close();
28         }
29
30     }
31
32     pointcut merge(GenericDAO x, Object o): target(x) && args(o) && execution(public void GenericDAO.alterar(Object));
33     after(GenericDAO x, Object o) returning() : merge(x, o) {
34         EntityManager em = getEntityManager();
35         try {
36             em.getTransaction().begin();
37             em.merge(o);
38             em.getTransaction().commit();
39         } catch (Exception e) {
40             System.out.println("error: " + e);
41         } finally{
42             em.close();
43         }
44     }
45
46     pointcut remove(GenericDAO x, Object o): target(x) && args(o) && execution(public void GenericDAO.deletar(Object));
47     after(GenericDAO x, Object o) returning() : remove(x, o) {
48         EntityManager em = getEntityManager();
49         try {
50             em.getTransaction().begin();
51             em.remove(o);
52             em.getTransaction().commit();
53         } catch (Exception e) {
54             System.out.println("error: " + e);
55         } finally{
56             em.close();
57         }
58     }
59
60     pointcut buscarPorId(GenericDAO x, Object o, Object i) : target(x) && args(o, i) &&
61     execution(public Object GenericDAO.buscarPorId(Object, Object));
62     Object around(GenericDAO x, Object o, Object i) : buscarPorId(x, o, i) {
63     EntityManager em = getEntityManager();
64         try {
65             o = em.find(Object.class, i);
66         } catch (Exception e) {
67             System.out.println("error: " + e);
68         } finally{
69             em.close();
70         }
71         return o;
72     }
73
74     pointcut buscarPorIdEstrangeiro(GenericDAO x, Object o, String sql, int i) : target(x) && args(o, sql, i) &&
75     execution(public Object GenericDAO.buscarPorId(Object, String, Object));
76     Object around(GenericDAO x, Object o, String sql, int i) : buscarPorIdEstrangeiro(x, o, sql , i) {
77         EntityManager em = getEntityManager();
78         Query query = em.createQuery(sql, Object.class);
79         query.setParameter(1, i);
80         return (Object) query.getSingleResult();
81     }
82 }
```

AlunoDAO

```
1  package dao;
2
3  import util.GenericDAO;
6
7  public class AlunoDAO implements GenericDAO<Aluno> {
8
9      public void salvar(Aluno o){
10         // Sets the query for inserting a 'o'.
11     }
12
13     public void deletar(Aluno o) {
14        // Sets the query for removing 'o'.
15     }
16
17     public void alterar(Aluno o) {
18             // Sets the query for updating 'o'.
19     }
20
21     public Aluno buscarPorId(Aluno o){
22         return o;
23     }
24
25     public Aluno buscarPorIdEstrangeiro(Aluno o, String sql, int i){
26         return null;
27     }
28
29 }
```

APÊNDICE D– DESCRIÇÃO DO EXPERIMENTO

Estamos implementando um sistema para gerenciamento de empréstimo de livros em uma biblioteca. O sistema tem 3 tipos de usuários, a bibliotecárias, os atendentes e os alunos. A bibliotecária é responsável pelas funções gerenciais dos sistemas, os atendentes pela função de empréstimo e os alunos apenas podem realizar consultas ao sistema. O sistema está na versão 1.0 e tem apenas funcionalidades básicas como mostrado no diagrama de caso de uso abaixo:



Neste momento estamos interessados em implementar a funcionalidade de empréstimo de livros. Para tanto é dado um conjunto de artefatos que auxiliam a realizar esta tarefa:

1. Requisitos (Descritos como Caso de Uso) e apresentados abaixo.
2. Diagrama de Classes.
3. Diagrama de Sequência da funcionalidade emprestar (está de acordo com a descrição dos casos de uso).
4. Classes de Domínio.
5. Script para geração da base de dados relacional do projeto em *Mysql*.

Artefatos 1 – Requisitos Descritivos como Casos de Uso

<table>
<tr><th colspan="2">Caso de Uso: Emprestar Livro</th></tr>
<tr><th>Fluxo Principal</th><th>Fluxo Alternativo</th></tr>
<tr><td rowspan="3">Fluxo de Principal (caminho básico):
1. O aluno apresenta os livros ao funcionário e a sua identificação.
2. O funcionário insere a identificação e os livros no sistema.
3. O sistema verifica se o Aluno está cadastrado.
4. O sistema verifica se o Aluno possui pendências.
5. O sistema cria um empréstimo.
6. Para cada livro:
6.1. O sistema verifica se não é exemplar que não pode ser emprestado.
6.2. O sistema cria um item de empréstimo.
6.3. O sistema associa o livro ao item.
7. O sistema calcula a data de devolução (Ponto de inclusão Calcula Data Devolução)
8. O sistema grava os dados do empréstimo.
9. O sistema imprime os dados do empréstimo.</td><td>3.a Aluno não cadastrado
3.a.1 – O sistema informa que o aluno não está cadastrado.
3.a.2 – O sistema finaliza o caso de uso.</td></tr>
<tr><td>4.a Aluno possui débito
4.a.1 – O sistema informa que o aluno está em débito.
4.a.2 – O sistema finaliza caso de uso.</td></tr>
<tr><td>6.1.a Livro reservado
6.1.a.1 – O sistema informa que o livro está reservado e não pode ser emprestado.
6.1.a.2 – O sistema informa a data de devolução do livro.
6.1.a.3 – Retorna ao passo 6.</td></tr>
</table>

Em conjunto como caso de uso emprestar livro a funcionalidade de calcular a data de devolução é executada que é implementada conforme a descrição abaixo:

Caso de uso: Calcula Data de Devolução	
Fluxo Principal	**Fluxo Alternativo**
1. Pega o número de livros do empréstimo. 1.1. Pega o prazo de devolução de cada livro. 1.2. Calcula a data de devolução do livro. 2. Selecione o maior prazo dentre todos os livros. 3. Retorna a data de devolução dos livros.	2.a. **Caso o cliente empreste 3 ou mais livros** 2.a.1 – Adiciona mais dois dias para cada livro após o 2º livro emprestado. 2.a.2 – Calcula a nova data. 2.a.3 – Retorna ao passo 3.

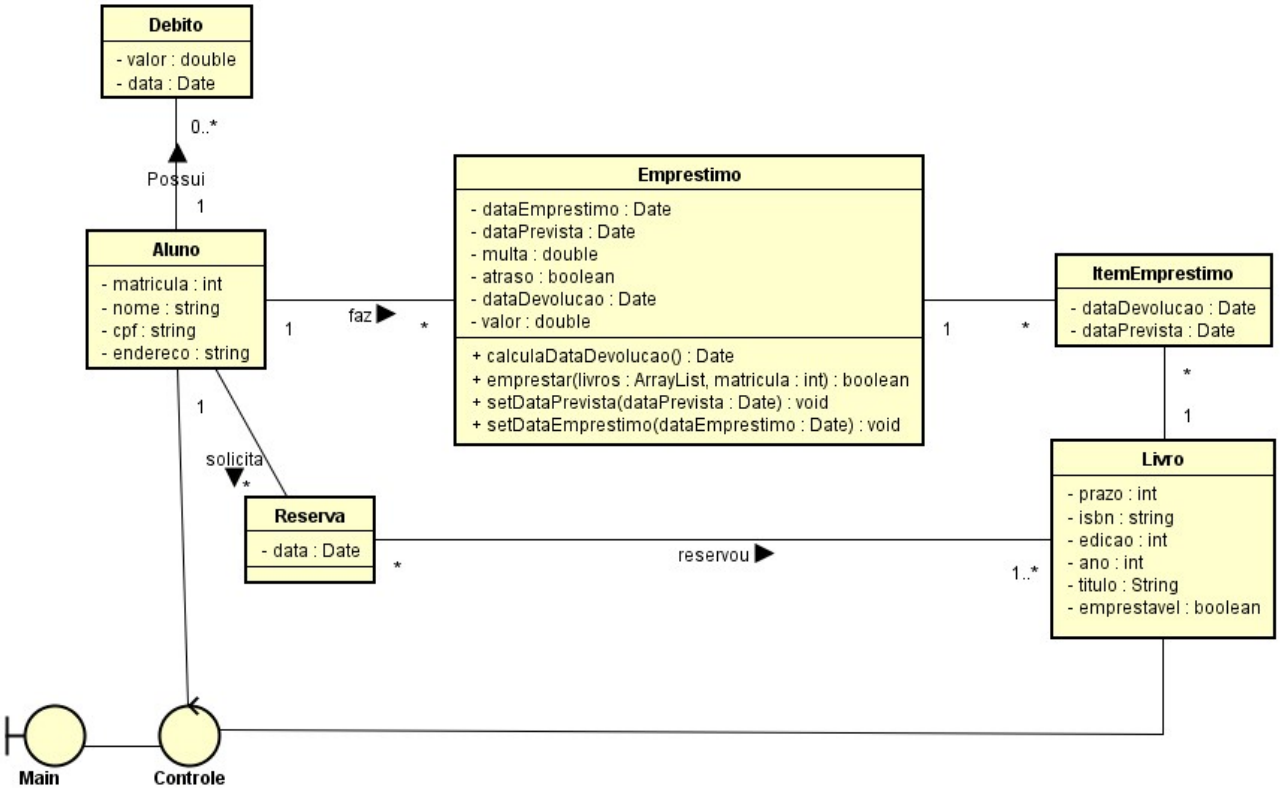
Tarefas

Todos os códigos e diagramas apresentados estão considerando apenas a camada de modelo, assim não tem nenhuma implementação da camada de persistência de dados. Sua tarefa é realizar a implementação da funcionalidade empréstimo (considerando as regras de negócios apresentadas nos artefatos) com a persistência de dados. Para isso, deve ser utilizado o padrão *Data Access Object* (DAO), utilizando o *Generic* DAO ao projeto.

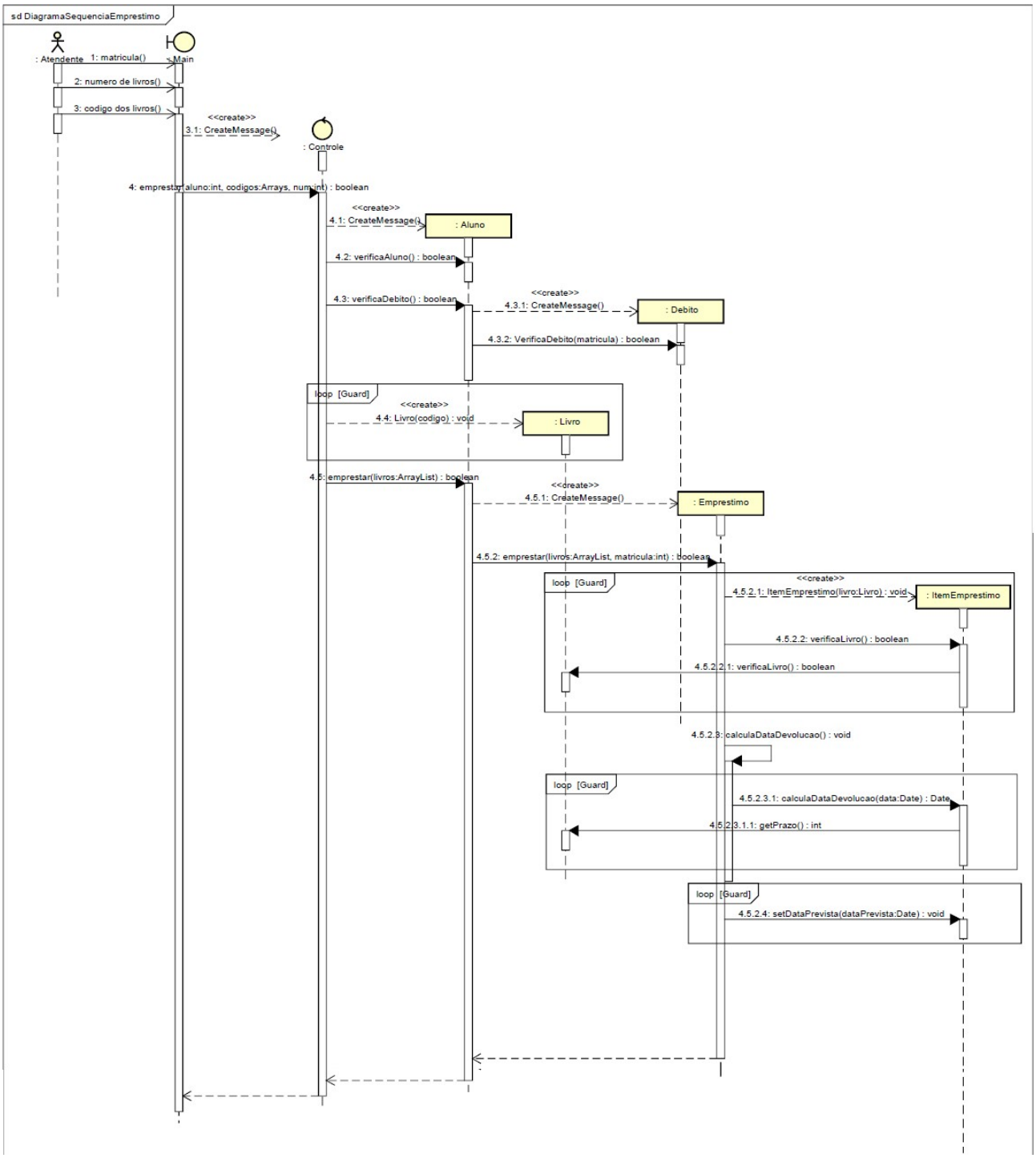
Para tanto, deve ser realizado várias implementações do mesmo projeto, conforme a descrição abaixo:

- **Implementação 1a** – Realizar a implementação da funcionalidade utilizando o Framework MDD e utilizando a tecnologia JPA.
- **Implementação 1b** – Realizar a implementação da funcionalidade sem utilizar o Framework MDD e utilizando a tecnologia JPA.
- **Implementação 2a** – Realizar a implementação da funcionalidade utilizando o Framework MDD e utilizando a tecnologia JDBC.
- **Implementação 2b** – Realizar a implementação da funcionalidade sem utilizar o Framework MDD e utilizando a tecnologia JDBC.

APÊNDICE E– DIAGRAMA DE CLASSE UTILIZADO NO EXPERIMENTO



APÊNDICE F– DIAGRAMA DE SEQUÊNCIA UTILIZADO NO EXPERIMENTO



APÊNDICE G– SCRIPT PARA GERAÇÃO DA BASE DE DADOS RELACIONAL DO PROJETO EM MYSQL

```
CREATE DATABASE `biblioteca` /*!40100 DEFAULT CHARACTER SET utf8 */;

CREATE TABLE `aluno` (
  `matricula` int(11) NOT NULL,
  `nome` varchar(45) DEFAULT NULL,
  `cpf` varchar(45) DEFAULT NULL,
  `endereco` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`matricula`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `debito` (
  `idDebito` int(11) NOT NULL,
  `matricula` int(11) DEFAULT NULL,
  `valor` double DEFAULT NULL,
  `data` date DEFAULT NULL,
  PRIMARY KEY (`idDebito`),
  KEY `matricula_idx` (`matricula`),
  CONSTRAINT `matricula` FOREIGN KEY (`matricula`) REFERENCES `aluno`
(`matricula`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `emprestimo` (
  `idemprestimo` int(11) NOT NULL,
  `dataPrevista` date DEFAULT NULL,
  `dataDevolucao` date DEFAULT NULL,
  `dataEmprestimo` date DEFAULT NULL,
  `multa` double DEFAULT NULL,
  `atraso` tinyint(4) DEFAULT NULL,
  `valor` double DEFAULT NULL,
  PRIMARY KEY (`idemprestimo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `livro` (
  `idlivro` int(11) NOT NULL,
  `titulo` varchar(45) DEFAULT NULL,
  `ISBN` varchar(45) DEFAULT NULL,
  `edicao` varchar(45) DEFAULT NULL,
  `ano` date DEFAULT NULL,
  `emprestavel` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`idlivro`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `itememprestimo` (
  `idItemEmprestimo` int(11) NOT NULL,
  `idEmprestimo` int(11) DEFAULT NULL,
  `dataPrevista` date DEFAULT NULL,
```

```
  `dataDevolucao` date DEFAULT NULL,
  `idLivro` int(11) DEFAULT NULL,
  PRIMARY KEY (`idItemEmprestimo`),
  KEY `idemprestimo_idx` (`idEmprestimo`),
  KEY `idlivro_idx` (`idLivro`),
  CONSTRAINT `idemprestimo` FOREIGN KEY (`idEmprestimo`) REFERENCES
`emprestimo` (`idemprestimo`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `idlivro` FOREIGN KEY (`idLivro`) REFERENCES `livro` (`idlivro`) ON
DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

APÊNDICE H– TERMO DE CONSENTIMENTO LIVRE E ESCLARECIMENTO

Uma abordagem para geração de código de persistência de dados JPA baseado em MDD e POA

Pesquisa realizada pela Engenharia de Software e Gestão do Conhecimento CCT da UENP, desenvolvida para o estudo da abordagem de geração de códigos de persistência dados JPA baseado em MDD e POA, para o Trabalho de Conclusão de Curso em Bacharelado em Ciência da Computação da Universidade Estadual do Norte do Paraná (UENP), pelo discente Felipe Igawa Moskado (felipe.moskado@gmail.com) e orientador Prof. André Menolli.

Por meio deste Termo de Confidencialidade, os Pesquisadores se comprometem a:

- Portar-se com discrição em todos os momentos da pesquisa acadêmica, não comentando ou divulgando qualquer tipo de informação que tenha sido repassada de forma oral ou escrita;
- Não divulgar nome do participante, em qualquer meio, a menos que expressamente autorizado por este;
- Não divulgar, em qualquer meio, os dados e informações individualizadas coletados durante o processo de pesquisa com o Participante;
- Divulgar, em formato de tese, artigos e apresentações, apenas os dados agregados, dos quais não se possa retirar ou inferir a identificação do Participante.

Eu, _____,

após ter lido e entendido as informações e esclarecido todas as minhas dúvidas referentes a este estudo com o Professor André Luís Andrade Menolli, ***CONCORDO VOLUNTARIAMENTE****, em participar do mesmo.*

_____ Data: ____/____/_____

Assinatura (do pesquisado ou responsável) ou impressão datiloscópica

Eu, Prof. André Luís Andrade Menolli, declaro que forneci todas as informações referentes ao estudo ao participante.

_____ Data: ____/____/_____

Assinatura

Equipe:

1- Nome: André Luís A. Menolli Telefone: (43) 35428014
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.

2- Nome: Felipe Igawa Moskado Telefone:(43) 996106734
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.

3- Nome: Luan de Souza Melo Telefone: (43) 999791318
Endereço: Universidade Estadual do Norte do Paraná, Centro de Ciências Tecnológicas- Campus Luiz Meneghel. Rod. 369 – KM 5,Vila Maria, CP 261 CEP 86360-000 - Bandeirantes - Paraná – Brasil.