UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Osvaldo Silva de Sousa Junior

UM FRAMEWORK PARA SUPORTAR DE FORMA SEMIAUTOMÁTICA A ATIVIDADE DE DESENVOLVIMENTO DE SOFTWARE PARA MAPREDUCE UTILIZANDO MDE

São Luís
2017

Osvaldo Silva de Sousa Junior

UM FRAMEWORK PARA SUPORTAR DE FORMA SEMIAUTOMÁTICA A ATIVIDADE DE DESENVOLVIMENTO DE SOFTWARE PARA MAPREDUCE UTILIZANDO MDE

Tese apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de DOUTOR em Engenharia de Eletricidade - Área de Concentração: Ciência da Computação.

Orientador: Denivaldo Cícero Pavão Lopes

Dr. em Ciência da Computação, UFMA

Coorientador: Aristófanes Corrêa Silva

Dr. em Ciência da Computação, UFMA

São Luís

2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Sousa Junior, Osvaldo Silva de.
Um framework para suportar de forma semiautomática a atividade de desenvolvimento de software para MapReduce utilizando MDE / Osvaldo Silva de Sousa Junior. - 2017.
196 f.

Coorientador(a): Aristófanes Corrêa Silva.
Orientador(a): Denivaldo Cícero Pavão Lopes.
Tese (Doutorado) - Programa de Pós-graduação em Engenharia de Eletricidade/ccet, Universidade Federal do Maranhão, São Luís, 2017.

1. Big Data. 2. Engenharia Dirigida por Modelos. 3. Framework. 4. Metamodelos. I. Lopes, Denivaldo Cícero Pavão. II. Silva, Aristófanes Corrêa. III. Título.

Osvaldo Silva de Sousa Junior

UM FRAMEWORK PARA SUPORTAR DE FORMA SEMIAUTOMÁTICA A ATIVIDADE DE DESENVOLVIMENTO DE SOFTWARE PARA MAPREDUCE UTILIZANDO MDE

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Osvaldo Silva de Sousa Junior e aprovada pela comissão examinadora.

Aprovada em 22 de Novembro de 2017.

BANCA EXAMINADORA

Denivaldo Cícero Pavão Lopes (orientador)

Dr. em Ciência da Computação, UFMA

Aristófanes Corrêa Silva (coorientador)

Dr. em Ciência da Computação, UFMA

Marcos Didonet Del Fabro

Dr. em Ciência da Computação, UFPR

Pedro de Alcântara dos Santos Neto

Dr. em Ciência da Computação, UFPI

María del Rosario Girardi Gutiérrez

Dra. em Ciência da Computação, UFMA

Francisco José da Silva e Silva

Dr. em Ciência da Computação, UFMA

Este trabalho é dedicado a minha amada esposa, Ana Amélia, que soube esperar, me incentivar e me amar durante os momentos mais difíceis dessa jornada. Dedico este trabalho também a memória do Professor Zair Abdelouahab por ter sempre confiado no meu potencial.

Resumo

A necessidade de analisar um grande volume e uma grande variedade de dados para extrair informações vem impulsionando investimentos em *Big Data*. Um exemplo seria os investimentos direcionados para a engenharia de *software* para plataformas de *Big Data*. Esses investimentos são recentes e emergentes, por isso vários desafios e oportunidades são encontrados na literatura, mas poucas abordagens foram propostas para suportá-los. Neste trabalho, um *framework* baseado em *Model-Driven Engineering* (MDE) e *Weaving* é proposto para suportar de maneira semiautomática a atividade de desenvolvimento de *software*, usando o modelo de *MapReduce* da plataforma de *Big Data*. Este *framework* foi denominado de *F2BD* e utiliza MDE para auxiliar no gerenciamento da complexidade do desenvolvimento de *software* através de modelos; e utiliza *Weaving* para unificar a visão entre modelos diferentes. Um processo de atividades é proposto para guiar a utilização do *F2BD*. Além disto, um metamodelo baseado em *Action Language for Foundational UML* (Alf) e uma notação gráfica denominada *VisualAlf* são propostos para complementar UML, objetivando suportar a descrição das ações modeladas nos corpos (i.e. campo *body*) dos métodos dos diagramas de classe UML. Propõe-se também metamodelos para *Platform-Description Model* (PDM) baseados em *MapReduce* e metamodelos para *Platform-Specific Model* (PSM) abstrato baseado em *Spark*. Definições de transformação de modelos escritas em *Atlas Transformation Language* (ATL) são propostas. Mostrou-se a aplicabilidade do *F2BD* através da construção de uma ferramenta (*TF2BD*) e a viabilidade da *TF2BD* através da construção de dois exemplos ilustrativos e uma avaliação experimental. A *TF2BD* suporta as tarefas envolvidas na atividade de desenvolvimento de *software*, disponibilizando editores para manipulação manual de modelos e definições de transformação para a geração automática de PSM, assim como do código fonte completo. Isto é possível, porque a *TF2BD* foi criada com base na arquitetura do *F2BD*. Assim, conclui-se que o *F2BD* é viável e pode ser utilizado para a construção de outras ferramentas.

Palavras-chave: Engenharia Dirigida por Modelos, Big Data, Metamodelos, *Framework*.

Abstract

The need to analyze a large volume and variety of data to extract information has been increasing investments in Big Data. One example would be investments targeted at software engineering for Big Data platforms. These investments are recent and emerging, so several challenges and opportunities are found in the literature, but few approaches have been proposed to support them. In this work, a framework based on Model-Driven Engineering (MDE) and Weaving is proposed to support the software development activity in a semiautomatic way, using the MapReduce model of the Big Data platform. This framework was called F2BD and uses MDE to assist in controlling the complexity of software development through models; and uses Weaving to unify the view between different models. An activity process is proposed to guide the use of F2BD. In addition, a metamodel based on Action Language for Foundational UML (Alf) and a graphical notation called VisualAlf are proposed to complement UML, aiming to support the description of the actions modeled in the bodies (i.e. body field) of methods of diagram class UML. Metamodels for Platform-Description Model (PDM) based on MapReduce and metamodels for abstract Platform-Specific Model (PSM) based on Spark are provided. Transformation definitions of models written in Atlas Transformation Language (ATL) are proposed. The applicability of F2BD was demonstrated through the construction of a tool (TF2BD) and the feasibility of TF2BD was demonstrated through the construction of two illustrative examples and an experimental evaluation. TF2BD supports the tasks involved in software development activity, providing editors for manual manipulation of models and transformation definitions for automatic generation of PSM as well as full source code. This is possible because TF2BD was built based on the F2BD architecture. Thus, it is concluded that F2BD is feasible and can be used for the construction of other tools.

Keywords: Model Driven-Engineering, Big Data, Metamodels, Framework.

Agradecimentos

A Deus, por ter conduzido meus passos até agora me dando a força necessária para não desistir.

À memória do professor Ph.D Zair Abdelouahab por ter me orientado até onde as suas forças permitiram.

Ao professor Dr. Aristófanes C. Silva por ter aceitado temporariamente o desafio de assumir a orientação deste trabalho, pelo auxílio imprescindível por meio de sua experiência nos momentos cruciais e por todos os conselhos que permitiram que este trabalho pudesse ser concluído.

Ao professor Dr. Denivaldo C. Pavão Lopes que iniciou coorientando este trabalho de maneira minuciosa com seus conselhos, sua paciência, sua amizade e sua experiência e por ter aceito assumir a orientação deste trabalho assim que o foi permitido.

A minha esposa, Ana Amélia, pelos incentivos, paciência e, acima de tudo, pela compreensão nesse período tão difícil das nossas vidas.

Aos meus pais, Osvaldo e Luiza, por acreditarem sempre em mim e por suas orações.

A todos os meus familiares e amigos pelo apoio direto ou indireto.

Aos diretores do NTI, Nélio A. Guilhon, que me acompanhou por um período, e José R. Santana Netto, que me acompanha hoje, pelo incentivo e compreensão.

Aos colegas do Departamento de Desenvolvimento/NTI pela ajuda e apoio sempre que foi necessário.

Aos colegas do LESERC pelo companheirismo.

Aos órgãos de fomento CNPq e FAPEMA pelo suporte financeiro oferecido ao LESERC.

Ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão pela oportunidade de realizar este trabalho.

"...tudo posso naquele que me fortalece"

Filipenses 4.13

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ADD Attribute Driven Design.

API Application Programming Interface.

ATL Atlas Transformation Language.

DSML Domain Specific Modeling Languages.

EBNF Extended Backus-Naur Form.

EMF Eclipse Modeling Framework.

F2BD Framework para Desenvolvimento de Sistemas de Software para MapReduce de Big Data.

GMF Graphical Modeling Framework.

HDFS Hadoop Distributed File System.

IDE Integrated Development Environment.

MDA Model-Driven Architecture.

MDD Model-Driven Development.

MDE Model-Driven Engineering.

MDT Model-Driven Testing.

OMG Object Management Group.

PDM Platform-Description Model.

PDY Processo de Desenvolvimento em Y.

PIM Platform-Independent Model.

PSM Platform-Specific Model.

RDD Resilient Distributed Datasets.

TF2BD Tool based on Framework to Develop Software Systems to MapReduce of Big Data.

UML Unified Modeling Language.

Sumário

1 Introdução

Este primeiro capítulo descreve o contexto, a problemática, a motivação, a hipótese, os objetivos e a justificativa desta tese. Em seguida, a metodologia de pesquisa empregada é descrita e, depois, a estrutura deste manuscrito é apresentada.

1.1 Contexto

A popularização do acesso à Internet vem proporcionando a geração de um volume e uma variedade de dados que computadores de grande porte sozinhos não têm como armazenar ou analisar. Esse volume e variedade de dados eram encarados como um problema, mas a necessidade cada vez mais crescente pela busca de informações relevantes fez com que as grandes corporações percebessem que esse volume e essa variedade de dados poderiam trazer um diferencial competitivo no atual mercado globalizado [89]. Neste contexto, o termo *Big Data* [16, 32] é usado para conceituar esse grande volume e variedade de dados que têm características diversificadas e um crescimento acelerado.

Sener et al. [89] demonstram que o interesse massivo por *Big Data* é recente, meados de 2012, e que esse interesse está adotando um crescimento exponencial. Nos últimos anos, o investimento massivo da comunidade científica para oferecer soluções para os mais diversos aspectos do *Big Data* é notório, por exemplo: visualização [22, 50, 67], análise dos dados [58, 86, 90], armazenamento [36, 46, 55] e segurança [29, 64, 96, 103]. Fang et al. [37] confirmam o interesse por *Big Data*, apresentando uma discussão sobre os desafios e sobre as soluções existentes na indústria e na academia a partir da perspectiva de engenheiros, cientistas da computação e estatísticos, enquanto Lee et al. [56] apresentam os desafios e as oportunidades relacionados aos dados geoespaciais.

Dentre os investimentos realizados, pode-se destacar os voltados para o desenvolvimento de *software* que permitem a Análise de *Big Data* para a extração de informações. Os *frameworks MapReduce* da plataforma *Hadoop* [47] e *Spark* [104] são largamente utilizados como soluções para auxiliar na implementação de *software* para

Análise de *Big Data*, pois encapsulam toda parte de distribuição e de comunicação na rede. No entanto, ressalta-se que o desenvolvimento de *software* não é focado apenas na implementação. Assim, desenvolver um *software* não é um processo elementar e deve ser planejado cuidadosamente, para que os artefatos produzidos se mantenham de acordo com o que foi pensado. Desse modo, isso facilitará que esse *software* possa ser mantido e/ou evoluído. Nesse contexto, soluções que visem auxiliar com a complexidade do desenvolvimento, para *software Big Data*, como um todo são necessárias.

1.2 Problemática

Uma preocupação emergente e recente, relacionada à pesquisa sobre a plataforma *Big Data*, diz respeito a como aplicar conceitos, métodos e técnicas derivados da Engenharia de *Software* nessa plataforma. Bagheri et al. [9] em uma chamada especial para um periódico, que trata especificamente sobre Engenharia de *Software* para *Big Data*, solicitam trabalhos que ofereçam de forma prática ou teórica o "desenvolvimento e fornecimento de métodos, técnicas, processos e ferramentas apropriados de Engenharia de *Software* que suportem o desenvolvimento eficiente e eficaz e a manutenção de aplicações de *Big Data* através das diferentes fases do ciclo de vida de desenvolvimento de *software*".

Enquanto, Baresi et al. [10] em um dos tópicos de uma outra chamada especial para periódicos, sobre Engenharia de *Software* para *Big Data*, levantam a seguinte questão: "Quais são os desafios de Engenharia de *Software* impostos pela construção de *software* para *Big Data*?". Assim, pode-se apontar que um dos problemas encontrados no contexto da Engenharia de *Software* aplicada à plataforma *Big Data* é a falta de soluções (*frameworks*, ferramentas, técnicas), que apoiem de maneira integral e com algum nível de automaticidade a atividade de desenvolvimento de *software* para essa plataforma.

A atividade de desenvolvimento é composta de tarefas como: análise, projeto e implementação. Estas são tarefas que abrangem desde a modelagem do problema até a criação do código fonte completo.[1] As tarefas de análise e projeto,

[1]Código fonte completo refere-se ao esqueleto dos métodos mais o seu comportamento (lógica) do método.

geralmente, utilizam modelos para compreender, descrever e documentar o problema. Recentemente, Guerriero et al. [40] e Chen et al. [21] apresentaram soluções que apoiam a modelagem de problemas para *Big Data*, mas não ofereceram soluções a nível de implementação. A tarefa de implementação, por sua vez, tem como objetivo produzir o código fonte completo. Essa tarefa para *Big Data* seria complexa, pois existem muitas variáveis envolvidas (rede, compartilhamento de recursos, paralelismo, entre outras), mas algumas soluções que permitem encapsular essa complexidade foram propostas pela Apache em forma de *frameworks*, como *MapReduce* da *Hadoop* [100] e o *Spark* [7]. Esses *frameworks* implementam ou se baseiam no modelo *MapReduce*[2].

Soluções isoladas para as tarefas da atividade de desenvolvimento voltadas para implementações que utilizam *frameworks MapReduce* de *Big Data* existem [21, 40, 82], mas essas soluções provocam uma falta de sincronismo entre o que é modelado e o código fonte criado. Essa falta de sincronismo prejudica a documentação, não preserva os investimentos[3], gera retrabalho e diminui a produtividade em manutenções e evoluções. Dessa forma, é necessário pensar em soluções que suportem de maneira automática ou semiautomática as tarefas da atividade de desenvolvimento para implementações que utilizam *frameworks MapReduce* de *Big Data*.

1.3 Motivação

Big Data vem chamando atenção de vários segmentos da sociedade e da comunidade acadêmica, por isso investimentos em pesquisas e em desenvolvimento de soluções automatizadas estão sendo realizados [45]. Investimentos recentes têm resultado em soluções isoladas que auxiliem nas tarefas da atividade de desenvolvimento, mas soluções que gerenciem a complexidade dessa atividade do início ao fim ainda são escassas. Assim, pode-se inferir que a proposta de soluções que

[2]O *framework MapReduce* utilizado como solução de processamento pela plataforma *Hadoop* leva o mesmo nome do modelo *MapReduce* a qual nos referimos. Esse modelo é baseado no paradigma de dividir para conquistar e pode ser resumido em duas fases: *Map* e *Reduce*. Na fase de *Map*, os dados são divididos em diversos nós e processados conforme processamento predeterminado, enquanto na fase de *Reduce*, o resultado dos processamentos são sintetizados e disponibilizados [32, 59]

[3]Preservar investimentos nesse trabalho se refere a lógica de negócio incorporada completamente em modelos. Geralmente na modelagem apenas uma visão macro da lógica do negócio é representada através da definição dos métodos, pois os modelos não oferecem uma forma padronizada e homogênea de incluir uma visão micro da lógica de negócio que é representada pela definição do comportamento do método.

permitam gerenciar a complexidade da atividade do desenvolvimento para *Big Data*, desde a modelagem do problema até a criação do código fonte, é um desafio [9,10,45].

Uma solução para esse desafio pode ser a utilização da abordagem baseada em modelos, conhecida como *Model-Driven Engineering (MDE)*, pois a proposta central dessa abordagem é gerenciar a complexidade presente no desenvolvimento e evolução de *software* e expressar os conceitos de domínio de forma eficiente [87]. Essa abordagem consegue alcançar tais objetivos, elevando o nível de abstração, utilizando metamodelos, modelos, notações e regras de transformações [79]. Por isso, vem sendo usada para diversos fins e nas mais diversas áreas, por exemplo: segurança [63], teste de *software* [70,92] e fusão de bancos de dados [19].

Graças a MDE e a modelos, pode-se abstrair os conceitos do modelo de *MapReduce*[4] e da lógica de negócio. Esses modelos apresentariam visões diferentes e independentes, que seriam unificadas posteriormente, por meio do *weaving* de modelos, para que o desenvolvimento possa ser realizado. As visões diferentes oferecem flexibilidade e portabilidade para as soluções *Big Data*. Os modelos podem ser utilizados também para descrever *frameworks* ou outras soluções existentes permitindo que eles possam ser utilizados no desenvolvimento. Isso irá garantir a extensibilidade para as soluções *Big Data*. Assim, também se pode automatizar algumas tarefas, pois os artefatos utilizados para o desenvolvimento são modelos. Isso permite diminuir o retrabalho e aumentar a qualidade do *software Big Data*.

1.4 Hipótese

Um *framework*, baseado nos conceitos de MDE e de *weaving*, permitirá criar soluções que suportem as atividade de desenvolvimento de *software* (análise, projeto e implementação) para o modelo de *MapReduce* de *Big Data*.

1.5 Objetivos

O objetivo geral do trabalho é propor um *framework* baseado em MDE e no modelo de *weaving* para suportar de forma semiautomática a atividade de

[4] *MapReduce* é o modelo mais utilizado para implementação de *software Big Data*.

desenvolvimento de *software* para plataforma de *Big Data* que usem o modelo *MapReduce*.

1.5.1 Objetivos Específicos

Os objetivos específicos são os seguintes:

- Estender o metamodelo de *UML* para permitir que a lógica do negócio possa ser incorporada no modelo, através da inclusão do comportamento dos métodos;
- Propor metamodelos para o *Platform Description Model - PDM* de *MapReduce* e para o *weaving* de modelos;
- Criar um fragmento do modelo da *Application Programming Interface (API)* de *Spark* conforme metamodelo da linguagem Java;
- Propor uma notação gráfica que permita ao usuário modelar a lógica de negócio de forma visual e intuitiva;
- Criar um processo de atividades para orientar a utilização do *framework* proposto;
- Implementar um protótipo de uma ferramenta como *plug-in* para *Integrated Development Environment - IDE Eclipse* com base no *framework* proposto e com base na API do modelo *MapReduce* utilizada pelo *framework Spark*;
- Aplicar o *framework* e a ferramenta propostos em exemplos ilustrativos para desenvolver *software* para *Big Data*;
- Realizar um estudo experimental da ferramenta criada em um contexto *in vitro* com estudantes e profissionais para desenvolver pelo menos um problema.

1.6 Justificativa

Sommerville [91] comenta que a falta de soluções que apoiem de maneira automática ou semiautomática a atividade de desenvolvimento para um processo de *software* é comum, visto que a diversidade de processos de desenvolvimento existentes é grande e que não existe um processo ideal.

Algumas das soluções existentes para desenvolvimento de *software*, por exemplo, as ferramentas de modelagem, tentam ser generalistas, ou seja, procuram atender a todos os domínios. Essa generalidade, muitas vezes, faz com que as soluções propostas sejam muito limitadas. Assim, faltam investimentos em soluções mais específicas. A especificidade pode auxiliar na diminuição de tarefas manuais, proporcionando um aumento de produtividade e na qualidade dos *software* produzidos [91]. Uma solução que seja específica para um contexto pode garantir que a documentação produzida e o código fonte gerado estejam sempre coesos e que os investimentos realizados sejam preservados, facilitando, assim, a geração, a manutenção e a evolução do código fonte de um *software*.

O *framework* proposto, por meio do uso de modelos que descrevam aspectos desejados, permite criar ferramentas especificas para uma plataforma, por exemplo *MapReduce* de *Big Data*. Essa especificidade potencializará o desenvolvimento, automatizando algumas tarefas que deveriam ser realizadas de maneira manual ao mesmo tempo que garantirá a preservação dos investimentos, a facilidade em manutenções corretivas e a evolução do *software* criado.

A especificidade decorrerá dos metamodelos propostos, a automatização das tarefas será dada através das definições de regras de transformações propostas, e a preservação da lógica de negócio será feita através de sua incorporação nos modelos. A lógica de negócio incorporada em modelos permite que os investimentos sejam preservados, assim como pode permitir que tarefas de manutenção, de evolução e de documentação sejam suportadas.

É importante destacar que este trabalho pretende auxiliar no desenvolvimento de soluções *Big Data*, que utilizem os conceitos do modelo de *MapReduce*, independente de sua implementação. Assim, este trabalho não propõe um novo *framework* de implementação, como o *MapReduce* da *Hadoop* ou o *Spark*, mas um *framework*, que auxilie no processo de desenvolvimento, utilizando *frameworks* existentes, modelos e transformações entre modelos para criar ferramentas que permitam gerar o código fonte completo (definição + comportamento do método).

1.7 Metodologia de Pesquisa

A metodologia de pesquisa proposta, para o desenvolvimento deste trabalho, pode ser listada a seguir:

1. Pesquisa bibliográfica, com intuito de coletar informações de livros, artigos, teses, dissertações, periódicos, anais de congressos, tutoriais e *websites* para uma ambientação da literatura sobre o estado da arte de: MDE, *Big Data* e engenharia de *software* aplicada a *Big Data*;

2. Proposta de um *framework* baseado em MDE para suportar o desenvolvimento de *software* para plataforma de *Big Data* que use o modelo *MapReduce*. O *framework* proposto será construído em três etapas: Na primeira etapa, define-se os modelos de entrada que irão capturar a lógica do negócio e as características (ou aspectos) do modelo *MapReduce* da plataforma *Big Data*. Na segunda etapa, define-se os modelos específicos de plataformas, de tal forma que o código do *software* se aproxime da linguagem alvo; e finalmente, na terceira etapa, define-se a maneira de gerar o código fonte completo na linguagem alvo;

3. Criação e extensão dos metamodelos necessários para suportar o *framework* proposto. Primeiramente, o metamodelo que especificará a criação do *Platform-Description Model (PDM)*, baseado no modelo de *MapReduce*, será definido. Em seguida, o metamodelo para especificar a criação do *Plataform-Specific Model* (PSM) abstrato para *MapReduce* será definido. Finalmente, a extensão do metamodelo de UML será feita para permitir que a lógica de negócio possa ser inclusa, através da definição do comportamento dos métodos;

4. Criação das definições de transformações necessárias para sustentar o funcionamento do *framework*. A proposta do *framework* é que ele possibilite que o processo seja semiautomático. Assim sendo, nessa etapa, um estudo dos metamodelos de origem e destino em cada fase do *framework*, é feito de tal maneira que se possa criar definições de transformação. Essas definições de transformação irão permitir que o modelo de origem se transforme automaticamente no modelo de destino através de motores de transformações. A linguagem de transformação *Atlas Transformation Language (ATL)* será utilizada para escrever tais definições de transformação;

5. Proposta de um processo de atividades para utilização do *framework* proposto. Para isso, define-se um conjunto de tarefas que orientem a utilização do *framework*, de tal forma que o código fonte para solução procurada possa ser encontrado;

6. Proposta de implementação de uma notação gráfica e extensão da linguagem Alf [72] para permitir que o usuário modele a lógica do negócio de forma visual e intuitiva. Nessa etapa, propõe-se a construção de representações gráficas para as classes que foram introduzidas no metamodelo de UML, de tal forma que o *Plataform-Independent Model (PIM)* possa ser construído graficamente. A ferramenta *Graphical Modeling Framework (GMF)* é utilizada para esse fim;

7. Criação de uma ferramenta, utilizando o *framework* proposto. Nessa etapa, primeiramente, define-se o metamodelo para o PSM abstrato com base na implementação da *Application Programming Interface* (API) do *framework Spark*. Em seguida, o PSM concreto e o modelo da API do *framework Spark*, que implementa o comportamento de *MapReduce*, são criados conforme o metamodelo da linguagem Java. Posteriormente, serão criadas as definições de transformações do PSM abstrato para o PSM concreto e, em seguida, do PSM concreto para o código fonte conforme a *Extended Backus-Naur Form (EBNF)* da linguagem Java;

8. Avaliação da ferramenta proposta. Um estudo experimental é feito em um contexto *in vitro* com estudantes e profissionais para desenvolver um problema baseado em *Big Data* com e sem a utilização da ferramenta proposta.

1.8 Estrutura do Manuscrito

Este manuscrito está estruturado em oito capítulos e cinco apêndices.

O primeiro capítulo contextualiza o trabalho apresentado, contendo a problemática, a motivação, os objetivos, a justificativa para solução proposta e a metodologia seguida.

O segundo capítulo apresenta a fundamentação teórica que subsidia este trabalho com intuito de levar o leitor ao entendimento mínimo necessário para bem compreendê-lo.

O terceiro capítulo apresenta o estado da arte. Nesse capítulo, os trabalhos diretamente relacionados a Engenharia de *Software* para *Big Data* serão apresentados.

O quarto capítulo apresenta o *framework* proposto. Nesse capítulo, será apresentado a arquitetura do *framework*, assim como, os metamodelos e modelos envolvidos, seguido da linguagem VisualAlf proposta e do processo de utilização do *framework* proposto.

No quinto capítulo, a implementação de uma ferramenta, usando o *framework* proposto será feita. A ferramenta será construída como um *Plug-in* para *Integrated Development Environment (IDE) Eclipse* e será composta de editores e regras de transformações.

No sexto capítulo, dois exemplos ilustrativos são apresentados para demonstrar a utilização da ferramenta construída, usando o *framework* proposto.

O sétimo capítulo contém uma avaliação experimental sobre a ferramenta construída no Capítulo 5.

O oitavo capítulo, que encerra os capítulos do trabalho, apresenta as conclusões, contribuições e trabalhos futuros.

O Apêndice A apresenta a catalogação de alguns trabalhos sobre MDE e *Big Data* nas suas mais diversas aplicabilidades.

O Apêndice B apresenta a notação gráfica proposta, assim como um exemplo ilustrativo usando VisualAlf.

O Apêndice C apresenta os editores criados para a manipulação dos modelos.

Finalmente, os Apêndices D e E, apresentam, respectivamente, o questionário utilizado para selecionar os participantes do estudo experimental e o questionário utilizado para avaliar a percepção da qualidade dos participantes após o experimento.

2 Contexto Tecnológico

Este capítulo apresenta a fundamentação teórica, contendo os principais conceitos e tecnologias utilizadas na concepção e desenvolvimento do *framework* proposto para suportar o desenvolvimento de *software* para *MapReduce* de *Big Data*.

Inicialmente, apresenta-se os conceitos de *Model-Driven Engineering* (MDE) incluindo *Model-Driven Architecture (MDA)* e *Weaving* de modelos. Posteriormente, os conceitos referentes a *Big Data* e o desenvolvimento para *Big Data* são apresentados.

2.1 *Model-Driven Engineering* (MDE)

O processo de criação de *software* envolve fatores empíricos e está em constante evolução. Os avanços conquistados ao longo de mais de quatro décadas só foram possíveis, porque profissionais vêm desenvolvendo pesquisas na área de engenharia de *software*. Os engenheiros de *software*, responsáveis por tais avanços, vêm pesquisando novos métodos e ferramentas para permitir que o desenvolvimento de *software* possa ser feito com rapidez, qualidade e, acima de tudo, com baixos custos [25].

Uma das grandes questões da Engenharia de *Software* nas últimas décadas é como gerenciar a complexidade de um sistema de *software* elevando a abstração. Mellor et al. [66] confirmam tal afirmação, quando comentam que "a história do desenvolvimento de *software* é uma história de elevação do nível de abstração". Parreiras [79] complementa, "elevar o nível da abstração é um dos princípios básicos da Engenharia de *Software* para reduzir a complexidade do *software* e aumentar a produtividade em seu desenvolvimento".

Nesse contexto, surge a abordagem, baseada em modelos, conhecida como MDE, para suportar o desenvolvimento de *software*. A MDE permite o gerenciamento da complexidade, utilizando modelos para suportar as tarefas do desenvolvimento, com o intuito de aumentar a produtividade e preservar os investimentos [24].

Hutchinson et al. [85] resumem a ideia de MDE quando afirmam que ela é "o uso sistemático de modelos como artefatos primários durante um processo de Engenharia de *Software*". De forma semelhante, Cabot et al. a definem como "uma metodologia para aplicar as vantagens da modelagem para atividades de engenharia de *software*". Enquanto Oliveira [69], conceitua MDE da seguinte maneira: é "uma abordagem de desenvolvimento de *software* que se concentra na criação de modelos que descrevem os elementos de um sistema e orientam sua implementação". Para Parreira [79], a MDE consegue alcançar tais objetivos, elevando o nível de abstração, utilizando modelos, notações e regras de transformações.

2.1.1 Modelos

O ser humano utiliza modelos para tentar compreender, prever, investigar e documentar acontecimentos antes deles serem realmente produzidos [18]. Considere, como exemplo, a construção de um prédio sem uma planta. Como dimensionar as instalações elétricas, hidráulicas e sanitárias? Como definir as questões estruturais: pilares, vigas e lages? Os engenheiros civis teriam imensa dificuldade de executar e acompanhar a construção, pois teriam que lidar com a complexidade do problema durante a execução e não previamente. A premissa é semelhante para o desenvolvimento de *software*. Quais os elementos-chaves do problema? Como eles estão relacionados? Quais limitações eles apresentam? Qual arquitetura deve ser usada? Por meio da utilização de modelos, essas questões são compreendidas antes do desenvolvimento do *software*. Dessa forma, pode-se afirmar que modelos são elementos essenciais para lidar com a complexidade da realidade, visto que em ambos os casos exemplificados o produto é originado com base nos modelos criados [52].

A importância de modelos se tornou tão relevante que abordagens como MDE procuram declarar que tudo em seu contexto são modelos [18]. Por exemplo, modelos são utilizados para descrever os artefatos de um *software* [52]. Assim, pode-se considerar que os modelos passaram de atores coadjuvantes para atores principais dentro desse âmbito de desenvolvimento [17, 25].

Alguns conceitos de modelos apresentados na literatura: Rodrigues [25] define modelo como: "um sistema que ajuda a definir e a dar respostas do sistema em estudo, sem a necessidade de considerá-lo diretamente"; enquanto Mellor et al. [66]

definem modelo, como: "um conjunto de elementos que descrevem alguma realidade física, abstrata ou hipotética"; já Cabot et al. [18] conceitua modelos como "uma representação simplificada ou parcial da realidade, definida para realizar uma tarefa ou chegar a um acordo".

2.1.2 Metamodelos

Os modelos são os principais artefatos em abordagens dirigidas por modelos, assim é natural que eles sejam definidos conforme outros modelos. Modelos que descrevem outros modelos são chamados de metamodelos [74] e a relação entre um modelo e seu metamodelo pode ser definida como uma relação de instanciação, ou seja, um modelo é uma instância de seu metamodelo. A Figura 2.1 apresenta a relação entre modelo e metamodelo.

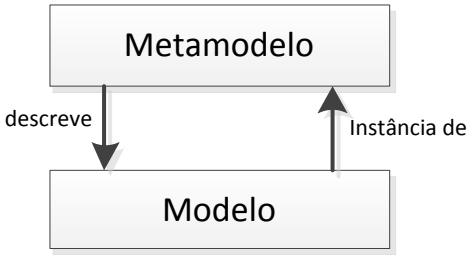


Figura 2.1: Relação entre modelo e metamodelo.

Metamodelos são essenciais em abordagens dirigidas por modelos, porque eles especificam a forma que os modelos podem ser construídos. Dessa forma, Mellor et al. [66] conceituam metamodelo como "um modelo de uma linguagem de modelagem que define a estrutura, a semântica e as restrições para uma família de modelos". Assim como, Cabot et al. [18] que afirma que "um metamodelo constitui basicamente a definição de uma linguagem de modelagem, uma vez que fornece uma maneira de descrever toda a classe de modelos que podem ser representados por essa linguagem".

Além das abstrações de modelos e metamodelos, existe outra chamada de metametamodelo. Um metametamodelo possui um nível de abstração maior, pois seu objetivo é descrever modelos que descrevem modelos, ou seja, os metametamodelos descrevem os metamodelos. A relação entre o metamodelo e seu metametamodelo é a mesma relação entre um modelo e seu metamodelo. A OMG define uma arquitetura de modelagem em quatro camadas para ilustrar a relação entre modelos, metamodelos

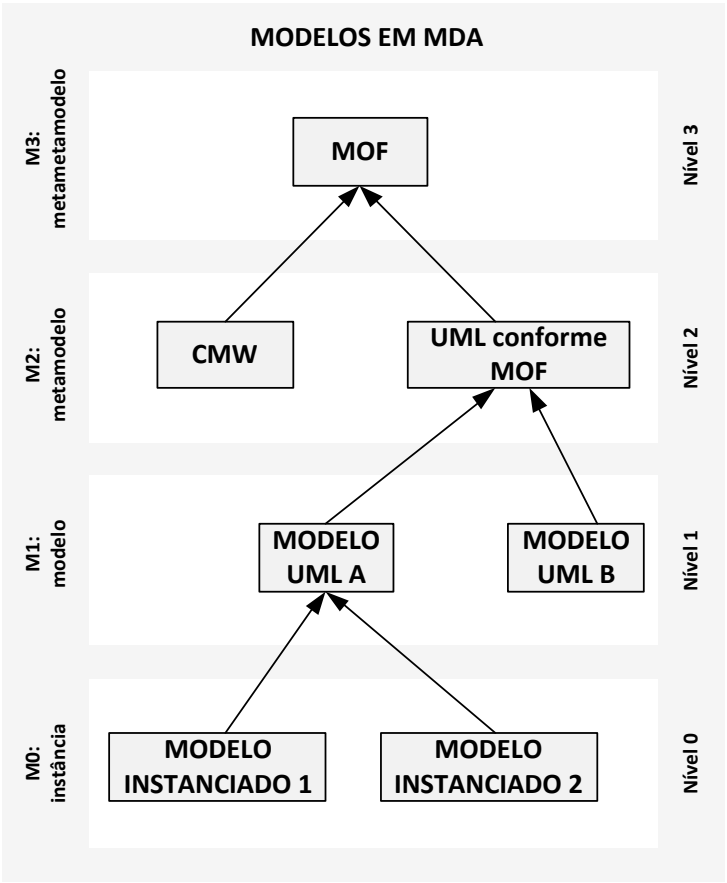


Figura 2.2: Arquitetura de quatro camadas da OMG aplicada para modelos construídos usando MDA (adaptado da OMG [73]).

e metametamodelos [73]. A Figura 2.2 apresenta a arquitetura de quatro camadas da OMG aplicada para modelos construídos usando MDA.

As camadas são descritas a seguir:

- **M0 (instância de M1)**: Nesta camada, os conceitos do mundo real são instanciados. Estes conceitos estão relacionados a um domínio computacional em uma plataforma final. A camada M0 é uma instância da camada M1;
- **M1 (modelo)**: Nesta camada, encontram-se os modelos do mundo real que descrevem como os conceitos devem ser instanciados em M0. Estes modelos estão escritos de acordo com a linguagem definida pelo metamodelo da camada M2;
- **M2 (metamodelo)**: A função desta camada é definir os metamodelos que irão descrever as linguagens que serão utilizadas em M1. Os metamodelos de M2 devem estar escritos de acordo com as metalinguagens definidas em M3;

- **M3 (metametamodelo)**: Esta é a camada de mais alto nível, que constitui a base da arquitetura de modelagem. Ela se auto descreve, ou seja, os modelos gerados nesta camada são instâncias dela mesmo.

Uma forma mais simples de compreender essa estrutura em camadas é considerar que independente de qual nível da arquitetura você esteja se referindo, um modelo deve estar conforme a seu metamodelo. Diz-se que um modelo está conforme seu metamodelo quando todos os elementos descritos no metamodelo podem ser instanciados pelo modelo [18]. Dessa forma, M0 é uma instância de M1, M1 é uma instância de M2, M2 é uma instância de M3 e M3 é uma instância dele mesmo, pois seria possível haver infinitas dessas relações, mas é senso comum na literatura não passar desse nível de abstração.

Um exemplo prático dessa arquitetura em camada pode ser visto na Figura 2.3. Na camada M0, pode-se observar uma instância da classe *Video*, descrita conforme M1. Na camada M1, pode-se observar que a classe *Video* é uma instância dos elementos *Attribute* e *Class*, descritos em M2 e que esses elementos foram instanciados em M2 como elementos *Class*, que foi descrito na camada M3 e que *Class* se auto descreve.

2.1.3 Transformações entre Modelos

Modelos podem ser criados manualmente por usuários ou ser gerados automaticamente por meio de um processo de conversão de um modelo de origem em um modelo de destino, denominado transformação entre modelos. Dessa forma, a transformação entre modelos é outro elemento essencial dentro das abordagens dirigidas por modelos, pois por meio dela é possível automatizar um ciclo de desenvolvimento de *software*, baseado em modelos [25].

Uma transformação é composta por diversas definições de transformação, que Parreiras [79] define como “um conjunto de regras de transformação, que juntas descrevem como um modelo na linguagem de origem, pode ser transformado em um modelo na linguagem de destino”. Em seguida, ele conceitua regra de transformação como “um ou mais elementos da linguagem de origem que podem ser transformados em um ou mais elementos da linguagem de destino”.

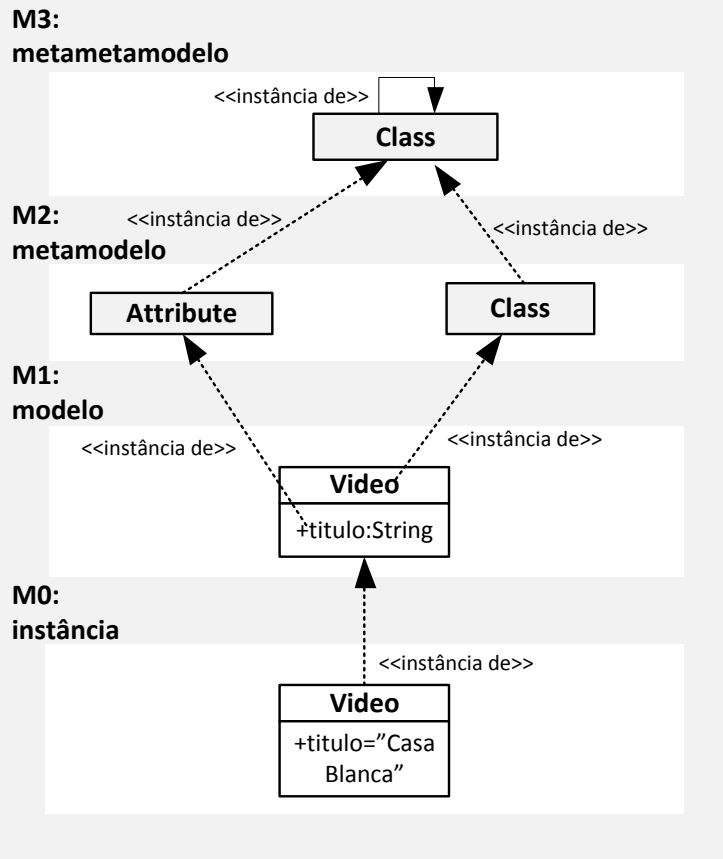


Figura 2.3: Arquitetura de quatro camadas da OMG aplicada para modelos construídos usando UML - baseado em Cabot et al. [18].

A Figura 2.4 apresenta um exemplo de transformação de um modelo de origem em um modelo de destino. Neste exemplo, o objetivo é gerar o *modelo B* a partir do *modelo A*. O primeiro passo é construir as regras de transformação. Estas regras irão descrever como um elemento do *modelo A* irá se transformar em um elemento do *modelo B*. O conjunto dessas regras de transformação formam uma definição de transformação. A transformação é realizada por um motor de transformação. Esse motor recebe como entrada o modelo de origem (*modelo A*) e a definição de transformação e gera como saída, automaticamente, o modelo de destino (*modelo B*).

Dois tipos de transformações são mais comuns em abordagens dirigidas por modelos [25]:

- **Transformações de Modelo para Modelo (M2M)**: Permite transformar um modelo em outro modelo ou em um conjunto de modelos que estejam mais próximos do domínio ou que atendam as necessidades dos interessados;
- **Transformações de Modelo para Texto (M2T)**: Permite transformar um modelo em artefatos, tais como: código fonte, arquivos XML, arquivos de texto, etc.

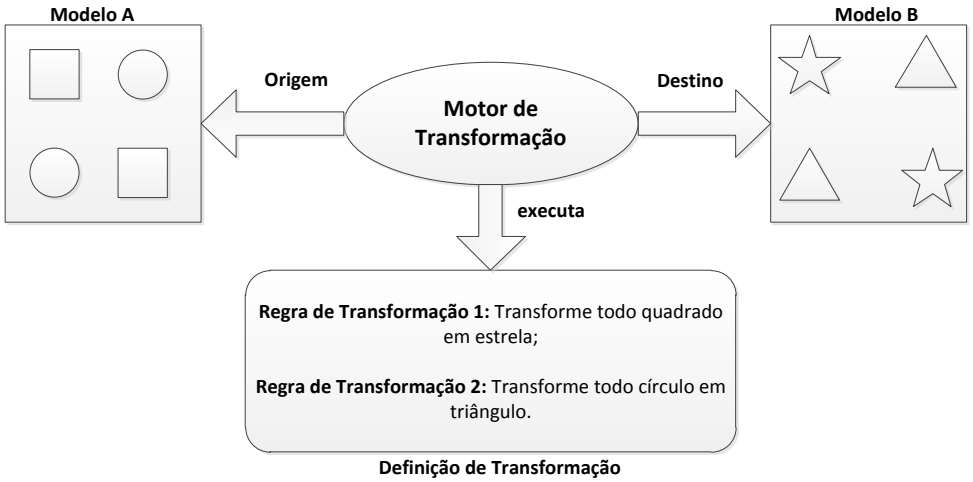


Figura 2.4: Exemplo de uma transformação de um Modelo A em um Modelo B.

As regras de transformação são especificadas através de uma linguagem [25]. Algumas linguagens de transformação são listadas a seguir:

- ***MOF Query/View/Transformation (QVT)***: é uma linguagem híbrida por aceitar construções declarativas e imperativas. Esta linguagem é especificada pela OMG e é formada por três sublinguagens: núcleos, relações e mapeamentos operacionais [77];

- ***MOFScript***: é uma linguagem de transformação que implementa modelo *MOFScript* para texto disponível como *plug-in* para *Eclipse* [68];

- ***Atlas Transformation Language (ATL)***: é uma linguagem híbrida, pois aceita construções declarativas e imperativas, sendo descrita por uma sintaxe básica, textual e gráfica para apresentação de visões parciais das transformações entre modelos [39].

- ***Epsilon Transformation Language (ETL)***: É uma linguagem capaz de transformar um número arbitrário de modelos de origem em um número arbitrário de modelos de destino. O ETL adota um estilo híbrido e possui uma especificação de regras declarativas, usando conceitos avançados, regras abstratas, *lazy* e primárias e resolução automática de elementos-alvo de suas contrapartes de origem. Além disso, como o ETL é baseado em *Epsilon Object Language (EOL)*, reutiliza suas habilidades imperativas para permitir aos usuários especificar transformações particularmente complexas e até interativas. [28]

2.1.4 Arquitetura Dirigida por Modelos (MDA)

Uma abordagem que usa modelos no desenvolvimento de *software* é o conceito que a OMG atribui a MDA[1] [74]. A ideia principal de MDA é criar diferentes modelos em diferentes níveis de abstração e juntá-los para formar uma implementação, sendo que alguns desses modelos serão independentes de plataforma, enquanto outros serão dependentes de plataforma [25].

Segundo Kleppe et al. [51], os principais modelos produzidos por MDA são:

- ***Plataform-Independent Model* - PIM**: é um modelo de alto nível de abstração independente de qualquer tecnologia de implementação que é criado para descrever a regra de negócio;
- ***Plataform-Specific Model* - PSM**: é um modelo específico de plataforma que descreve a regra de negócio do *PIM* com base nos conceitos de uma plataforma específica;
- **Código fonte**: gerado a partir do *Platform-Specific Model (PSM)* para uma linguagem específica para que possa ser compilado e, posteriormente, executado.

Um ciclo de vida para desenvolvimento de *software*, baseado em MDA, é apresentado por Kleppe et al. [51]. Esse ciclo de vida de desenvolvimento de *software* é similar ao ciclo de vida tradicional, mas com algumas diferenças na natureza dos artefatos e como eles são gerados durante o processo. A Figura 2.5 ilustra o ciclo de vida que é descrito da seguinte maneira: Primeiramente, o usuário cria um PIM, que irá descrever a lógica do negócio. Depois, um motor de transformação executa uma definição de transformação de modelos que toma como entrada este PIM e gera um PSM. Uma vez gerado o PSM, outra definição de transformação do tipo modelo-à-código toma como entrada o PSM e gera o código fonte. Essas transformações irão garantir o máximo de fidelidade desde os requisitos até o código durante todo o ciclo

[1] Rodrigues [25] apresenta em seu trabalho uma estrutura hierárquica com as principais abordagens baseadas em modelo. As abordagens são classificadas pelos níveis de abstração alto, médio e concreto. MDE é o nível de abstração mais alto, logo ele posiciona MDA como uma instância de alto nível de MDE. Na instância de abstração média, Rodrigues [25] posiciona as abordagens que oferecem ferramentas que auxiliam os usuários a definirem modelos e transformações. Exemplos de abordagens com nível médio de abstração são *Eclipse Modeling Framework* (Eclipse Modeling Framework (EMF)) e *Microsoft Software Factories*. No nível de abstração concreta, estão as abordagens que oferecem soluções para problemas específicos, por exemplo, *Web Service Software Factory* que permite criar *Web Services*.

de vida. A ideia é que todos os artefatos, durante o ciclo de vida, sejam modelos formais, para que as definições de transformações executadas, através de ferramentas de transformações, gerem artefatos altamente consistentes.

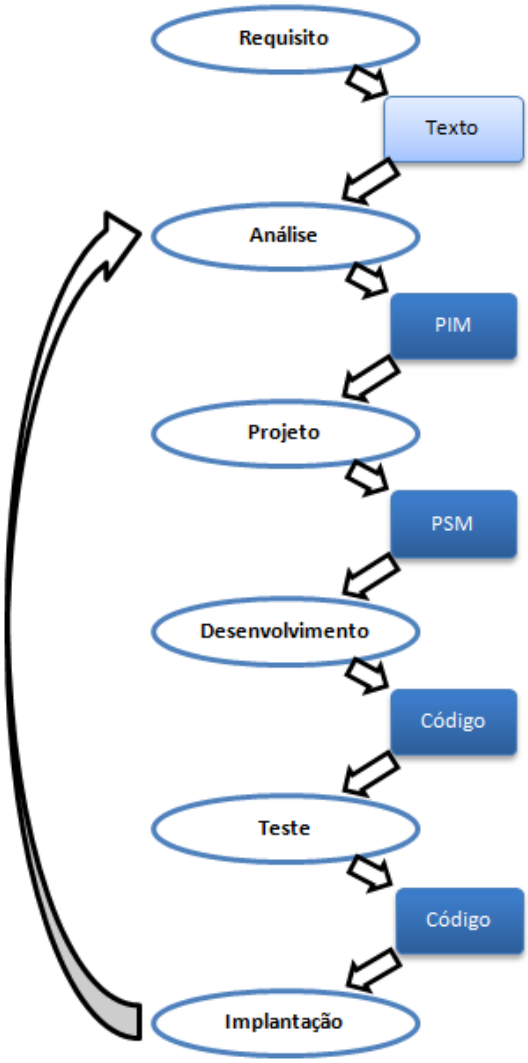


Figura 2.5: Ciclo de Vida do Desenvolvimento com MDA - extraído de Kleppe et al. [51].

A Figura 2.6 apresenta uma visão do processo de transformação utilizado pela abordagem MDA para transformar um modelo independente de plataforma (PIM) em um modelo específico de plataforma (PSM). O motor de transformação apresentado na Figura 2.6 é uma ferramenta que executa as transformações, assim ela recebe como entrada o PIM, que é escrito conforme uma linguagem, e as definições de transformações, que contêm as regras que mapeiam os elementos das duas linguagens de origem e destino, para oferecer como resultado o PSM, que também é escrito conforme uma linguagem.

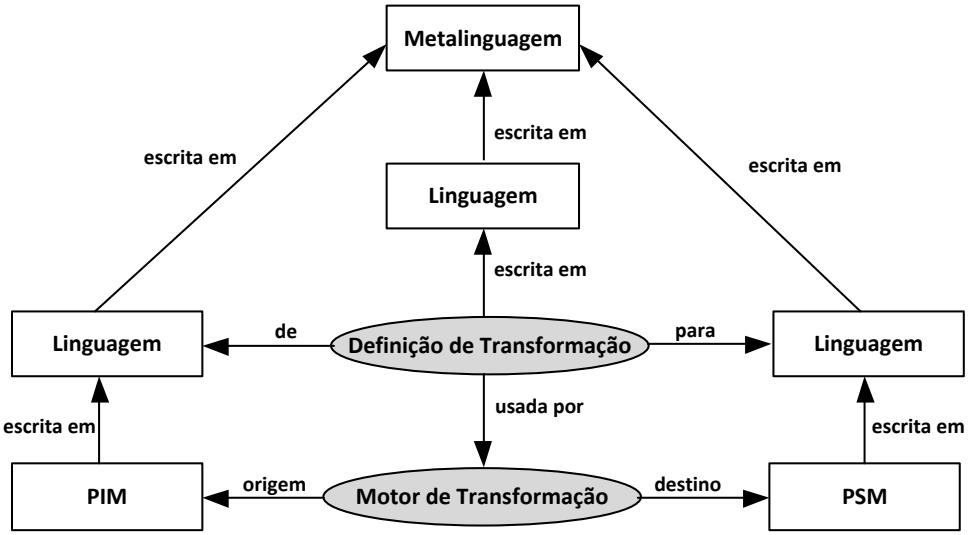


Figura 2.6: MDA - Framework básico de Transformação - baseado em Hammoudi et al. [43].

Um outro modelo, denominado por *PDM*, auxilia na descrição de plataformas, e vem sendo usado na literatura em trabalhos como os de Pablo et al. [63] e de Bézivin et al. [14, 15]. Esse modelo é utilizado para descrever plataformas computacionais, tais como: servidores de aplicações, servidores de banco de dados, *frameworks* e arquitetura de *software* [25].

As principais melhorias no processo de desenvolvimento de *software*, usando MDA, são [69]:

- **Produtividade**: A maior parte do código pode ser gerado a partir de transformações que são criadas apenas uma vez, e executadas quantas vezes forem necessárias;
- **Portabilidade**: Uma regra de negócio modelada, através de um PIM, pode ser transformada para diversos PSMs diferentes;
- **Interoperabilidade**: Geração de pontes entre os modelos;
- **Manutenção e Documentação**: O usuário pode focar seus esforços no PIM, porque ele é uma representação de alto nível do código. Eventuais mudanças no PIM apenas farão com que as transformações para o PSM, e, posteriormente para o código, sejam refeitas.

2.1.5 Conceito de *Weaving* de Modelo

Ruscio [84] afirma que, na modelagem de *software*, é importante isolar as preocupações para evitar a construção de modelos que sejam difíceis de manusear, de manter e de reusar. Além disto, Ruscio [84] ressalta que, em determinado momento, a visão do todo também é importante e que para tanto precisa-se de mecanismos que realizem a integração de modelos.

Fabro et al. [35] expressam o mesmo ponto de vista de Ruscio [84], pois afirmam que com o uso de metamodelos pode-se lidar com domínios distintos ou diferentes níveis de abstração, fazendo com que essa seja uma maneira conveniente de separar preocupações. No entanto, uma vez que as preocupações foram separadas, elas podem ser reagrupadas através de correspondências entre os elementos do modelo, que podem ser utilizadas como base para construir um modelo mais amplo e detalhado da realidade que foi abstraída.

Neste contexto, Fabro et al. [33] promovem o conceito de *weaving* de modelo como sendo "uma operação genérica que estabelece uma correspondência bem definida entre elementos complexos de um modelo". Esse conceito não é novo, e sua aplicação típica é voltada para área de integração e evolução de metadados de banco de dados [84].

Por exemplo, A Figura 2.7 apresenta dois modelos, um que descreve ações que são realizadas em um documento, e um outro que descreve autorizações. Esses dois modelos representam duas realidades diferentes e independentes, mas que podem ser unificadas para gerar um terceiro modelo, que descreve as autorizações necessárias para que uma determinada ação seja executada em um documento. Essa unificação pode ser feita através da correspondência entre os dois modelos. Essa correspondência geralmente é feita de forma direta, interligando cada elemento dos modelos, como pode ser visto na correspondência 2 da Figura 2.7, mas na grande maioria dos casos a situação é mais complexa, como pode ser visto nas correspondências 1 e 3 da Figura 2.7, pois pode haver mais de um elemento do mesmo modelo para interligar com um ou mais elementos do outro modelo. A correspondência 1 mostra que para visualizar ou imprimir um documento basta ter a autorização de visualizador, enquanto na correspondência 2 para copiar um arquivo

basta ser gestor de escrita e, finalmente, na correspondência 3 para apagar um arquivo deve ser gestor de escrita e gestor de leitura.

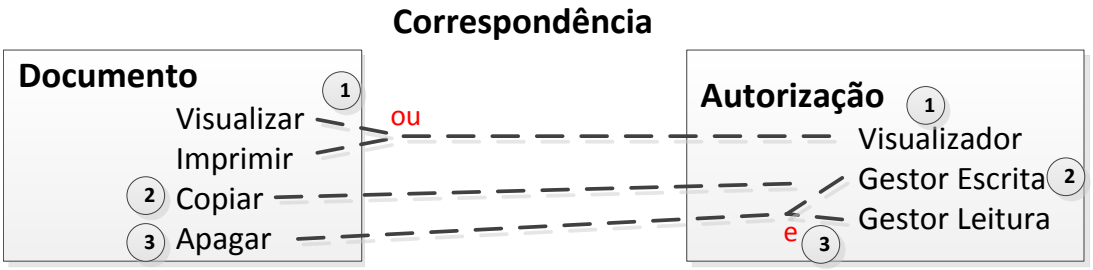


Figura 2.7: Exemplo de uma Operação de *Weaving*.

A operação de *weaving* entre modelos pode ser feita automaticamente através de métodos de correspondência de elementos estruturais ou manualmente através de uma ferramenta apropriada que dê suporte para essa tarefa [34, 62].

2.2 *Big Data*

Big Data é um termo que vem sendo utilizado para determinar um crescimento exponencial dos dados [81]. Na literatura, alguns trabalhos conceituaram *Big Data*, inicialmente, levando em consideração somente o aspecto do tamanho [32]. Por exemplo, Ebach et al. [30] definem *Big Data* como "um termo que se refere a uma grande base de dados", na mesma linha Tankard [97] define *Big Data* como sendo "uma sempre crescente quantidade de informação, que as organizações estão armazenando, processando e analisando, devido ao número crescente de fontes de informação em uso".

A definição de *Big Data* foi evoluindo e ganhando definições mais elaboradas como, por exemplo, a dada pelo centro de tecnologia da informação da Intel [20]: "Um grande conjunto de dados que apresenta uma magnitude (volume) expressiva; mais diversificada, incluindo dados estruturados, não-estruturados e semi-estruturados (variedade), e obtidos mais rápido (velocidade) do que você ou sua organização teve de lidar antes". Emani et al. [32] citam o conceito de Media [65] da seguinte forma: "os dados são muito grandes, movem-se muito rápido ou não se encaixam nas estruturas das arquiteturas de banco de dados".

Na literatura, é comum serem encontradas três características básicas de *Big Data* que justificam tais definições [37, 105]:

- **Volume**: a grande quantidade de dados;
- **Velocidade**: está relacionada com a velocidade de retorno sobre os dados analisados;
- **Variedade**: os dados dentro do contexto de *Big Data* podem apresentar formas estruturadas, semi-estruturadas e não estruturadas.

Segundo Gil [38] e Pereira [81], duas novas características podem ser adicionadas, tais como:

- **Valor**: extrair informações do volume de dados que sejam relevantes para auxiliar na tomada de decisões ou para habilitar novas oportunidades de negócios;
- **Confiabilidade (*Veracity*)**: A informação extraída deve ser confiável e precisa.

A Figura 2.8 apresenta graficamente a evolução das características atribuídas para *Big Data* proposta em Gil [38].

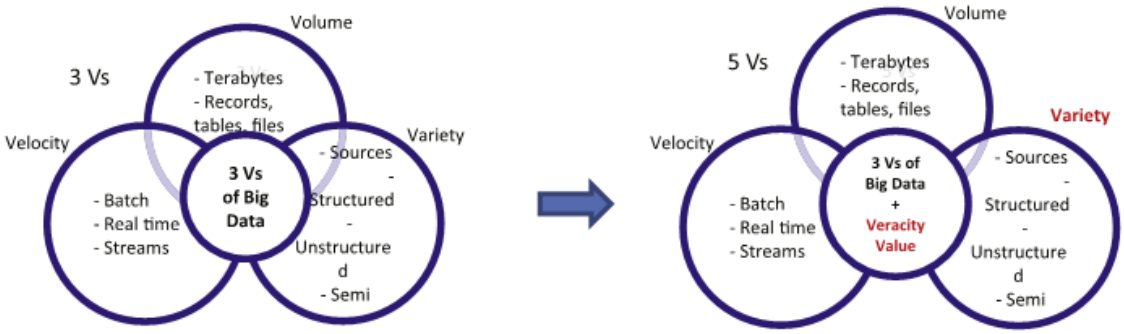


Figura 2.8: Evolução de forma gráfica das características de *Big Data*- extraído de Gil [38].

Emani et al. [32] comentam sobre uma nova característica, além das cinco definidas, a visualização. Essa característica levanta a importância de prover ferramentas que auxiliem na visualização dos resultados encontrados.

Big Data vem se tornando um termo cada vez mais popular em praticamente todos os aspectos da sociedade, incluindo empresas, governo, saúde e pesquisa em quase todas as áreas: ciências da vida, engenharia, ciências naturais, etc [45, 105]. Uma leitura interessante sobre *Big Data* é encontrada no trabalho de Jagadish [45], que desmistifica alguns mitos que foram criados ao longo do tempo sobre *Big Data*.

2.2.1 Desenvolvimento para *Big Data*

Big Data envolve um grande volume de dados e para tanto precisa lidar com técnicas que permitam realizar processamento paralelo [100]. Desse modo, a técnica mais utilizada é baseada nos princípios de dividir para conquistar, e pode ser resumida em duas fases: *Map* e *Reduce* [32, 59]. Lin [57] conceitua *MapReduce* como "uma instância específica de uma classe geral de linguagens, que realizam processamento paralelo, em que os cálculos são conceitualizados como grafos direcionados, onde os vértices representam operações em registros que flutuam ao longo das bordas direcionadas".

O funcionamento das fases de *Map* e de *Reduce* poderia ser descrito, de maneira simples, da seguinte forma: A fase de *Map* recebe um conjunto de dados de entrada. Esses dados são divididos conforme um procedimento a ser executado que retorna um novo par de chaves e valores. Na fase *Reduce*, os conjuntos de chaves e valores gerados pela fase de *Map* são unidos conforme procedimento a ser executado. O retorno da fase de *Reduce* gera uma ou mais saídas, dependendo do problema a ser resolvido.

Por exemplo, no aplicativo de contar palavras, a fase de *Map* teria o papel de dividir as palavras do texto, formando pares de chaves e valores (palavra, frequência), enquanto a fase de *Reduce* teria o papel de receber esses pares de chaves e valores e somá-los para obter as frequências das palavras divididas, dando o total geral de cada palavra, como pode ser visto na Figura 2.9. O texto contendo palavras foi dividido em três partes, onde cada parte foi encaminhada para uma função *Map*. As funções *Map* processam os textos enviados, formando os conjuntos de pares de chaves e valores que depois são repassados para as funções *Reduce*. A fase *Reduce* irá processar os resultados conforme o processamento desejado, que nesse caso é a soma das frequências das palavras. A saída é o conjunto de pares de chaves e valores sumarizados pela fase *Reduce*.

Desenvolver *software* para *Big Data* é complexo, devido a grande quantidade de variáveis envolvidas para construir uma solução computacional (rede, compartilhamento de recursos, paralelismo, etc), mas existem ferramentas e/ou *frameworks* que encapsulam essa complexidade. Os *frameworks* mais comuns, por

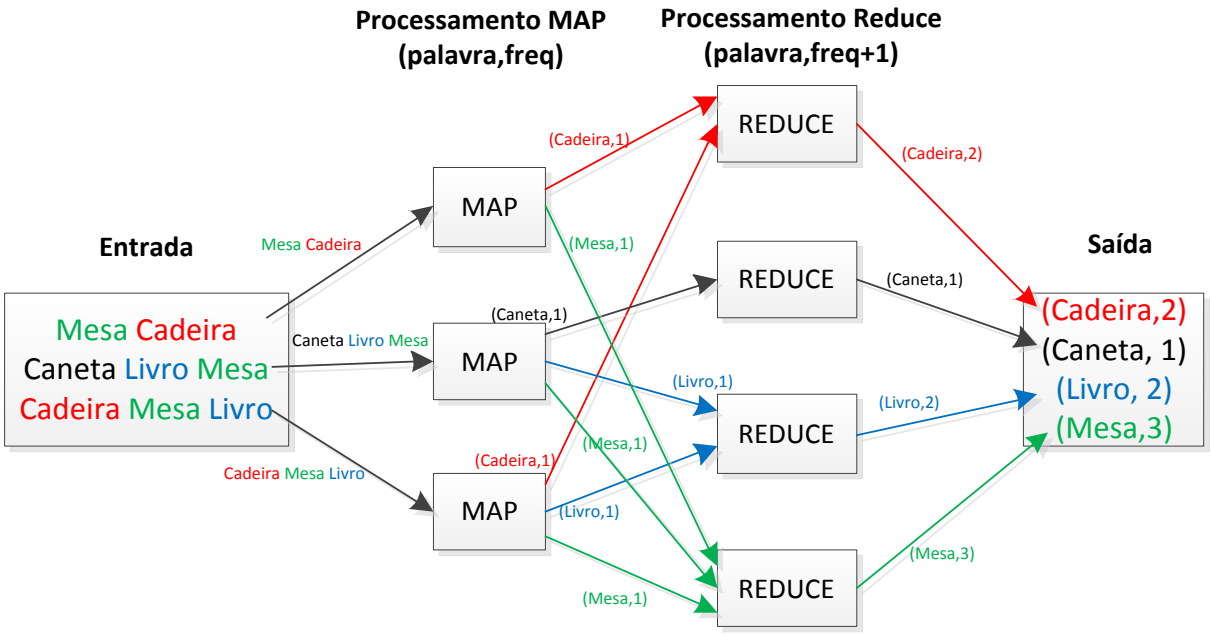


Figura 2.9: Exemplo de execução das funções Map e Reduce.

exemplo, são: a plataforma *Hadoop* [59] e o *framework Spark* [49]. Os conceitos de *Map* e *Reduce* são implementados por ambos de maneiras distintas [57].

Apache *Hadoop*

Hadoop é uma plataforma que fornece soluções para armazenamento distribuído e para aumentar o poder computacional, utilizando uma arquitetura mestre-escravo [44]. De maneira simplista, pode-se afirmar que cada nó na rede é composto por dois componentes: *MapReduce*[2] [3] e *Hadoop Distributed File System (HDFS)*[4]. O primeiro componente é responsável pelo processamento, enquanto o segundo é responsável pelo armazenamento. Um nó é responsável por gerenciar o processamento e o armazenamento, este é denominado mestre, e os demais que são gerenciados são denominados escravos. A Figura 2.10 representa uma visão simplificada da arquitetura mestre/escravo da plataforma *Hadoop*.

Com a utilização destes dois componentes, toda a complexidade de infraestrutura envolvida é omitida, permitindo que soluções complexas possam ser desenvolvidas de forma distribuída [59].

[2] A plataforma *Hadoop* possui um *framework* chamado *MapReduce* responsável pelo processamento dos dados e esse *framework* implementa o modelo *MapReduce* de desenvolvimento.

[3] "é uma estrutura para a execução de algoritmos altamente paralelizáveis e distribuíveis através de grandes conjunto de dados usando um grande número de computadores." [59]

[4] é projetado para armazenar uma grande quantidade de dados e prover acesso a esses dados para muitos clientes distribuídos ao longo de uma rede de computadores [59]

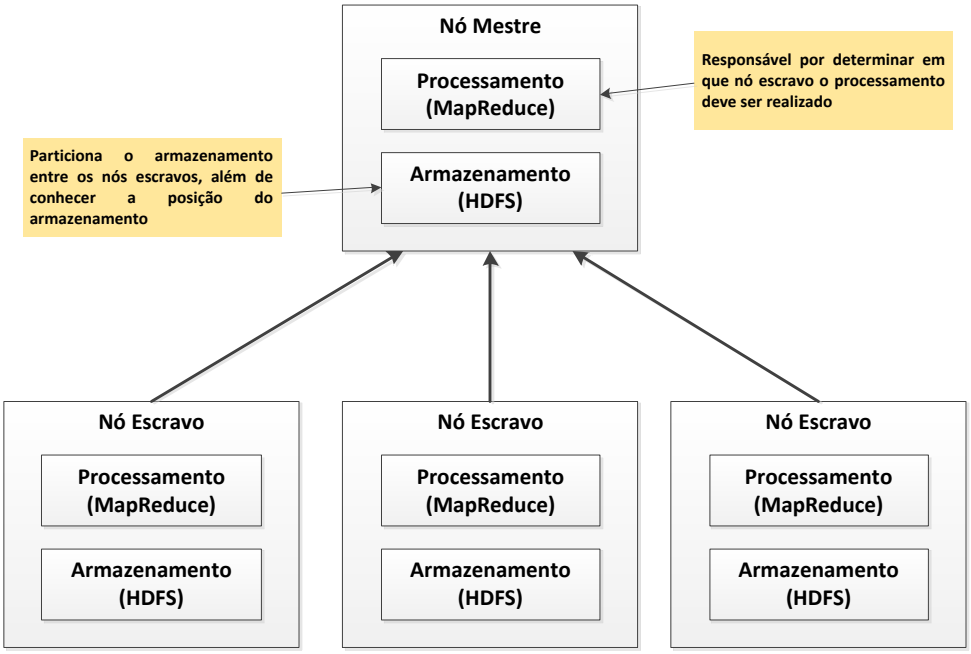


Figura 2.10: Arquitetura simplificada de Hadoop- extraído de Holmes [44].

Apache *Spark*

Apache *Spark* "é um *framework* rápido e geral para o processamento de dados em grande escala" [7]. Ele fornece APIs de alto nível para as linguagens *Scala*, *Java* e *Python*, para que possam fazer trabalhos paralelos e fáceis de serem escritos [8].

O *framework Spark* não oferece uma solução própria de armazenamento, visto que seu propósito principal é realizar apenas processamento dentro de um *cluster*. Entretanto, segundo Penchikala [80], *Spark* utiliza o sistema de arquivos HDFS e toda e qualquer fonte de dados que sejam compatíveis com *Hadoop* para realizar o armazenamento de dados, como por exemplo: HDFS, Cassandra [4], HBase [5], Hive [6], etc.

Em relação ao processamento, seu objetivo é incrementá-lo. *Spark* é comparável ao *MapReduce* da *Hadoop* em termo de funcionalidade, mas em termo de performance *Spark* é até 100x mais rápido quando executa aplicações em memória e até 10x mais rápido quando executa aplicações em disco [7, 80]. Essa diferença de performance é justificável, pois as saídas de cada etapa em *MapReduce* devem ser armazenadas no sistema de arquivos antes da próxima etapa iniciar [80], enquanto em

Spark é utilizado o conceito de uma abstração chamada de *Resilient Distributed Datasets (RDD)*[5] [104].

2.3 Síntese

Nesse capítulo, apresentou-se uma revisão dos principais conceitos envolvidos e utilizados para o desenvolvimento do presente trabalho, com o intuito de favorecer um melhor entendimento do *framework* proposto.

Os principais conceitos de MDE e suas aplicações foram apresentados. Discorreu-se também sobre os conceitos de MDA e seus benefícios como produtividade, portabilidade e interoperabilidade, além do conceito de *weaving* de modelos.

Fechando o capítulo, contextualizou-se *Big Data* e citou-se suas principais características, além de ter sido apresentada uma visão geral sobre *Hadoop* e *Spark*, que são soluções amplamente utilizadas no desenvolvimento de *software* para *Big Data*.

[5] "RDD são estruturas de dados distribuídas tolerantes a falhas, que permitem aos usuários persistirem explicitamente seus resultados intermediários na memória, controlarem seu particionamento para otimizar o posicionamento de dados e os manipularem, usando um rico conjunto de operadores" [104].

3 Estado da Arte

Este capítulo tem como objetivo descrever as pesquisas mais recentes sobre Engenharia de *Software* aplicada a *Big Data*. Posteriormente, uma discussão sobre os trabalhos relacionados irá finalizar este capítulo. Essa discussão será realizada, analisando e comparando alguns critérios estabelecidos como base para o desenvolvimento deste trabalho.

3.1 Engenharia de *Software* para *Big Data*

A engenharia de *software* aplicada a *Big Data* ainda é uma questão recente e apresenta uma série de desafios a serem vencidos. Nesta seção, alguns trabalhos que enumeram estas oportunidades e desafios serão apresentados. No Apêndice A, alguns trabalhos que descrevem a utilização de MDE e de *Big Data* nas suas mais diversas aplicabilidades estão catalogados.

3.1.1 *Research Directions for Engineering Big Data Analytics Software* [78]

Otero et al. [78] destacam quatro desafios: especificar os requisitos, projetar e construir o *software*, além de testar se o *software* está de acordo com os requisitos. Para especificação de requisitos, a utilização de um modelo matemático com o objetivo de facilitar a verificação, é proposto; para o projeto, ele aponta diversas qualidades que um *software*, para análise de *Big Data*, deve apresentar: usabilidade, performance, confiabilidade, disponibilidade, segurança, interoperabilidade, escalabilidade e teste. Entretanto, os autores só discorrem sobre confiabilidade e segurança. Na construção, Otero et al. [78] apontam os desafios mais comuns na literatura: negligência, manutenção, depuração e reprodutibilidade. Apontam, também, que a construção de tais *software* têm uma complexidade razoável, porque envolvem diversas tecnologias.

Com relação aos testes, Otero et al. [78] recomendam *metamorphic testing*, pois permite o teste e a validação do aprendizado de máquina.

3.1.2 *Embrace the Challenges: Software Engineering in a Big Data World* [3]

Anderson [3] reconhece que o projeto e o desenvolvimento de *software* para *Big Data* são complexos e possuem diversos desafios. Dessa maneira, ele apresenta seis desafios para engenharia de *software* associada a *Big Data*, através da experiência obtida com a construção de dois projetos EPIC *collect* e EPIC *analysis*. Esses projetos visam investigar como as pessoas estão usando as redes sociais em tempo de crise. Durante o desenvolvimento dos projetos, ele encontrou os seguintes desafios: a falta de ferramentas que dessem suporte aos desenvolvedores para projetar e desenvolver; a importância das equipes multidisciplinares; o uso de ciclos de vida altamente iterativos; a necessidade de escolher adequadamente os *frameworks* disponíveis para *Big Data*; operações simples se transformam em desafios significativos em uma escala maior e a necessidade de obter modelos de dados adequados para produzir sistemas que sejam escaláveis, robustos e eficientes. Anderson [3] espera que com os desafios levantados consigam demonstrar a complexidade envolvida em projetos para *Big Data*, e que algumas das decisões tomadas possam auxiliar os engenheiros de *software* e os pesquisadores a resolverem seus problemas.

3.1.3 *Research Opportunities for the Big Data Era of Software Engineering* [27]

DeLine [27] afirma que o desenvolvimento de *software* para *Big Data* está se tornando uma prática generalizada. Assim, equipes multidisciplinares estão surgindo, por exemplo, estatísticos e analistas de dados estão trabalhando diretamente com profissionais da área de computação, como: desenvolvedores, testadores e gerentes de programas. Para DeLine [27], esse momento é crucial para que os pesquisadores da área de engenharia de *software* possam aproveitar para estudar, compreender e melhorar suas práticas. Dessa forma, em seu trabalho, DeLine [27] cita três grandes oportunidades: a primeira relacionada à produtividade, a segunda relacionada à

confiabilidade e a terceira relacionada à comunicação e à coordenação. Em termos de produtividade, ele acredita que devem ser oferecidas ferramentas que permitam o menor tempo de resposta possível, e que automatizem o máximo o processo. Sobre confiabilidade, os pesquisadores devem oferecer ferramentas para ajudar as equipes de *software* a ter confiança nos resultados alcançados. Assim, ele sugere que os dados distribuídos sejam tratados como um contrato implícito para que anomalias sejam procuradas. Finalmente, para comunicação e coordenação, ele afirma que as equipes precisam de maneiras mais acessíveis para discutir as incertezas, os riscos e o suporte à tomada de decisão baseada nos dados.

3.1.4 *Big Picture of Big Data Software Engineering: With Example Research Challenges* [60]

Madhavji et al. [60] lembram que as atividades chaves para especificação de requisitos são: criação e compreensão de modelos de domínio da aplicação, elicitação de requisitos com os interessados, desenvolvimento de modelos comportamentais e funcionais, validação, etc. Alertam que a especificação de requisitos ainda é bem emergente e que uma compreensão mais clara é necessária para separar requisitos: de infraestrutura, de técnicas e de ferramentas analíticas e de usuários finais para uso do *Big Data*. Sobre arquitetura, eles comentam que existem soluções isoladas e *frameworks* com infraestruturas robustas para análise de *Big Data*, mas ainda é necessário investir em padrões e arquiteturas de referência que permitam a integração de todas estas infraestruturas. Em relação aos testes, salientam a complexidade que envolve soluções *Big Data*, devido ao número de variáveis existentes e dinamismo entre elas. Assim, levantam a seguinte questão: como construir um sistema de teste representativo, usando recursos de *hardware* e *software* que represente uma fração dos dados do sistema de produção? Sugere três soluções: *Scaling down resources*, *Scaling data representatively*, *Workload capture-replay*.

3.1.5 *Early Experience with Model-driven Development of MapReduce based Big Data Application* [82]

Rajbhoj et al. [82] relatam a complexidade da utilização das tecnologias existentes para análise de *Big Data* mesmo por parte de desenvolvedores intermediários. Dessa forma, apresentam uma proposta prática para o desenvolvimento de *software* para *MapReduce* de *Big Data*. A abordagem utilizada é Model-Driven Development (MDD), que é um paradigma de desenvolvimento que usa modelos como artefatos primários para o processo de desenvolvimento. O objetivo do trabalho é eliminar a complexidade do desenvolvimento de *MapReduce*, facilitando o desenvolvimento e aumentando a produtividade. Para tanto, foi criado um metamodelo que permite especificar a estrutura do programa de aplicação analítica em um nível mais alto de abstração.

O metamodelo apresenta classes como a classe *AnalyticsTask* que representam os problemas que devem ser resolvidos. A classe *AnalyticsTask* pode conter vários *Job*, que são classes que representam sub tarefas que devem ser executadas. A sequência de execução é determinada pela associação *nextJob*. Um *Job* necessita também dos arquivos de entrada representados pela classe *DataFile* e os arquivos de entrada precisam descrever a formatação, que é descrita pela classe *StreamFormat*. O processamento que o *Job* deve executar são representados pelas classes *Mapper* e *Reducer*. O metamodelo proposto é específico para soluções que envolvam aprendizado de máquina, por isso ele apresenta a classe *KMeanClusterJob*. O metamodelo proposto pelos autores pode ser observado na Figura 3.1.

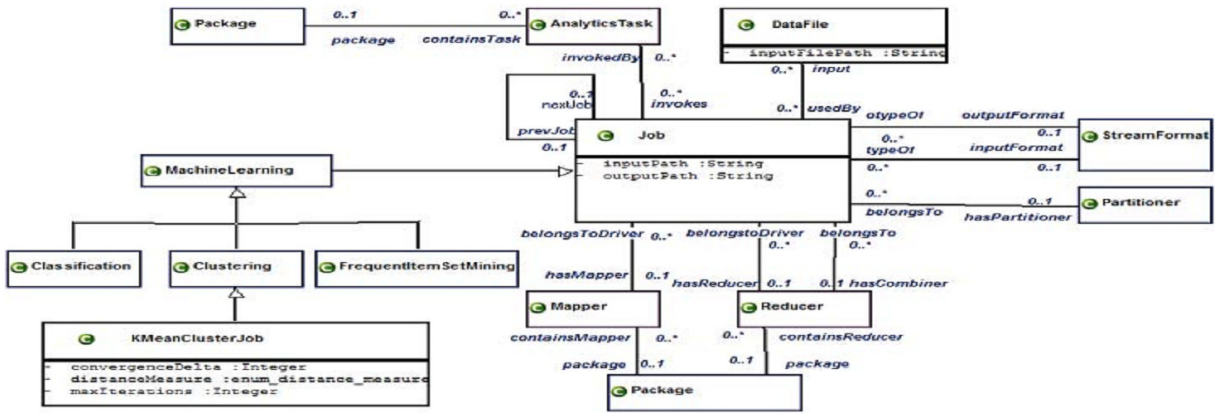


Figura 3.1: Metamodelo para MapReduce - Extraído de Rajbhoj et al. [82].

Nesse trabalho, Rajbhoj et al. [82] geram automaticamente apenas o esqueleto do código fonte, para que a lógica de negócio possa ser encorporada, posteriormente, de forma manual. A geração é feita através de uma transformação do tipo modelo para texto que se preocupa em gerar o código, conforme as exigências da API de *MapReduce* para execução de uma tarefa.

Para demonstrar seu trabalho, utiliza um exemplo ilustrativo chamado *Internet Protocol Television (IPTV)*. IPTV é um sistema em que o serviço de televisão digital é entregue, usando a arquitetura da Internet e métodos de rede. A ideia é mapear os perfis dos consumidores para identificar produtos que sejam do interesse deles.

Segundo Rajbhoj et al. [82], o desenvolvimento desse exemplo passou de aproximadamente 7 semanas para 2 semanas, após a aplicação de MDD.

3.1.6 *Big Data System Development: An Embedded Case Study with a Global Outsourcing Firm* [21]

Chen et al. [21] levantam a questão sobre o desafio de desenvolver *software* para *Big Data*. Como atender os 5V de *Big Data* (Velocidade, Veracidade, Volume, Variedade e Valor)? Qual tecnologia utilizar? Como lidar com cenários de constante mudança? Essa preocupação é relevante, pois segundo os autores 55% dos projetos *Big Data* não foram concluídos. Chen et al. [21] focam em seu trabalho questões relacionadas às fases de requisitos e de projeto.

Chen et al. [21] apresentam uma proposta prática para o desenvolvimento de *software* baseado em *Big Data*. O trabalho propõe um novo método para combinar automaticamente projetos de arquitetura e abordagens de modelagem de dados, estendendo um método conhecido como *Attribute Driven Design (ADD)* [88]. O método proposto foi denominado de *Big Data system Design method (BDD)*. A Figura 3.2 apresenta o modelo do processo utilizado pelo BDD. Esse processo se concentra apenas nas fases de Análise de Requisitos (AR) e Projeto procurando integrar tarefas de análise de arquitetura, técnicas de modelagem de dados e seleção de tecnologias. De maneira geral, na fase de AR, as tarefas são concentradas nas regras de negócios e na definição da qualidade dos atributos. Na fase de projeto, uma extensão do método ADD foi

concebida para permitir a integração de técnicas de análise de modelagem de dados. Esta fase é direcionada pela seleção de tecnologias em três repositórios que concentram respectivamente: arquiteturas de referência e *frameworks*; padrões de projetos, táticas e modelos de dados e, finalmente, um catálogo de tecnologias *Big Data*.

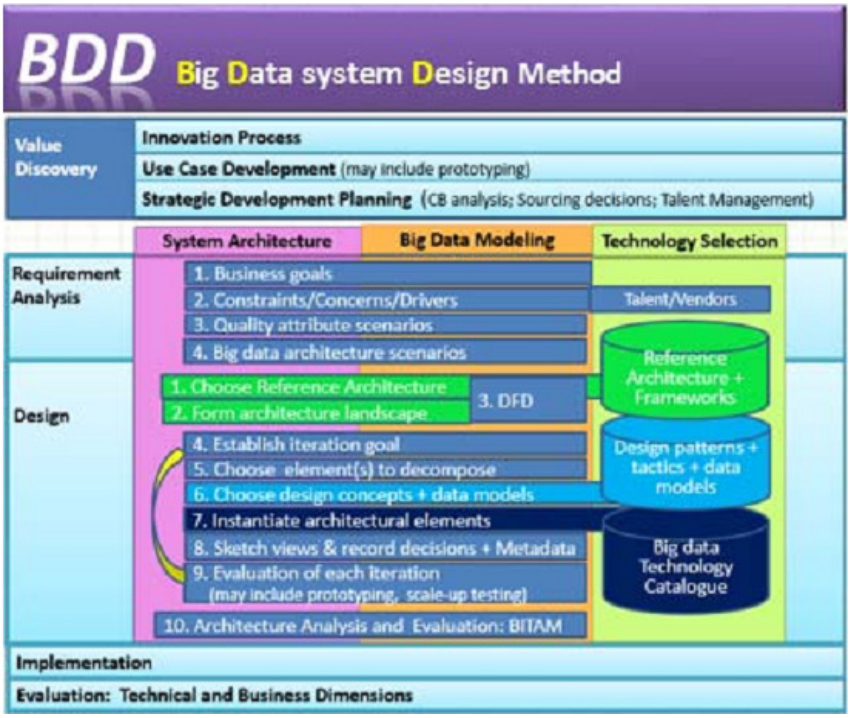


Figura 3.2: Modelo do Processo BDD - Extraído de Chen et al. [21].

Esse trabalho foi construído com a colaboração técnica de uma empresa chamada *Softserve*. Essa empresa foi escolhida por estar aberta a inovações e por oferecer estudos de casos voltados para *Big Data*. A validação do trabalho ocorreu através da aplicação do método em alguns estudos de casos oferecidos pela empresa.

3.1.7 *Towards a Model-driven Design Tool for Big Data Architectures* [40]

Guerriero et al. [40] ressaltam que as tecnologias *Big Data* estão se tornando indispensáveis para indústrias modernas, mas alertam que a aplicação dessas tecnologias possui um custo considerável. Eles destacam que um dos principais custos relacionados a utilização de *Big Data* estão associados ao tempo gasto para aprender a projetar soluções com os *frameworks*, utilizados no mercado. Afirmam ainda, que o custo associado com esse aprendizado poderia ser sanado, utilizando MDE para o desenvolvimento de aplicações *Big Data*.

Guerriero et al. [40] propõem uma arquitetura para a criação de projetos dirigidos por modelos para *Big Data* que pode ser vista na Figura 3.3. A arquitetura proposta contém três componentes: componente DPIM que permite descrever as características da arquitetura *Big Data*; O componente DTSM que permite a avaliação de múltiplos *layouts* e alternativas de arquitetura e o componente DDSM que permite produzir um mapa de implantação para a visão implementável do projeto de aplicativo de *Big Data* realizado e refinado dentro do componente DTSM. A concepção dos componentes DPIM e DTSM foram realizados através do estudo de conceitos e de ferramentas/*frameworks* amplamente utilizados.

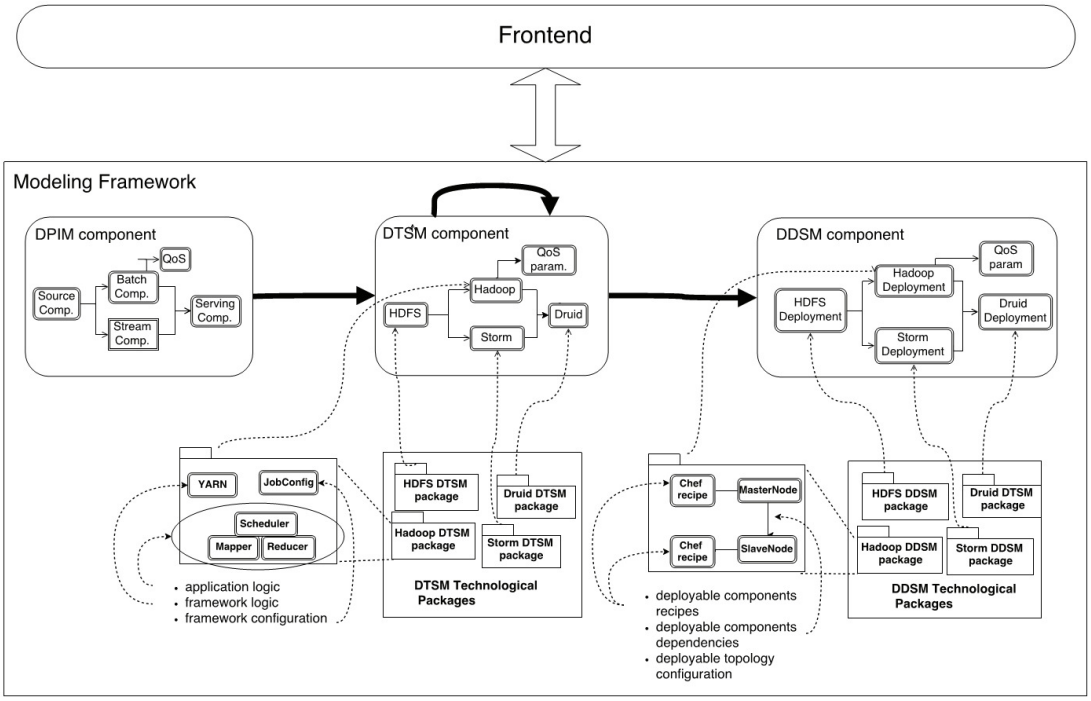


Figura 3.3: Uma arquitetura para design de *Big Data* orientado por modelo - Extraído de Guerriero et al. [40].

Cada um dos componentes possui um modelo, isto é, o componente DPIM tem um modelo DPIM, o componente DTSM tem um modelo DTSM, e o componente DDSM tem um modelo DDSM. O modelo do DPIM é criado manualmente, e os demais modelos são gerados através da execução de transformações. Todos os modelos da ferramenta são gerados conforme os metamodelos propostos. Por exemplo, um fragmento do metamodelo DPIM é apresentado na Figura 3.4.

Guerriero et al. [40] validam sua arquitetura, usando o exemplo ilustrativo de contar palavras implementado conforme a plataforma *Hadoop*. Os autores concluem

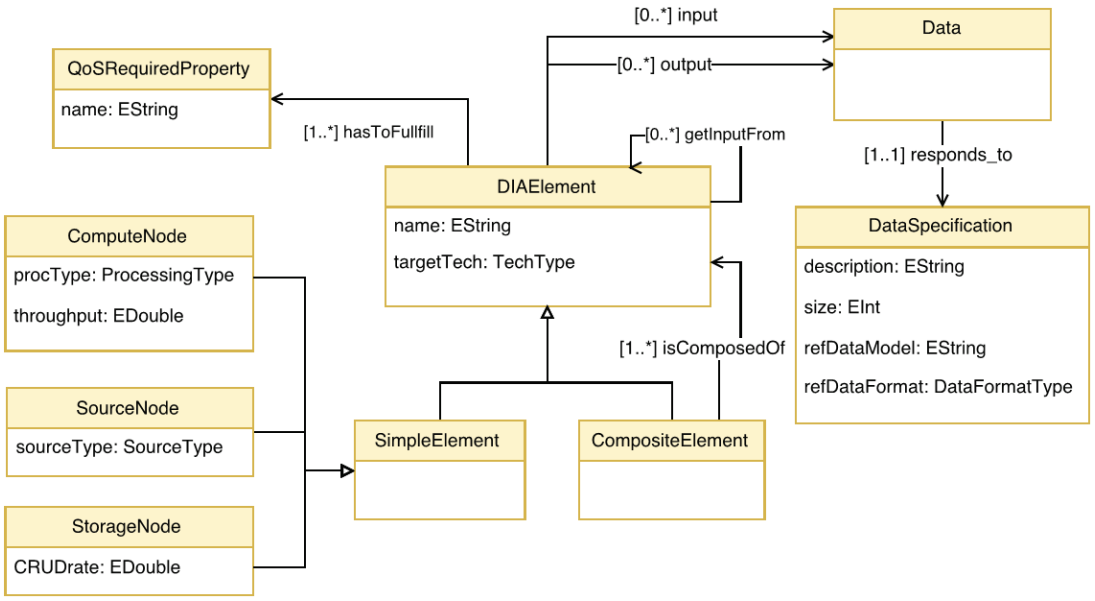


Figura 3.4: Fragmento do metamodelo para o componente DPIM - Extraído de Guerriero et al. [40].

que MDE tem um grande potencial para auxiliar na concepção de projetos e no desenvolvimento de aplicações *Big Data*.

3.1.8 Discussão

A engenharia de *software* aplicada a *Big Data* é emergente e recente. Todos os trabalhos apresentados na Seção 3.1 trazem contribuições importantes dentro desse contexto. Alguns fazem apenas discussões entre as oportunidades e desafios que necessitam de soluções, enquanto outros tentam apresentar soluções concretas. Assim, para que haja uma compreensão adequada da potencialidade do trabalho que vem sendo proposto, alguns critérios de avaliação foram estabelecidos.

Dentre os itens avaliados, citam-se:

1. Trabalho Teórico ou Prático: Esclarece se a proposta é apenas uma discussão ou se oferece alguma solução concreta para auxiliar no desenvolvimento de *software* para *Big Data*;

2. Abordagem Principal: Abordagem central utilizada para fundamentar a solução proposta;

3. Usa modelos como artefatos de desenvolvimento: Se o trabalho utiliza modelos como artefatos para auxiliar em alguma fase do processo de desenvolvimento;

4. Auxilia na fase de Análise: Se o trabalho oferece algum tipo de ferramenta que permita auxiliar na tarefa de análise do problema a ser solucionado;

5. Auxilia na fase de Projeto: Se o trabalho oferece algum tipo de ferramenta que permita auxiliar na tarefa de projeto do problema a ser solucionado;

6. Auxilia na fase de implementação: Se o trabalho oferece algum tipo de ferramenta que permita auxiliar na tarefa de implementação do problema a ser solucionado;

7. Gera o código fonte completo: Se o trabalho gera o código fonte a partir da lógica do negócio de forma completa, ou seja, definição da estrutura do código fonte mais o comportamento dos métodos e não só a definição dos mesmos;

8. Auxilia na manutenção/evolução: Se o trabalho oferece algum mecanismo que auxilie diretamente na manutenção/evolução do código;

9. Auxilia na documentação: Se o trabalho oferece algum mecanismo que auxilia na geração de algum nível de documentação;

10. Realizou estudo experimental: Se o trabalho realizou algum estudo experimental da solução proposta com terceiros.

Esses critérios de avaliação foram escolhidos para que seja realizada a comparação das características do *framework* proposto com os trabalhos relacionados disponíveis na literatura e pré-selecionados neste trabalho. O critério *Trabalho Teórico ou Prático* terá como resposta "T", quando o trabalho for de cunho teórico, e a resposta "P", quando o trabalho apresentar soluções concretas. O critério *Abordagem Principal* terá como resposta "Não oferece", se o trabalho não apresentar uma abordagem central que o fundamente ou o nome da abordagem, caso o trabalho apresente. Enquanto os critérios *auxiliam na fase de análise, projeto, implementação e documentação* terão como resposta "Automática", "Manual" ou "Não". A resposta "Automática" ou "Manual" serão atribuídas caso o trabalho possua ferramentas para auxiliar no desenvolvimento. O termo "Automático" será empregado para determinar que a ferramenta realiza algum nível de automaticidade através do acionamento do usuário, e o termo "Manual" determina que não existe automaticidade, pois a ferramenta

apenas auxilia o usuário através do seu uso. Os demais critérios terão como resposta "Sim" quando a resposta for positiva e "Não" quando a resposta for negativa.

A Tabela 3.1 apresenta um quadro comparativo entre os trabalhos relacionados, utilizando os critérios de avaliação pré-determinados.

Os trabalhos de Otero et al. [78], Madhavji [60], Anderson [3] e DeLine et al. [27] são classificados como teóricos, pois realizam discussões sobre problemas e oportunidades relacionados à engenharia de *software* aplicada a *Big Data*, mas não oferecem uma solução prática, ou seja, demonstrada. Em seus trabalhos, abordagens centrais não são encontradas, mas sim um conjunto de sugestões de como se poderia resolver determinados problemas nesse contexto. Tão pouco são oferecidas ferramentas que auxiliem, mesmo que em alguns trabalhos seja comentado como elas são importantes para auxiliar na atividade de desenvolvimento.

Enquanto, nos trabalhos de Chen et al. [21] e Guerriero et al. [40], soluções práticas para algumas das tarefas da atividade de desenvolvimento de *software* para *Big Data* são apresentadas. Ambos oferecem soluções para as tarefas de análise e projeto. O primeiro utiliza uma abordagem baseada em ADD e realiza o processo de forma manual, em contra partida, no segundo trabalho, uma abordagem baseada em MDE é utilizada, tendo seu processo realizado de forma manual para a análise, e automática para o projeto, através do uso de transformações entre modelos. Nenhum dos dois trabalhos oferecem soluções para implementação do código, seja ele somente com as definições dos métodos (esqueleto), seja ele completo (definição mais comportamento dos métodos). O trabalho de Guerriero et al. [40] tem potencial de gerar o código a nível de definições, no entanto, em seu trabalho não está claro se realmente o código é gerado.

O trabalho de Rajbhoj et al. [82] é outro trabalho que apresenta uma solução prática, por meio do uso de MDD, para o contexto de desenvolvimento de *software* para *Big Data*. A tarefa de análise não é contemplada no trabalho, pois não foi apresentado nenhum modelo independente de plataforma para descrever o problema. No entanto, a tarefa de projeto e implementação são contempladas. A tarefa de projeto é contemplada através de um metamodelo específico para plataforma *MapReduce*, implementada por *Hadoop*, enquanto a tarefa de implementação é contemplada por meio da execução de transformações de modelo para texto. Essas transformações

geram apenas o esqueleto do código fonte, ou seja, a regra de negócio será incorporada posteriormente de forma manual, através da inclusão do comportamento dos métodos criados.

Ao analisar os trabalhos relacionados, percebe-se a necessidade de uma solução que possa suportar, além das tarefas de análise e projeto, a implementação do código fonte de maneira completa, assim como oferecer auxílios em outras tarefas pertinentes ao *software* desenvolvido, como: manutenção, evolução e documentação.

3.2 Síntese

Neste capítulo, sete trabalhos relacionados à Engenharia de *Software* aplicada para *Big Data* foram apresentados. Uma tabela comparativa entre esses trabalhos foi preenchida, utilizando os critérios avaliativos definidos. Encerra-se o capítulo com uma discussão sobre os trabalhos relacionados com base nos critérios elencados.

Na Seção 7.13, uma comparação entre os trabalhos relacionados à Engenharia de *Software* para *Big Data* e a nossa proposta é apresentada.

Tabela 3.1: Análise dos Trabalhos relacionados

Itens Avaliados	Research Directions for Data Engineering Big Analytics Software [78]	Embrace the Challenges: Software Engineering in a Big Data World [3]	Research Opportunities for the Big Data Era of Software Engineering [27]	Big Picture of Big Data Software Engineering: With Example Research Challenges [60]	Early Experience with Model-driven Development of MapReduce based Big Data Application [82]	Big Data System Development: An Embedded Case Study with a Global Outsourcing Firm [21]	Towards a Model-driven Design Tool for Big Data Architectures [40]
Trabalho Teórico ou Prático	T	T	T	T	P	P	P
Abordagem Principal	Não oferece	Não oferece	Não oferece	Não oferece	Baseado em MDD	Baseado em ADD	Baseado em MDE
Usa modelo como artefato de desenvolvimento	Não	Não	Não	Não	Sim	Sim	Sim
Auxilia na fase de Análise	Não	Não	Não	Não	Não	Manual	Manual
Auxilia na fase de Projeto	Não	Não	Não	Não	Manual	Manual	Automático
Auxilia na fase de implementação	Não	Não	Não	Não	Automática	Não	Não
Gera o código fonte completo	Não	Não	Não	Não	Não	Não	Não
Auxilia na Manutenção/ Evolução	Não	Não	Não	Não	Não	Não	Não
Auxilia na Documentação	Não	Não	Não	Não	Manual	Manual	Manual
Realizou estudo experimental	Não	Não	Não	Não	Não	Não	Não

4 *Framework* para Desenvolvimento de *Software* para *MapReduce* de *Big Data* (F2BD)

Inúmeros *software* estão sendo desenvolvidos para *Big Data*. Entretanto, na literatura, poucos *frameworks* que suportem a atividade de desenvolvimento (análise, projeto e implementação) para *Big Data* são encontrados. Atualmente, muitas discussões sobre os desafios e as oportunidades voltados para a área da engenharia de *software*, aplicada a *Big Data*, são encontradas na literatura [3, 27, 60]. Nesse capítulo, a proposta de um *framework*, baseada em MDE e *Weaving* é apresentada, unindo teoria e prática para suportar a atividade de desenvolvimento de *software* para a plataforma *Big Data* que usem o modelo *MapReduce*.

O capítulo inicia apresentando a fundamentação teórica que deu origem ao *framework* proposto. Posteriormente, a arquitetura do *framework* será apresentada, seguida da apresentação dos metamodelos criados e estendidos. Em seguida, uma notação gráfica, assim como, a definição de uma padronização, para auxiliar na criação do PIM, será apresentada. Logo após, um processo, que orienta a utilização do *framework* proposto, será apresentado. Por fim, uma discussão sobre as principais contribuições do *framework* proposto encerra este capítulo.

4.1 Fundamentação Utilizada para Construção do *Framework* Proposto

Soluções isoladas para as tarefas da atividade de desenvolvimento voltadas para *Big Data* existem, ver Seção 3.1, mas essas soluções podem provocar uma falta de sincronismo entre o que é modelado e o código fonte criado. Essa falta de sincronismo prejudica a documentação, não preserva os investimentos, gera retrabalho e diminui a produtividade em manutenções e evoluções. Desse modo, é necessário pensar em soluções que suportem de maneira automática ou semiautomática as tarefas da atividade de desenvolvimento (análise, projeto e implementação) para *Big Data*.

Algumas abordagens, baseadas em modelos como MDA, afirmam que um problema pode ser solucionado usando diferentes níveis de abstração [25]. Um nível de abstração mais alto permite que se tenha uma visão do problema sem considerar aspectos específicos que seriam necessários para solução final. Nessa abordagem, os aspectos serão considerados separadamente, permitindo que o desenvolvimento se torne mais flexível e permita maior integração de soluções. Dessa forma, ela consegue oferecer suporte a uma melhor produtividade no desenvolvimento, além de fornecer aspectos de qualidade, como manutenção e interoperabilidade [24]. No caso de *Big Data*, essa abordagem pode auxiliar no desafio de integrar soluções diferentes a um mesmo problema levantado por Madhavji et al. [60].

Para uma melhor compreensão sobre essa questão, imagine uma solução para análise de perfil de usuários que usam o *twitter*. Para esse exemplo, imagine que as entidades do problema foram abstraídas em um modelo e que o aspecto referente a *Big Data* foi abstraído em outro. A Figura 4.1 apresenta os dois modelos com suas visões independentes, mas necessárias para uma possível solução do problema. Como tornar essas visões independentes em uma única visão, ou seja, como é possível fazer a análise de perfil executar com aspectos *Big Data*?

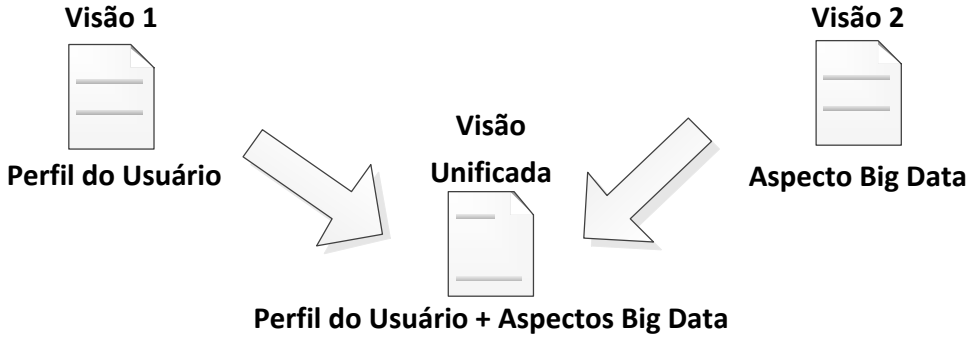


Figura 4.1: Exemplo de unificação de visões.

Na literatura, uma maneira de unificar modelos é através da operação de *weaving*, utilizada em trabalhos como os de Bézivin et al. [13–15], Pablo et al. [63], Belangour et al. [11] e Fabro et al. [33,35]. Esse operador, diferente de outros existentes[1], permite que haja a entrada de n modelos ao mesmo tempo e que se obtenha como resultado um modelo de saída unificado.

[1]Bernstein [12] em seu trabalho apresenta outros operadores que podem ser utilizados para manipulação de modelos.

Esses trabalhos [11, 13–15, 33, 35, 63] promovem a operação de *weaving* de modelos. Alguns desses trabalhos, como o de Pablo et al. [63], fazem *weaving* de modelos entre o PIM e o PDM. O PIM, geralmente, contém a lógica do negócio relacionada ao domínio de interesse, e o PDM apresenta características ou aspectos que se deseja agregar à lógica do negócio (i.e. PIM) para gerar o PSM. O operador de *weaving* nesse contexto é o mais apropriado, visto que vários aspectos diferentes, cada um representado por um PDM, poderiam ser agregados para um mesmo PIM, caracterizando uma entrada com *n* modelos. Assim, o modelo que abstrai as entidades do exemplo, mencionado nesta seção, seria o PIM e o modelo que abstrai o aspecto de *Big Data*, no mesmo exemplo, seria o PDM.

Um processo de desenvolvimento que descreve bem essa abordagem é o Processo de Desenvolvimento em Y (PDY). O PDY consiste em relacionar modelos complementares e localizados em extremos opostos da parte superior do Y, tendo como saída um modelo de *weaving* [11].

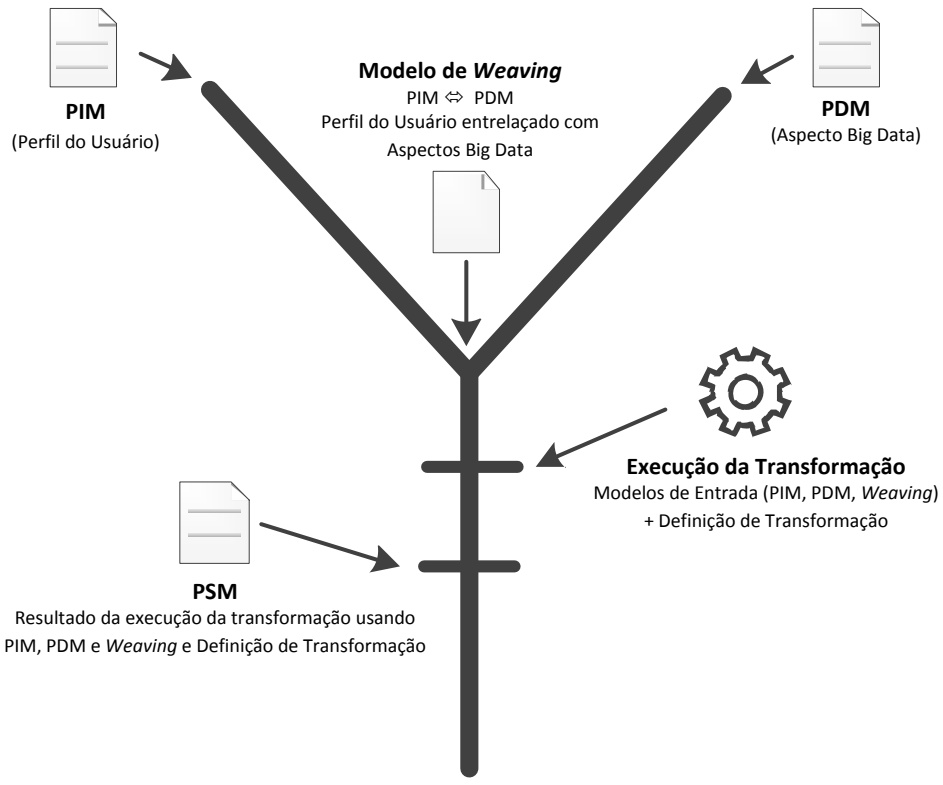


Figura 4.2: O processo de desenvolvimento de *software* em Y - baseado em Bézivin et al. [15].

Na Figura 4.2, os modelos de PIM e PDM estão posicionados um em cada ponta do Y e são relacionados através da operação de *weaving*, que resulta em um modelo de *weaving*. Em seguida, uma transformação é executada tendo como entrada

a definição de transformação, o modelo de *weaving*, o PIM e o PDM para gerar um PSM.

4.2 Arquitetura do *Framework* Proposto (F2BD)

O *Framework para Desenvolvimento de Sistemas de Software para MapReduce de Big Data (F2BD)* é apresentado na Figura 4.3. Este *framework* é baseado em MDE, *weaving* de modelos e PDY. A ideia básica do *framework* proposto é suportar o entrelaçamento entre os modelos que representam a lógica do negócio (PIM) e os modelos que representam o PDM (i.e. *Big Data*, mas específico o modelo *MapReduce*) para, através de definições de transformação, gerar o PSM, em seguida, gerar o código fonte e demais artefatos de *software*.

Dessa forma, esse *framework* se diferencia dos demais, pelo domínio adotado, metamodelos e modelos propostos, além da proposta da extensão do metamodelo de UML. A Figura 4.3 apresenta a visão geral da arquitetura do *framework* proposto.

A utilização desse *framework* permitirá a implementação de ferramentas que suportem a atividade de desenvolvimento de *software* para o modelo *MapReduce* de *Big Data*.

O *framework* proposto possui as seguintes características:

1. **Automação de tarefas:** A atividade de desenvolvimento se tornará semi-automática, visto que os modelos de entrada (PIM e modelo de *weaving*) deverão ser criados manualmente. Posteriormente, o processo se tornará automático, através da aplicação de definições de transformações;

2. **Preservação dos investimentos:** Este *framework* suporta a criação de um PIM, conforme uma extensão do metamodelo de UML, que inclui a linguagem VisualAlf, que permite a descrição dos comportamentos dos métodos;

3. **Facilitar tarefas de Manutenção, Evolução e Documentação:** A lógica do negócio é representada através de modelos (PIM) e, depois, combinada com a descrição de plataformas (PDM) por meio do modelo de *weaving*. O PIM, o PDM e o modelo de *weaving* são utilizados para gerar o PSM. Por sua vez, este

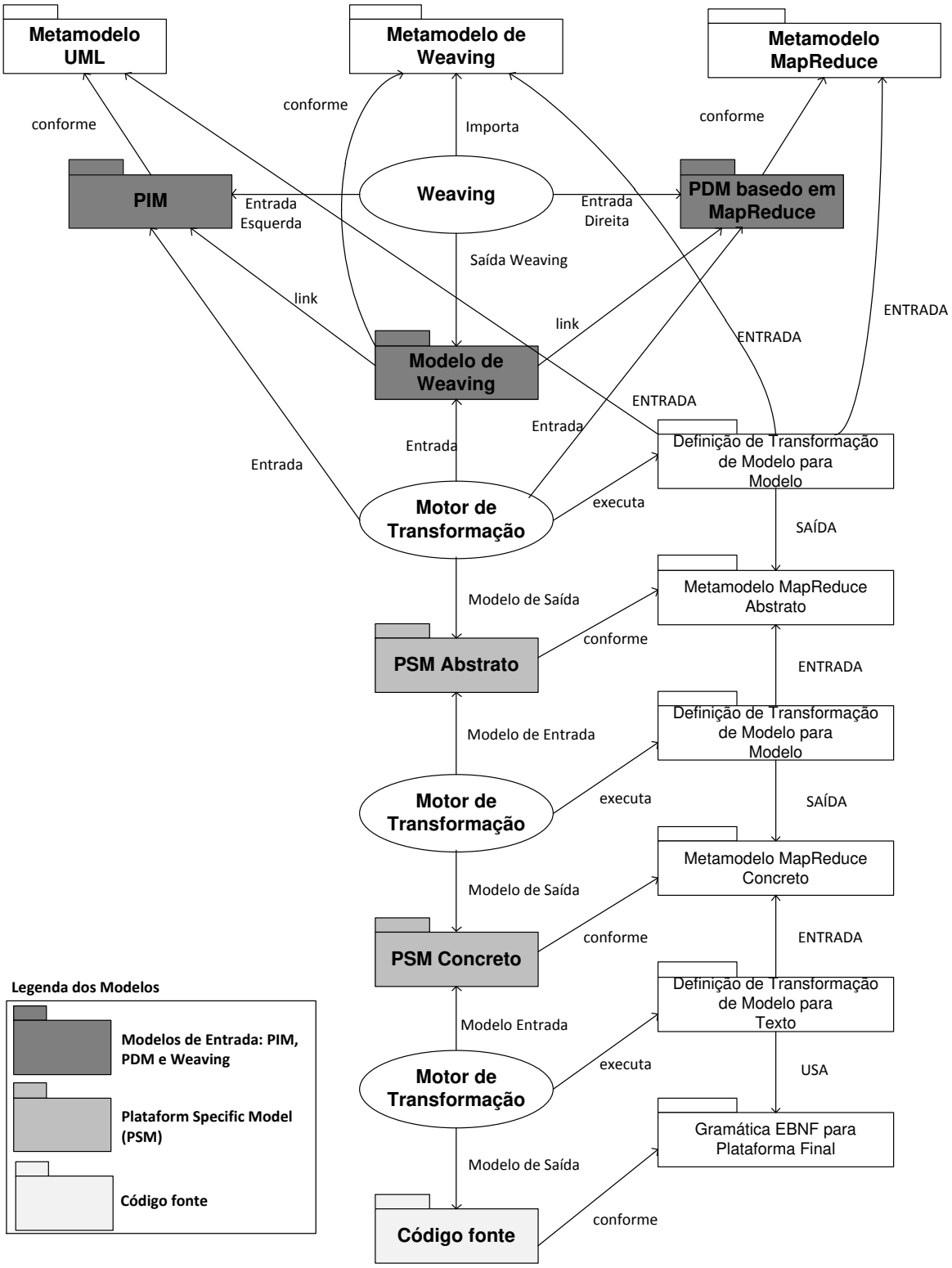


Figura 4.3: Arquitetura do *Framework* para Desenvolvimento de *Software* para *MapReduce* de *Big Data* (F2BD).

último é utilizado para gerar o código fonte do sistema de *software*. Quando houver necessidade de fazer manutenção no sistema, esta é feita primeiramente no modelo PIM e, em seguida, as definições de transformação geram os modelos

contendo as modificações decorrentes da manutenção. Quando há necessidade de fazer evoluir o sistema para uma nova versão da plataforma ou utilização de outra plataforma, o PIM em conjunto ou não com o PDM ou um novo PDM são utilizados para gerar um novo PSM, baseado na nova plataforma. Os modelos PIM, PDM, *weaving* e PSM e suas descrições textuais servem como documentação do sistema, pois são representações em diferentes níveis de abstração do sistema de *software* real.

A arquitetura do *framework* será apresentada em três seções: a primeira, na Seção 4.2.1, tratará dos modelos de entrada; a segunda, na Seção 4.2.2, tratará os modelos específicos de plataforma; e, finalmente, a terceira, na Seção 4.2.3, abordará sobre como o código fonte do sistema de *software* é gerado.

4.2.1 Modelos de Entrada

Os modelos de entrada utilizados pelo *framework* são: o PIM, o PDM e o modelo de *weaving*. Esses modelos são importantes para o sucesso da construção do *software*, utilizando o *framework* proposto, pois será com base neles, que as demais tarefas do *framework* (geração dos modelos específicos de plataforma e posterior geração do código fonte) serão realizadas. Dessa forma, a construção desses modelos deve ser feita de maneira que eles reflitam a solução do problema.

O PIM utilizado está conforme o metamodelo de UML que foi estendido para permitir que os métodos modelados pudessem ter seu comportamento incluído de forma bem definida através da proposta de VisualAlf. Uma discussão mais detalhada sobre a extensão do metamodelo de UML será apresentada nas Seções 4.3.1 e 4.4. A linguagem UML foi adotada para descrever o nosso PIM, pois ela é uma linguagem independente de plataforma, além de ser a mais utilizada para modelagem de *software*. Esses fatos podem influenciar na utilização e na aceitação do *framework* proposto. A proposta de VisualAlf contém um metamodelo baseado em Alf [72] e uma notação gráfica para descrever o corpo de métodos. A descrição do comportamento dos métodos das classes de UML, através de VisualAlf, assegura que as características de UML ser gráfica e intuitiva permaneçam.

O PDM é criado conforme o metamodelo de *MapReduce* proposto na Seção 4.3.2. Esse modelo é que descreve as características de *MapReduce* que deverão ser entrelaçadas com os elementos do PIM, de tal forma que a solução computacional comece a ter características de uma solução *Big Data*. O PDM é criado apenas uma vez, utilizado quantas vezes for necessário e pode ser estendido para acompanhar a evolução do modelo *MapReduce*.

O modelo de *weaving* é construído conforme o metamodelo de *weaving* proposto na Seção 4.3.3. O modelo de *weaving* é responsável por descrever os relacionamentos entre os elementos do PIM e do PDM. A partir do modelo de *weaving*, aspectos do modelo *MapReduce*, representados pelo PDM, serão associados ao PIM.

A Figura 4.4 apresenta um fragmento da arquitetura do *framework* proposto que ressalta os modelos de entrada e seus metamodelos.

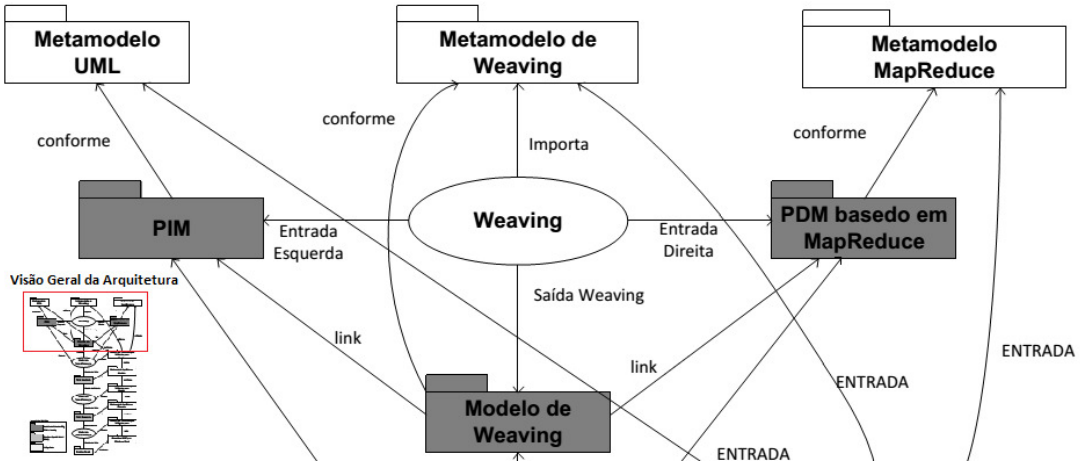


Figura 4.4: Modelos de Entrada do *Framework* Proposto.

4.2.2 Modelos Específicos de Plataforma

O *framework* lida com a complexidade do *software* trabalhando em diferentes níveis de abstração, ou seja, alguns modelos serão dependentes de plataforma e outros serão independentes de plataforma. À medida que as tarefas vão sendo realizadas, o nível de abstração vai diminuindo e, em contrapartida, a especificidade vai aumentando. Para minimizar a mudança de um nível de abstração alto para um mais baixo e para flexibilizar o desenvolvimento, o *framework* oferece dois níveis de PSM: um abstrato e outro concreto[2].

[2]Um dos primeiros documentos a trazer a ideia de modelo abstrato e concreto foi apresentado pela OMG [74]. Eram denominados de *Implementation Language Environment Independent* e *Implementation*

O PSM abstrato tem detalhes sobre uma plataforma específica, mas sem detalhes de implementação, enquanto o PSM concreto tem detalhes de uma plataforma específica para uma implementação. Por exemplo, pense em um serviço *Web*. O PSM abstrato traria os detalhes da plataforma de serviço *Web*, mas sem trazer nenhum detalhe de implementação, enquanto o PSM concreto traria os detalhes da mesma plataforma implementados para uma linguagem. Um exemplo de PSM concreto poderia ser a API da *Apache Axis* para serviços *Web* usando *Java*.

No *framework* proposto, em vez de gerar diretamente um PSM concreto a partir de um PIM, propõe-se gerar um PSM abstrato a partir de um PIM, um PDM e um modelo de *weaving*. Em seguida, a partir deste PSM abstrato se gera um PSM concreto. Por exemplo, a partir do PSM abstrato definido para o modelo *MapReduce* do *framework Spark*, pode-se gerar o PSM concreto para qualquer uma das linguagens que implementam *Spark* que no caso são *Java*, *Python* e *Scala*.

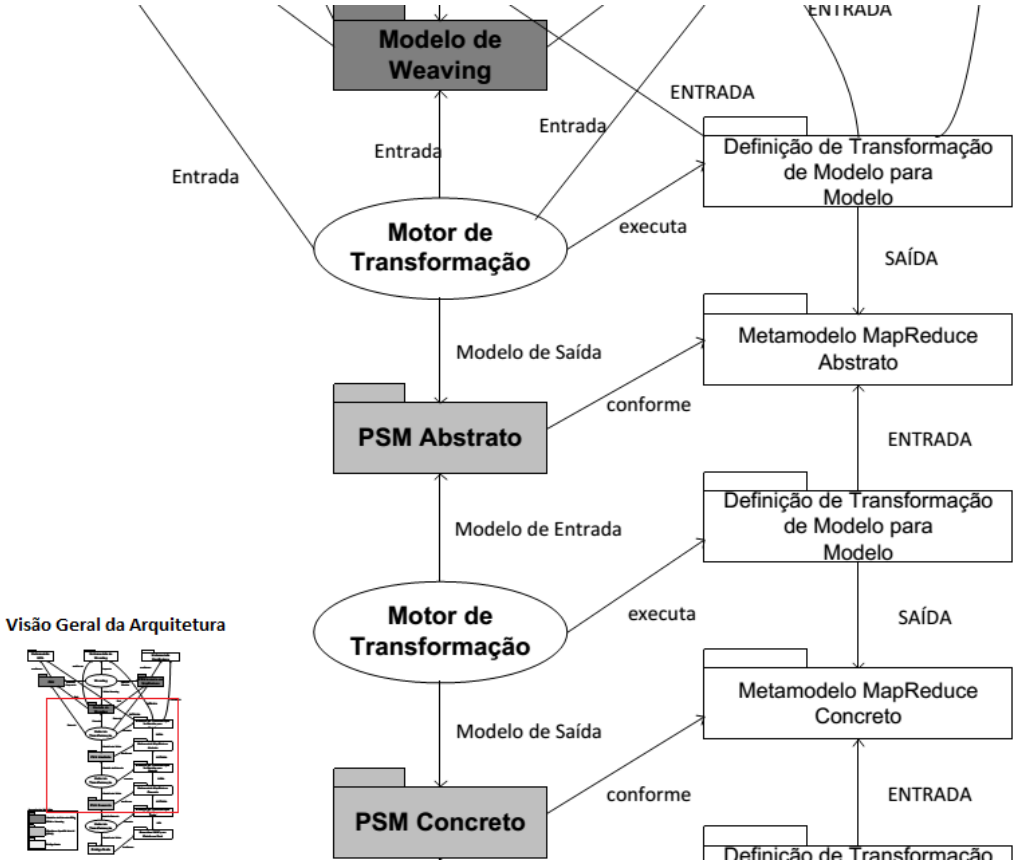


Figura 4.5: Modelos Específicos de Plataforma do *Framework* Proposto.

A Figura 4.5 mostra o processo de geração do PSM abstrato, que deve estar conforme um metamodelo que o descreva. A geração do PSM abstrato a partir de

Language Environment Specific, respectivamente. Um segundo trabalho, trazendo esses conceitos, foi apresentado por Almeida et al. [2].

um PIM e PDM é feita de maneira automática, através de um motor de transformação que recebe como entrada: os modelos PIM, PDM e modelo de *weaving* e definições de transformações de modelo para modelo. Essas definições de transformação serão responsáveis por determinar como o PIM, PDM e modelo de *weaving* dados como entrada vão se transformar no modelo de saída, que, no caso, é o PSM abstrato.

Ainda na Figura 4.5, é possível ver o processo de geração do PSM concreto, que deve estar conforme um metamodelo que o descreva. A geração do PSM concreto ocorre através de uma nova chamada para o motor de transformação, que terá como entrada: o PSM abstrato, um modelo de uma API que implemente os conceitos de *MapReduce* e um novo conjunto de definições de transformações de modelo para modelo. O modelo da API que implemente *MapReduce* deve estar conforme ao metamodelo do PSM concreto. É importante ressaltar que o propósito do *framework* é suportar o desenvolvimento de soluções para *Big Data*, mas utilizando soluções existentes. Assim, o modelo de uma API como entrada para geração do PSM concreto garante que seja possível a utilização de soluções existentes para chegar no resultado final desejado.

4.2.3 Código Fonte

Nesta última etapa, a tarefa consiste em gerar o código fonte e demais artefatos de *software* que serão utilizados na compilação e implantação do sistema de *software*, baseado em *Big Data*.

A Figura 4.6 mostra que uma definição de transformação de modelo para texto descreve como os elementos do PSM concreto devem ser manipulados computacionalmente pelo motor de transformação para gerar o código fonte conforme a gramática EBNF da plataforma final. É importante destacar que, se o código fonte gerado não está correto, as alterações devem ser realizadas nos modelos de entrada e as transformações executadas novamente até que o código fonte esteja de acordo com o especificado pelos usuários.

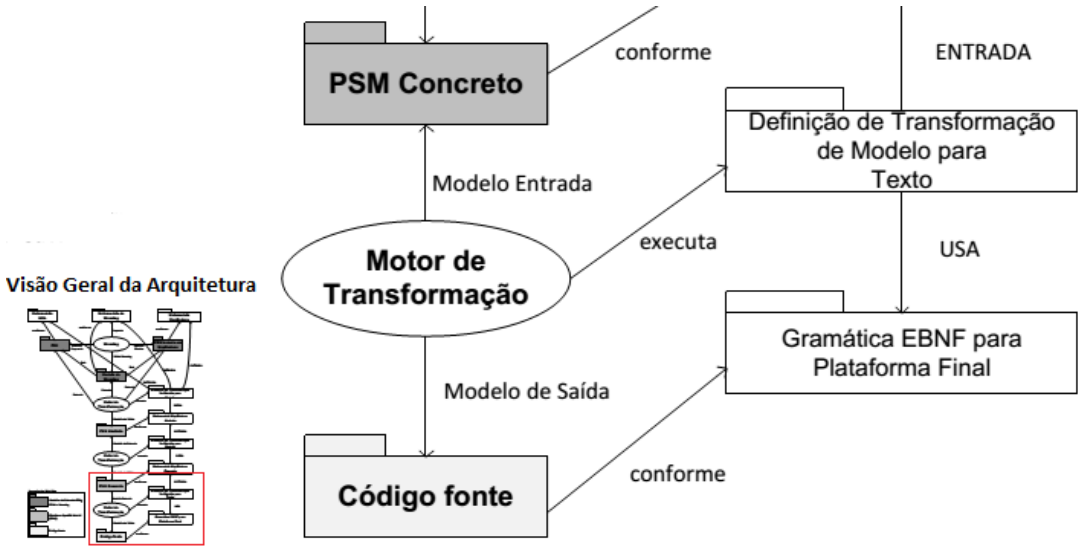


Figura 4.6: Geração do Código Fonte no *Framework* Proposto.

4.3 Metamodelos

Os metamodelos são essenciais para descrever como modelos são construídos. O *framework* proposto tem como seu principal artefato modelos com diferentes níveis de abstração, assim cada um desses modelos está escrito conforme um metamodelo.

Nessa seção, os metamodelos propostos, que descrevem os modelos utilizados pelo *framework* proposto, serão apresentados. Primeiramente, a proposta de extensão do metamodelo de UML será apresentada na Seção 4.3.1, contemplando a possibilidade de inclusão da lógica de negócio, através da definição do comportamento dos métodos das classes de UML. Em seguida, o metamodelo para descrever o PDM, baseado em *MapReduce*, e o metamodelo para descrever modelo de *weaving* serão apresentados, respectivamente na Seção 4.3.2 e na Seção 4.3.3.

O *framework* proposto apresenta um conjunto de tarefas que devem ser elaboradas para possibilitar a construção de ferramentas que suportem o desenvolvimento de *software* para *MapReduce* de *Big Data*, assim, os metamodelos específicos de plataforma serão apresentados no Capítulo 5 que tratará sobre a implementação de uma ferramenta, utilizando o *framework* proposto.

4.3.1 Metamodelo para Visual Alf

O PIM utilizado pelo *framework* proposto é baseado no metamodelo de UML, mas a linguagem UML não apresenta em seu metamodelo uma forma padrão para incluir comportamento nos métodos das classes. Esse comportamento é inserido através de campos de textos livres, como o campo *body* da UML, onde pode ser colocado trechos de códigos de qualquer linguagem. Assim, o *framework* propõe a extensão do metamodelo de UML para resolver o problema de falta de padronização na ação de definir comportamento para os métodos das classes, pois não foi encontrado na literatura um metamodelo que ofereça uma padronização que se adeque à realidade deste trabalho. Uma discussão mais detalhada sobre essa problemática será apresentada na Seção 4.4.

O comportamento (lógica de negócio) será incorporado no modelo de maneira padrão, a partir das novas classes definidas no metamodelo estendido. A extensão do metamodelo de UML ocorrerá a partir da classe *UMethod* que se relacionará com algumas classes do metamodelo proposto, por exemplo, a classe *Block*, *Invocation* e *Instantiation*. Este ponto de extensão foi escolhido, pois o metamodelo proposto pretende descrever comportamento para métodos definidos em modelos UML. A maioria dos elementos, que descreve comportamento no metamodelo proposto, vai herdar a super classe *Instruction*, que, por sua vez, vai estar contida em uma classe *Block*. Assim, a classe *Block* serve como um contêiner de instruções que juntas irão descrever o comportamento para uma instância da classe *UMethod*. Dessa forma, o objetivo desta classe é substituir o campo *body* da classe *UMethod*, que aceita qualquer texto, por um conjunto de instruções bem definidas.

Primeiramente, as classes que herdam da super classe abstrata *Instruction* serão apresentadas na Figura 4.7, e em seguida as classes que herdam da super classe abstrata *Result* serão apresentadas na Figura 4.8. As principais classes, que herdam da super classe abstrata *Instruction*, são descritas a seguir:

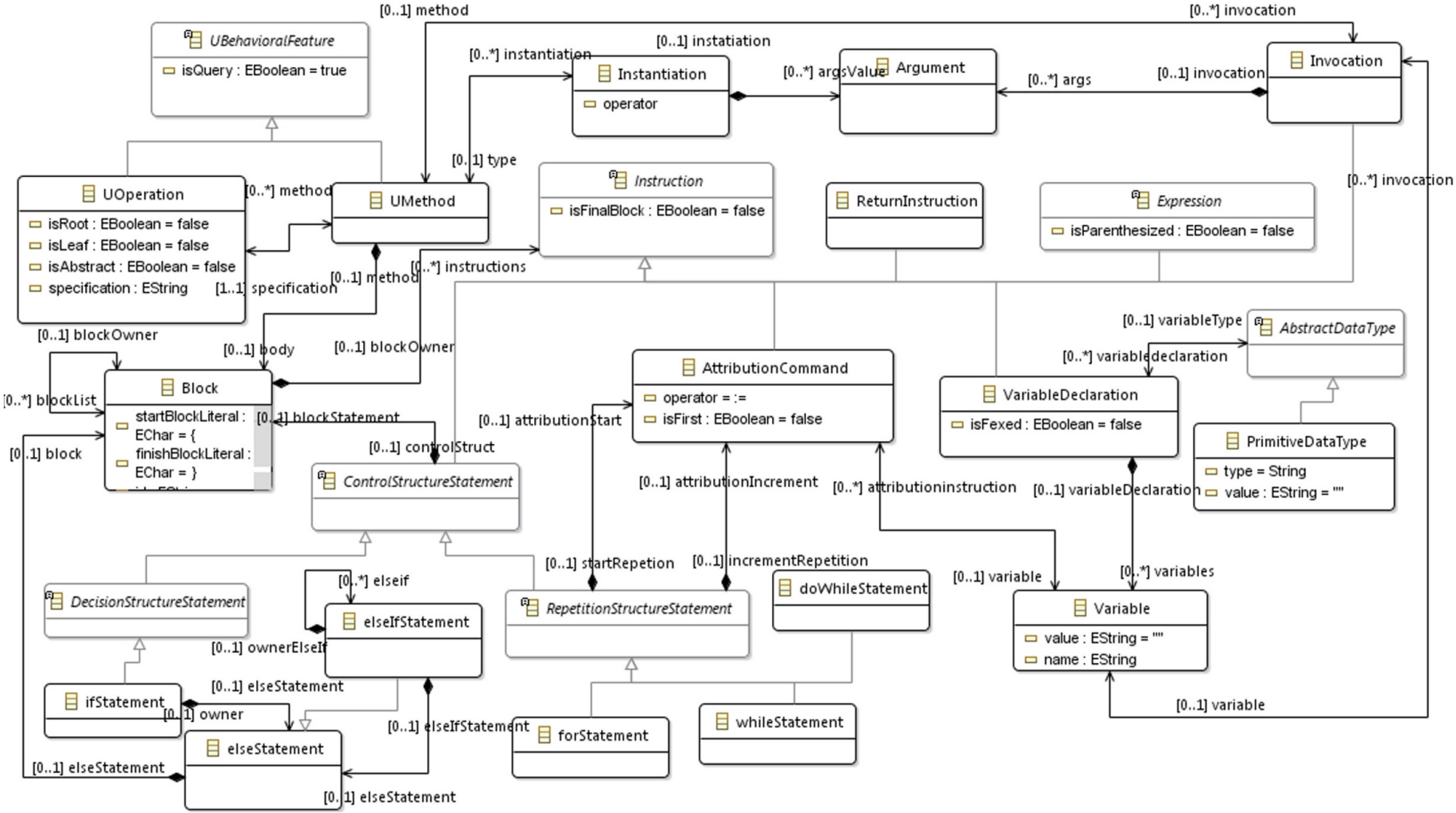


Figura 4.7: Fragmento do Metamodelo da Linguagem Visual: Instruções

- **ControlStructureStatement:** Super classe abstrata que representa as instruções de controle, tais como instruções condicionais (*IfStatement*) e instruções de repetição (*ForStatement*, *WhileStatement*, etc);

- **VariableDeclaration:** Classe que define um tipo de dado para uma ou mais instâncias da classe *Variable*. A classe *Variable* determina um identificador (nome) para cada tipo de dado;

- **AttributionCommand:** Classe que atribui um valor para uma instância da classe *Variable*. O valor a ser atribuído pode ser qualquer instância da classe *Result* que será detalhada na Figura 4.8;

- **ReturnInstruction:** Retorna um valor que pode ser qualquer instância da classe *Result*;

- **Expression:** Super classe abstrata, que representa as expressões utilizadas, e detalhada na Figura 4.8;

- **Invocation:** Classe que invoca instâncias da classe *UMethod*, a partir de outras instâncias da classe *Variable*, passando argumentos ou não através da classe *Argument*.

Na Figura 4.8, é possível visualizar as classes que herdam da super classe abstrata *Result*. A classe *Result* é outra importante classe dentro da extensão do metamodelo de UML. Esta classe agrupa instruções que podem ser atribuídas ou retornadas a uma instância da classe *Variable*, por exemplo. Assim, pode-se ver que a classe *Result* possui um relacionamento direto com classes como: *AttributionCommand*, *Argument* e *ReturnInstruction*. As principais classes que herdam essa super classe são listadas a seguir:

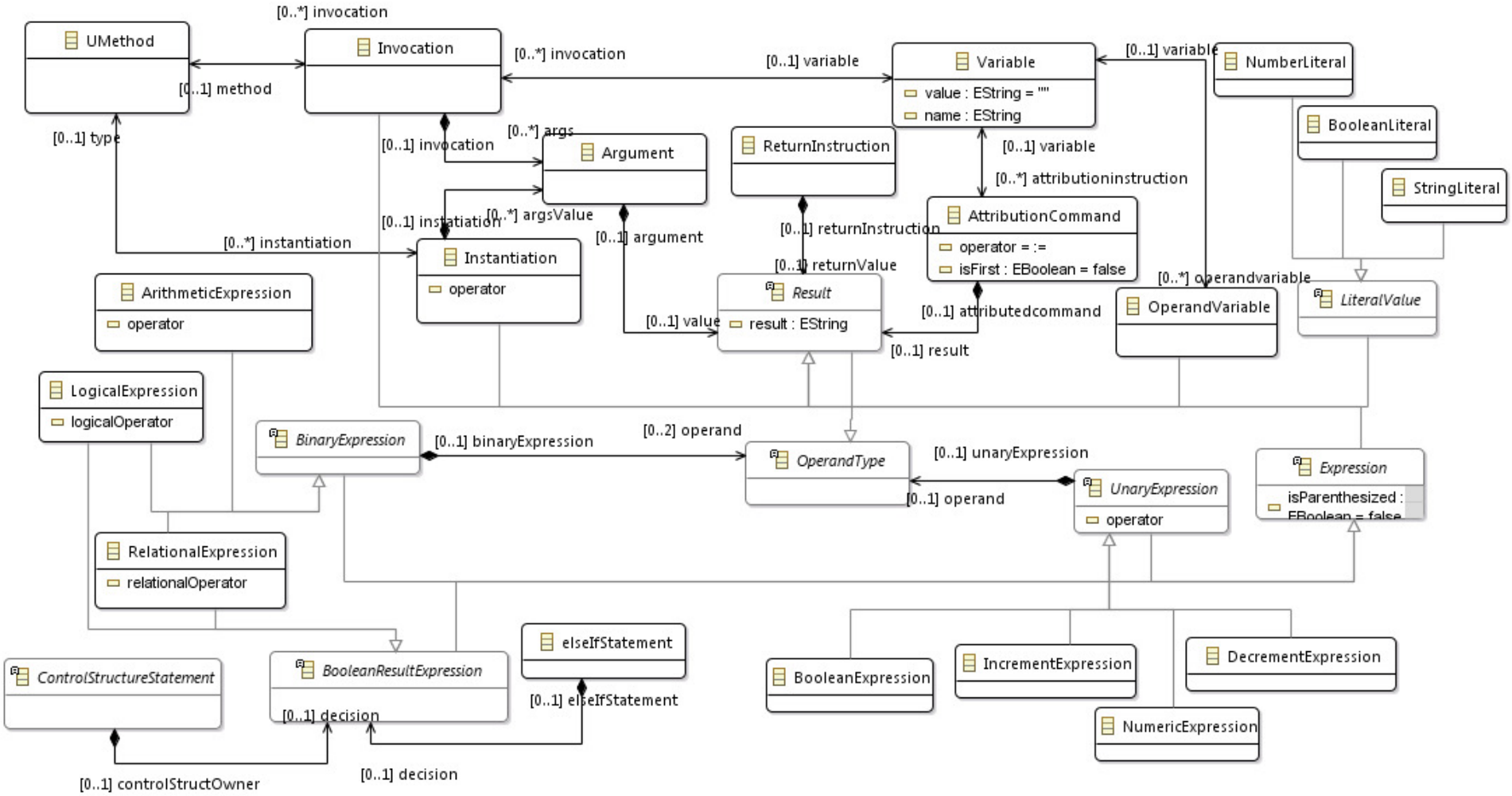


Figura 4.8: Fragmento do Metamodelo da Linguagem Visual: Expressões.

- **Expression:** super classe abstrata, que representa as expressões que são divididas pelas classes abstratas: *BinaryExpression* e *UnaryExpression*. A classe *BinaryExpression* representa expressões com dois operandos (*OperandType*), tais como: aritméticas (*ArithmeticExpression*), lógicas (*LogicalExpression*) e relacionais (*RelationalExpression*). Enquanto a classe *UnaryExpression* representa expressões com apenas um operando (*OperandType*), tais como: incremento (*IncrementExpression*), decremento (*DecrementExpression*), etc;

- **BooleanResultExpression:** Super classe abstrata, que agrupa comportamento lógico, utilizada por classes que herdam *ControlStructureStatement*. Suas classes concretas são: *RelationalExpression* e *LogicalExpression*;

- **LiteralValue:** Super classe abstrata, que atribui valores literais aos elementos, representada pelas classes: *StringLiteral*, *BooleanLiteral*, *NumberLiteral*;

- **OperandVariable:** Classe que retorna uma instância da classe *Variable*;

- **Instantiation:** Classe que cria um objeto, através da chamada de seu construtor representado por uma instância da classe *UMethod*.

4.3.2 Metamodelo do PDM

O metamodelo proposto para o PDM é fundamental na arquitetura do *framework*, pois é ele que descreverá o que poderá ser criado no PDM. Como a função de um PDM é descrever as características de uma plataforma. Assim o objetivo do PDM no *framework* proposto é descrever as características do modelo *MapReduce* para a plataforma *Big Data*, conforme o metamodelo que está sendo apresentado nessa seção.

O conceito de *MapReduce* é fundamentalmente comportamental, ou seja, as operações se resumem em ações comportamentais de mapear (dividir) e depois de reduzir (sintetizar). Assim, o metamodelo proposto para o PDM se concentra em descrever esses comportamentos.

O metamodelo proposto para criar o PDM é apresentado na Figura 4.9. Esse metamodelo contém classes como: pacotes (*Package*), classes (*Class*), métodos (*Method*), parâmetros (*Parameter*), etc. Estas classes são necessárias para modelar o comportamento de *Map* e *Reduce*.

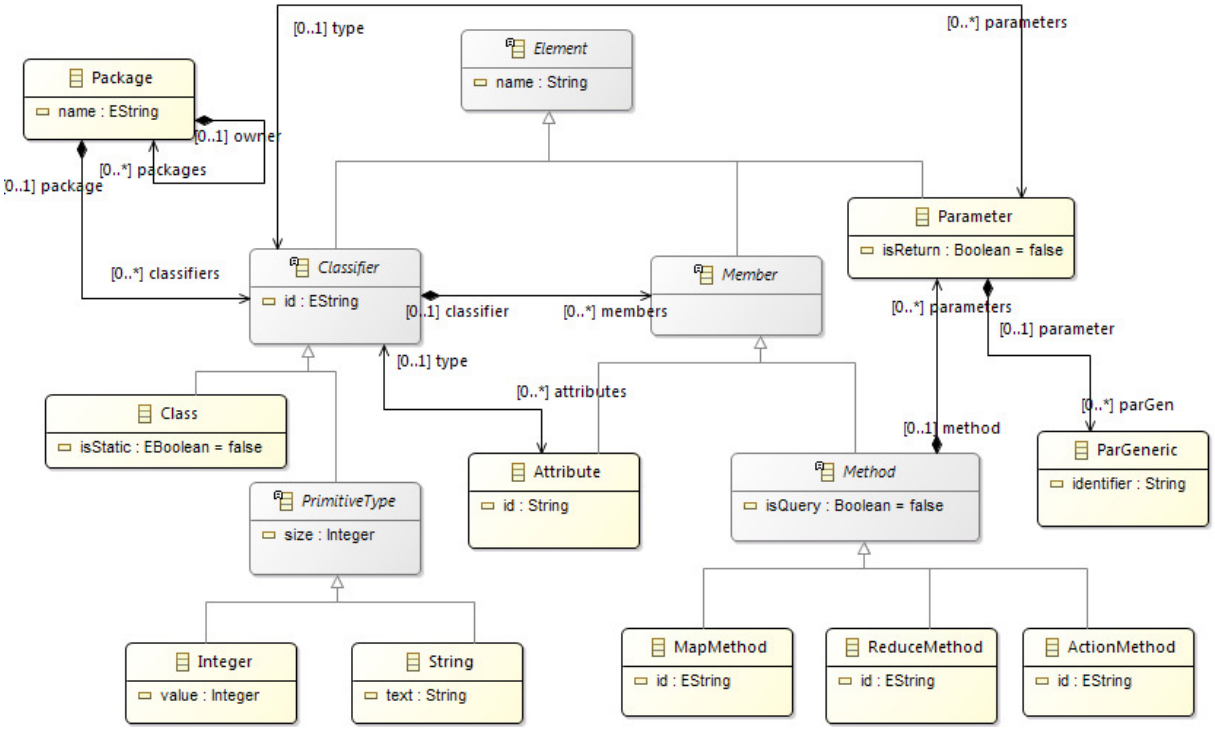


Figura 4.9: Metamodelo do PDM para *MapReduce* para *Big Data*.

O metamodelo para o PDM proposto permite criar um modelo que atende as especificações das implementações voltadas para plataforma *Hapoop* e para o *framework Spark*. A classe *Classifier* permite criar classes que representem os *Mapper* e *Reducer*, que são usados pela plataforma *Hadoop*, enquanto os comportamentos definidos nessas classes podem ser utilizados para indicar o comportamento utilizado no *framework Spark*. As classes que herdam a super classe abstrata *Method* indicam três tipos de comportamento: *MapMethod*, *ReduceMethod* e *ActionMethod*. A primeira descreve comportamento de *Map*, a segunda descreve comportamento de *Reduce* e a terceira descreve comportamento tradicional.

A palavra comportamento está sendo utilizada no contexto desse trabalho com o objetivo de descrever os tipos de ações que um método de uma classe realizará. Essas ações estão sendo classificadas em:

1. **Map**: quando um método tem o objetivo de dividir algo;
2. **Reduce**: quando um método tem o objetivo de sintetizar algo;
3. **Tradicional**: quanto um método tem o objetivo que não seja de agrupar ou sintetizar algo.

Assim, por exemplo, quando é utilizado o termo comportamento de *Map*, está se fazendo referência a um método que divide algo, da mesma forma que quando o termo comportamento tradicional é utilizado, está se fazendo referência a um método que não agrupa e nem sintetiza algo.

A diferenciação destes comportamentos é essencial para que o modelo de *weaving* seja criado e as transformações de modelos para modelos e de modelos para texto possam ser aplicadas.

4.3.3 Metamodelo de *Weaving*

O metamodelo de *weaving*, proposto nesta seção, é de fundamental importância para o *framework* proposto, pois nele é possível encontrar a descrição do que pode ser criado no modelo de *weaving*.

A Figura 4.10 apresenta um fragmento do metamodelo de *Weaving* proposto. As classes *WLeftModel* e *WRightModel* descrevem, respectivamente, os modelos da esquerda e da direita que serão entrelaçados. As classes *WLeftItem* e *WRightItem* representam, respectivamente, referências aos elementos dos modelos da esquerda e da direita.

Uma instância da classe *WModel* representa o modelo de *weaving*, que será criado através do agrupamento de uma ou várias instâncias da classe *WItem*. A classe *WItem* é a responsável por estabelecer a associação entre os elementos do modelo da esquerda (*WLeftItem*) e os da direita (*WRightItem*). O modelo de *weaving* criado, conforme o metamodelo descrito, deverá conter somente a correspondência dos elementos do PIM, que necessitem assumir um dos comportamentos definidos no PDM.

O *framework* proposto foi concebido para preservar os investimentos feitos em *software*, inclusive permitindo que a lógica do negócio possa ser modelada no PIM. Assim, o metamodelo de *weaving* proposto também deve prever a possibilidade da preservação da lógica de negócio, além das classes mínimas necessárias para descrever a operação de *weaving*. Esta necessidade da inclusão da lógica de negócio no modelo de *weaving* decorre do fato que o comportamento *Big Data*, necessário exclusivamente

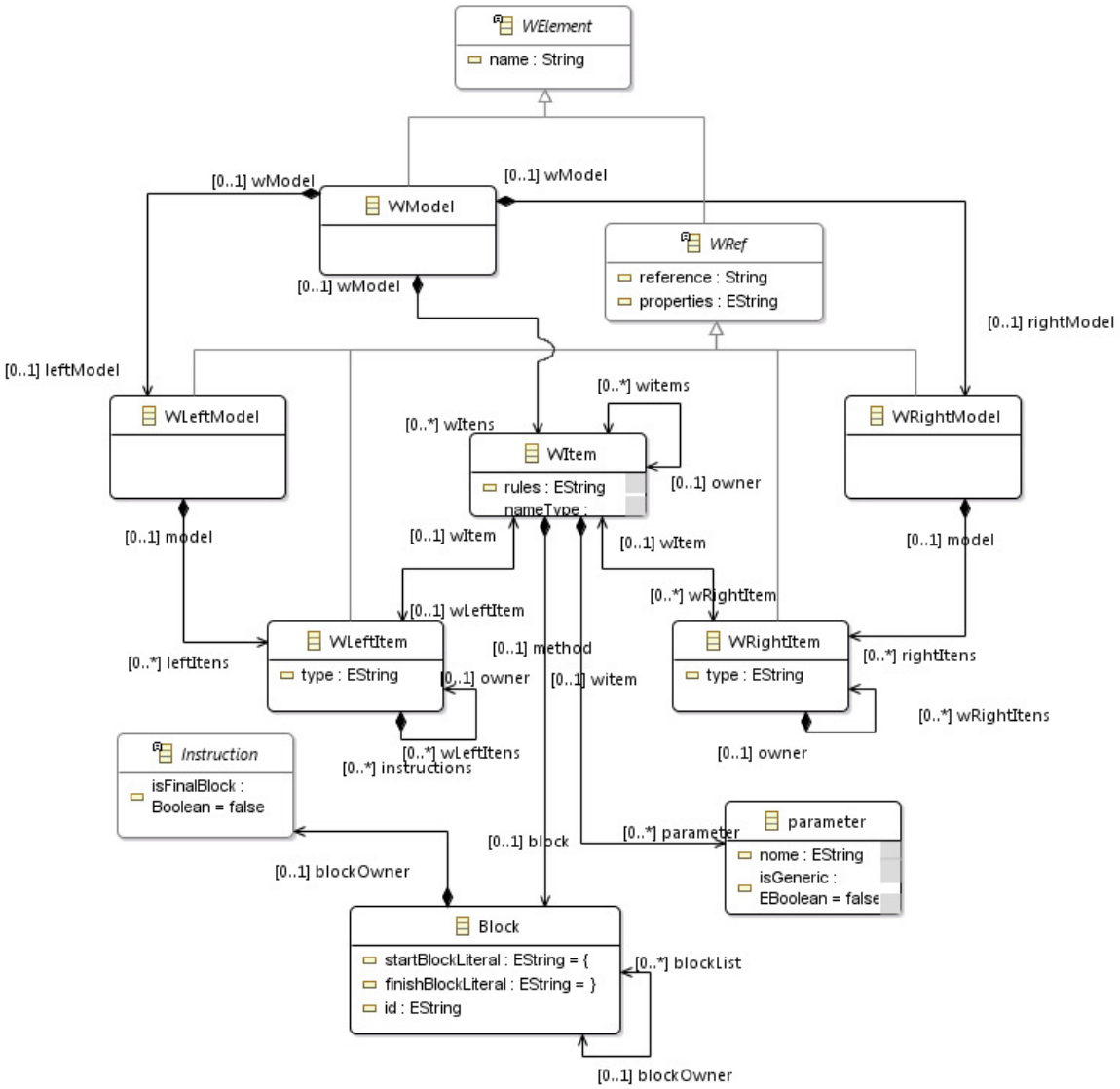


Figura 4.10: Fragmento do Metamodelo de *Weaving* - baseado em Fabro et al. [35].

para o problema que será construído, precisa ser definido em um local que não tirasse a independência do PIM e nem a generalidade do PDM.

O metamodelo de *weaving* permite a inclusão da lógica de negócio através da classe *WItem*, que tem um relacionamento direto com a classe *Block*. Assim como no metamodelo do PIM, a classe *Block* tem a função de conter várias instâncias da classe *Instruction*, dessa forma, permitindo a descrição da lógica de negócio. A lógica de negócio incluída em uma instância da classe *WItem*, através da classe *Block* determinará o comportamento *MapReduce* necessário ao entrelaçamento.

4.4 VisualAlf: Notação Gráfica

A UML é uma linguagem visual criada para especificar artefatos de *software*. A UML, inicialmente, não previu a possibilidade de construir modelos executáveis. Uma das primeiras iniciativas para adicionar um suporte para construir modelos executáveis em UML foi a partir da sua versão 1.5, quando foi adicionado a sua especificação o conceito de ações semânticas [75]. Posteriormente, a OMG trouxe os conceitos de *Foundational UML (fUML)* [76] e *Action Language for Foundational UML (Alf)* [72]; a primeira formaliza um subconjunto de UML para construir modelos executáveis, e a segunda corresponde à sintaxe concreta para fUML.

A ideia de especificar comportamentos executáveis dentro de um modelo representado por uma linguagem visual não vem sendo utilizada pela indústria e tem sido subvalorizada e muitas das vezes negligenciada, pela academia. Um exemplo de sucesso do uso de linguagem visual, aplicada nas engenharias, é a ferramenta conhecida como LabVIEW [23].

Analisando o metamodelo da UML, percebe-se que cada diagrama desta linguagem de modelagem possui elementos em comum e interligados, dependendo do contexto. Por exemplo, a noção de objeto do diagrama de sequência é comum à noção de objeto do diagrama de colaboração, assim como a noção de objeto também é comum ao diagrama de atividade. Assim, nota-se que a característica de unificação da UML não é apenas pela junção de várias linguagens, por exemplo, as de Jacobson, Booch e Rumbaugh, mas também por que ela unifica diversos pontos de vista.

Os diagramas de UML oferecem uma visão macro do sistema de *software* e sempre se tentou dar uma visão micro (comportamental ou executável), através de campos de texto livre, como: o campo *body* da UML. No entanto, não há um padrão em um campo de texto livre, porque poderia ter um *body* com instruções C++ e em outro com instruções Java dentro de uma mesma classe.

A utilização de Alf possibilitou construir modelos com comportamento, utilizando uma linguagem padrão, mas não resolveu a questão do campo de texto livre. A própria especificação de Alf orienta que o comportamento escrito textualmente em Alf precisa ser adicionado no campo *body* [72]. Nesse contexto, a UML volta ao mesmo problema inicial, pois a linguagem textual Alf existe independente do

metamodelo de UML, assim como as demais linguagens, apesar de ter sido criada para complementá-la. Dessa forma, a interligação de Alf com UML em relação aos metamodelos é necessária para que se possa atribuir a visão micro da lógica de negócio de forma padrão e visual nos modelos de UML.

Neste cenário, é proposta uma extensão no metamodelo de UML para permitir que instruções, baseadas em Alf, possam ser integradas nos elementos de um modelo UML, ao contrário da OMG que inseriu a linguagem Alf de forma textual no campo *body* dos métodos das classes de UML. Assim, é fornecido um metamodelo (veja Seção 4.3.1) e uma representação gráfica para Alf. Essa representação gráfica tem a intenção de preservar a filosofia da UML de ser visual e interativa.

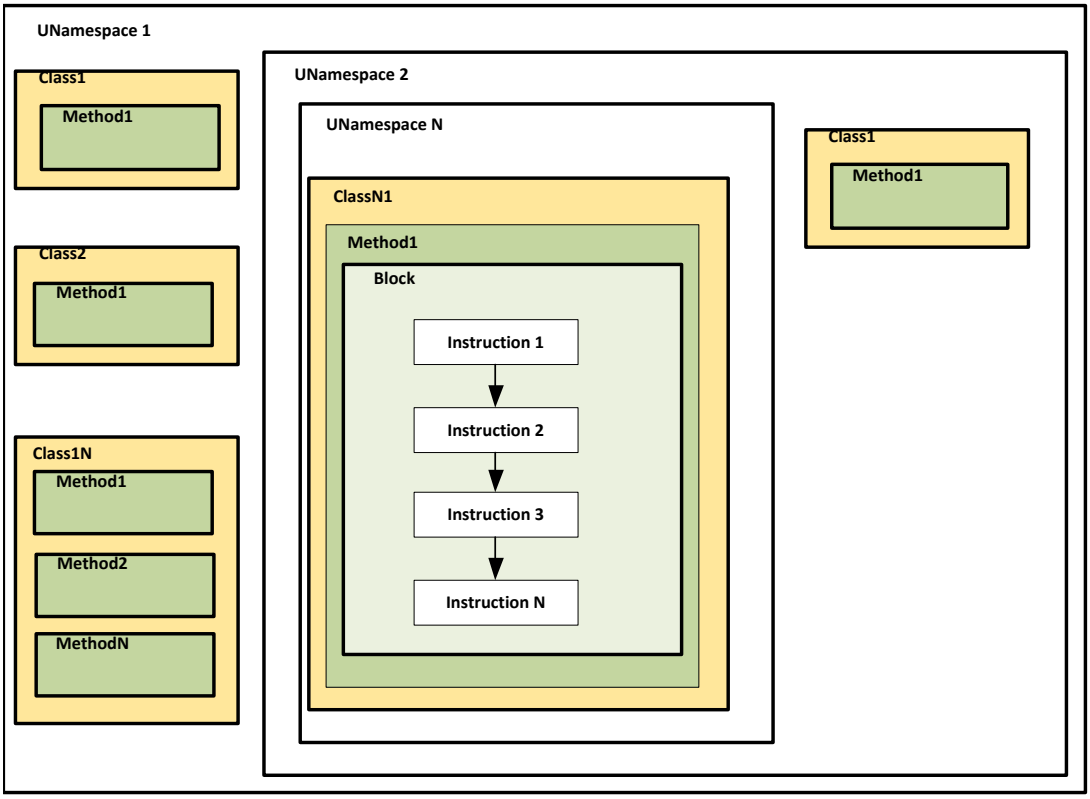


Figura 4.11: Proposta de uma notação gráfica para VisualAlf.

A Figura 4.11 apresenta uma proposta de notação gráfica para Alf. A notação gráfica permite que os métodos sejam agrupados em suas classes e que as classes sejam agrupadas em *namespaces*. Entretanto, o principal objetivo da notação gráfica de VisualAlf é que o comportamento de um método seja modelado através da inclusão de uma sequência de instruções, conforme o metamodelo de VisualAlf. No Apêndice B, é apresentado um maior detalhamento sobre a notação gráfica e um exemplo ilustrativo da sua implementação.

É importante refletir sobre o proposito de uma representação gráfica. A representação gráfica tem por objetivo descrever um elemento ou uma situação de forma visual. Essa representação, se bem feita, poderá facilitar a compreensão e o manuseio dos elementos que compõe um modelo. Esse aspecto é muito positivo, mas, no entanto, a inserção da representação gráfica, geralmente, ocasiona uma certa lentidão no processo envolvido, pois a interatividade, na grande maioria dos casos, se concentra no uso de dispositivos de entrada, como o *mouse* e na limitação de tamanho dos dispositivos de saída, como monitores. Dessa forma, a representação gráfica pode ser utilizada em qualquer situação, mas vai depender muito do proposito de sua utilização, assim como da interatividade oferecida. Neste trabalho, serão oferecidas duas formas de interação uma em forma de árvore, mais próxima da forma textual, e a outra de forma gráfica, por meio da representação visual.

4.5 Coesão na Criação do PIM

As regras de transformação de modelos são reutilizáveis, isto implica que, uma vez criadas e inseridas em ferramentas, toda a complexidade de passar ou acrescentar ou unificar informações de um modelo para outro fica mascarada para o usuário de uma ferramenta baseada em MDE. Entretanto, a situação se complica quando regras de transformação de modelos têm como entrada modelos, por exemplo PIM, que são criados sem padronização. Mesmo regras de transformação de modelos bem concebidas e escritas podem gerar modelos inadequados para uso, em decorrência de um modelo de entrada mal concebido e estruturado.

De fato, a utilização de modelos não garante que o produto final seja de qualidade, mas a utilização correta de uma linguagem de modelagem e o uso de boas práticas aumentam as chances de sucesso do produto final. Sendo assim, é imprescindível que a concepção e a criação de modelos em seus diferentes níveis, i.e. PIM, PDM, PSM e código fonte sejam realizadas, utilizando princípios de engenharia de *software* como a coesão. A coesão garante que as responsabilidades sejam definidas de maneira satisfatória, ou seja, uma classe ou um método não devem assumir responsabilidades ou comportamentos que não são seus.

A característica principal do PIM é ser independente de plataforma, mas isso não quer dizer que a solução do problema, que é modelada usando o PIM, não apresente características recorrentes a uma plataforma específica, mas sim que a solução será modelada sem considerar qualquer mecanismo oferecido por uma plataforma específica. Por exemplo, se a solução computacional de um problema é feita usando *MapReduce* é por que o problema tem características de *Map* (dividir) ou *Reduce* (sintetizar). Assim, pode-se representar esse comportamento de *Map* e de *Reduce* sem utilizar alguma plataforma específica. Mas a questão é: como é possível separar o que é comportamento *Map*, de um comportamento *Reduce* ou de um comportamento tradicional dentro da solução que será desenvolvida no PIM? Os métodos inseridos no PIM, dentro do contexto desse trabalho, são gerados de forma coesa, ou seja, cada um tem uma responsabilidade ou comportamento bem definido, e essas responsabilidades ou comportamentos são mapeadas em: *Map*, *Reduce* ou tradicional. Esse mapeamento feito facilitará no entrelaçamento entre o PIM e o PDM.

Por exemplo, uma das possíveis soluções do problema contar palavras[3] pode ser resumida nos seguintes passos lógicos:

1. **Obter o texto**: recuperar o texto;
2. **separar as palavras**: fazer com que o texto se torne uma coleção de palavras;
3. **agrupar as palavras**: juntar as palavras idênticas;
4. **contar as palavras**: contar quantas palavras existem por grupo.

Uma possível solução computacional é apresentada na Listagem do Código 4.1. Nessa listagem, é possível observar que todos os passos lógicos mencionados estão agrupados em um único método como, por exemplo, o método *main*. É possível perceber que a solução adotada nesse caso não é coesa, inviabilizando, assim, uma possível classificação do comportamento do método *main*, visto que ele possui mais de um comportamento associado.

Outra possível solução é apresentada na Listagem do Código 4.2. Nessa listagem, é possível observar que o método *main* delegou funções, ou seja,

[3]O problema de contar palavras será um dos exemplos ilustrativos que serão apresentados para testar a ferramenta construída usando *framework* proposto

Código 4.1: Contar palavras sem coesão.

```
class ContarPalavras {

 public void main(){
  1. Obter o texto;
  2. separar as palavras;
  3. agrupar as palavras;
  4. contar as palavras;
 }

}
```

outros métodos assumiram alguns dos comportamentos existentes no método *main*, que foi apresentado na Listagem do Código 4.1. Dessa forma, a classificação do comportamento pode ser feita, visto que agora cada método possui um comportamento bem definido, ou seja, coeso. Geralmente, os métodos que manipulam coleções e que necessitem trabalhar uma grande quantidade de dados serão os métodos que irão ser classificados como *Map* ou *Reduce*, e os demais métodos irão ser classificados como tradicionais. Por exemplo, os métodos *separarPalavras* e *agruparPalavras* podem ser classificados como *Map*, enquanto o método *contarPalavras* pode ser classificado como *Reduce*, os demais métodos seriam métodos tradicionais.

Código 4.2: Contar palavras com coesão.

```
class ContarPalavras {

 public void main(){
  1. obterTexto();
  2. separarPalavras();
  3. agruparPalavras();
  4. contarPalavras();
 }

 public String obterTexto(...){
     ...
    }
 public List separarPalavras(...){
     ...
    }
 public Map agruparPalavras(...){
     ...
    }
 public Map contarPalavras(...){
     ...
    }

}
```

Quando os exemplos ilustrativos no Capítulo 6 forem apresentados, uma das primeiras ações será mapear os métodos do PIM de forma que eles apresentem comportamento coeso, para que possam ser mapeados entre um dos comportamentos

definidos no PDM. Sempre que se estiver comentando sobre esse mapeamento, será adotado o seguinte nome *Big Data Behaviour Identifier (BBI)*.

4.6 Processo para Aplicação do *Framework* Proposto

Nesta seção, o processo proposto para auxiliar na utilização do F2BD (i.e. *framework* proposto) é apresentado.

O processo proposto pode ser visualizado na Figura 4.12, e seus passos são descritos a seguir:

1. **Criar o PIM**: Um modelo do negócio é criado. Nela a lógica de negócio deverá ser incluída;

2. **Criar/Alterar o PDM**: O modelo, que descreve as características *MapReduce* que serão associadas ao PIM, deve ser criado. Se esse modelo existir, basta ser atualizado ou utilizado, caso não exista, ele deve ser criado;

3. **Criar o Modelo de *Weaving***: Esse modelo relacionará o PIM e o PDM, para que seja possível determinar quais comportamentos do PIM irão assumir comportamentos de *MapReduce*;

4. **Gerar o PSM Abstrato**: Gerado através da execução do motor de transformação, tendo como entrada: PIM, PDM, Modelo de *Weaving* e definição de transformação de modelo para modelo;

5. **Criar/Alterar o modelo da API *MapReduce***: O *framework* não implementa os conceitos de *MapReduce*, mas permite que APIs existentes, que implementem, sejam usadas. Nessa etapa, um modelo de API *MapReduce* deve ser oferecida e deve estar conforme o metamodelo do PSM concreto. Se esse modelo existir, será atualizado ou utilizado, caso não exista, ele deve ser criado;

6. **Gerar o PSM concreto**: Gerado através da execução do motor de transformação, tendo como entrada: PSM abstrato, Modelo da API *MapReduce* e definição de transformação de modelo para modelo;

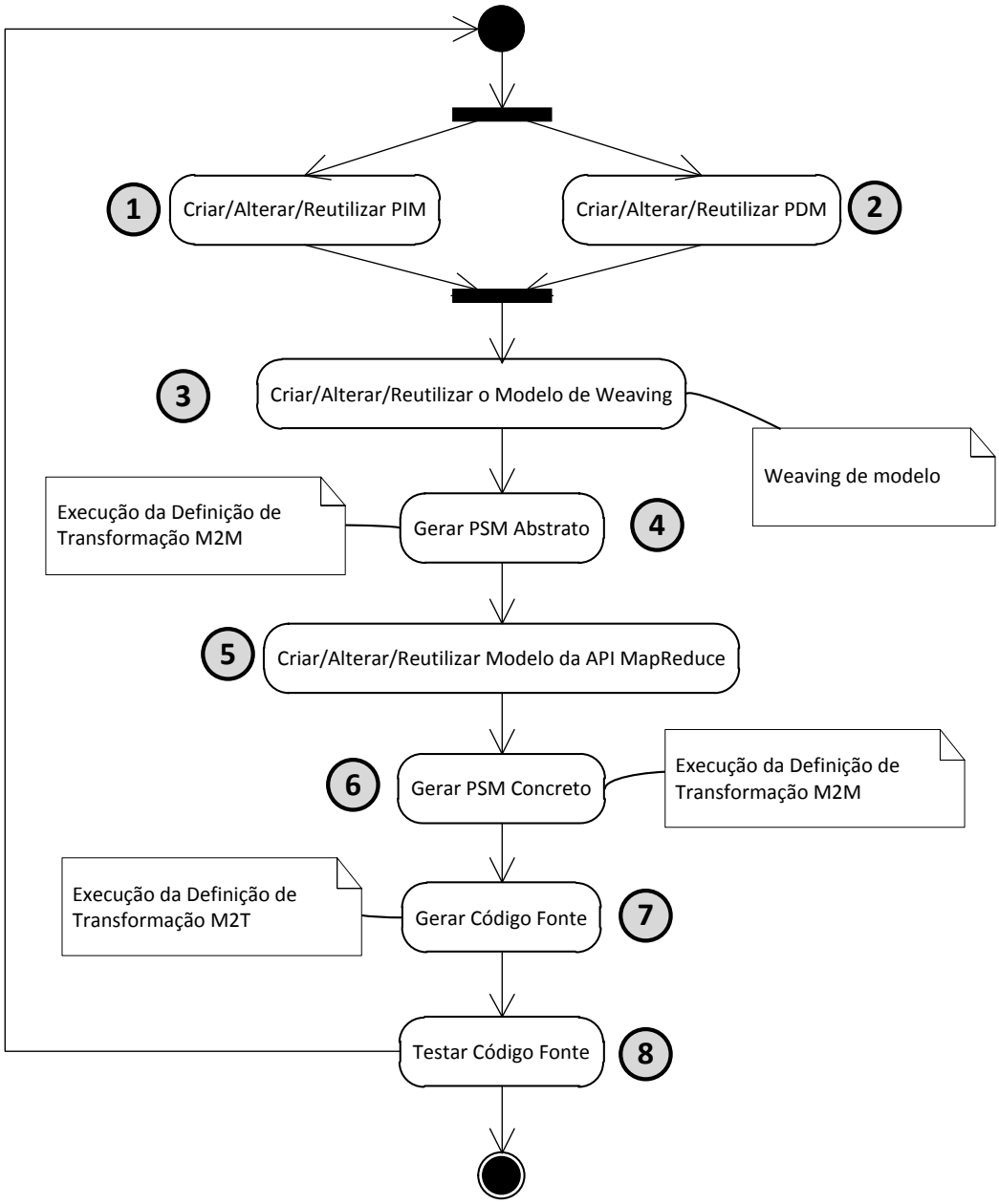


Figura 4.12: Processo para Aplicação do *Framework* Proposto.

7. **Gerar o código-fonte conforme a gramática EBNF da linguagem de programação**: Gerado através da execução do motor de transformação, tendo como entrada: PSM concreto e definição de transformação de modelo para texto;

8. **Testar o código fonte**: Esta etapa consiste em testar se o código fonte foi gerado corretamente. Após ajustes feitos, o processo deve ser executado novamente.

4.6.1 Papéis e Tarefas envolvidas

Hammoudi et al. [43] definem, em seu trabalho, dois tipos de usuários que estão diretamente ligados com o processo de transformação:

- **Usuários Especialistas**: Usuários que têm conhecimentos na criação/manipulação de metamodelos e geração de definições de transformações que serão validadas por usuários de domínio;
- **Usuários de Domínio**: Usuários que têm conhecimentos nos modelos de negócio para gerenciá-los e para executar as transformações escritas pelos usuários especialistas.

Com base nos dois tipos de papéis[4], propostos por Hammoudi et al. [43], e pensando nos papéis necessários para aplicar o *framework* proposto (i.e. F2BD), sugere-se os seguintes papeis:

- **Analista de Sistemas**: Responsável por criar, atualizar e evoluir os modelos de negócio e por definir como o sistema de *software* será concebido e modelado;
- **Desenvolvedor**: Responsável por complementar os modelos de negócio gerados pelo Analista de sistemas, detalhando a lógica do negócio dentro dos métodos modelados;
- **Analista de Modelos**: Responsável por criar e manter os metamodelos e as definições de transformações existentes.

Os dois primeiros papéis podem ser considerados usuários de domínio e o último pode ser considerado usuário especialista conforme definição de Hammoudi et al. [43]. A separação entre os papéis de "Analista de Sistemas" e "Desenvolvedor" são sugestões para empresas que conseguem organizar suas equipes com papéis bem definidos, mas em empresas de menor porte, onde não há equipe numerosa ou uma definição clara de papéis, somente um usuário de domínio poderia executar todo o processo. Para sucesso da utilização do processo é importantíssimo que exista um usuário que tenha o perfil do papel "Analista de Modelo", pois toda a

[4]Segundo Kruchten [54], "Um papel é uma definição abstrata de um conjunto de atividades executadas e dos respectivos artefatos"

estruturação que fundamenta a implementação da arquitetura do *framework* seria de sua responsabilidade.

A Tabela 4.1 descreve as tarefas definidas no processo proposto juntamente com os papéis responsáveis por cada tarefa. Nessa tabela, um resumo de como as tarefas de desenvolvimento estão associadas com as tarefas definidas no processo proposto, além de determinar como a tarefa do processo proposto será executada (manual ou automaticamente), por qual papel e qual será o artefato produzido depois de sua execução. Finalmente, um breve comentário da tarefa é apresentado na última coluna.

Tabela 4.1: Descrição das tarefas, dos artefatos e dos papéis do processo proposto.

Atividades do Processo Proposto	Tarefas da Atividade de Desenvolvimento	Como	Papel	Artefato	Comentário
Criar o PIM	Análise e Projeto	Manual	Analista de Sistemas	PIM modelado / sem comportamento	
	Implementação	Manual	Desenvolvedor	PIM modelador / com comportamento	O PIM é manipulado por dois papeis. Primeiro pelo analista de sistemas que modela as classes e a relação entre elas e em seguida pelo desenvolvedor que irá adicionar o comportamento aos métodos.
Criar/Alterar PDM	Análise e Projeto	Manual	Analista de Sistemas	PDM	O analista de sistemas é responsável por criá-lo, se ele não existir, e de atualizá-lo caso seja necessário.
Criar o Modelo de *Weaving*	Análise e Projeto	Manual	Analista de Sistemas	*Weaving* / sem comportamento	
	Implementação	Manual	Desenvolvedor	*Weaving* / com comportamento	Assim como, no PIM o modelo de *weaving* será entregue em duas etapas. Na primeira, o analista de sistemas determiná quais elementos do PIM serão associados a comportamentos do PDM. Em seguida, o desenvolvedor irá implementar o comportamento *MapReduce* indicado.
Gerar o PSM Abstrato	Implementação	Automática	Desenvolvedor	PSM abstrato	O desenvolvedor executa as definições de transformação criadas pelo Analista de Modelos para geração do PSM abstrato
Criar/Alterar o modelo da API {MapReduce}	Implementação	Manual	Analista de Sistemas	Modelo da API *MapReduce*	O analista de sistemas é responsável por criar o modelo da API que será usada para abstrair as complexidades apresentadas por *software* da plataforma *Big Data*.
Gerar o PSM concreto	Implementação	Automática	Desenvolvedor	PSM concreto	O desenvolvedor executa as definições de transformação criadas pelo Analista de Modelos para geração do PSM concreto.
Gerar Código fonte	Implementação	Automática	Desenvolvedor	Código fonte em forma textual	O desenvolvedor executa as definições de transformação criadas pelo Analista de Modelos para geração do Código fonte.
Testar o Código Fonte	Implementação	Manual	Desenvolvedor	Código final ou solicitação para ajustes	O desenvolvedor realiza testes preliminares. Caso haja problemas comunica e o processo retorna ao início.

4.7 Síntese

Este capítulo apresentou o "*Framework* para Desenvolvimento de *Software* para *MapReduce* de *Big Data*" (F2BD) que é baseado em MDE, *Weaving* e PDY. Este *framework* une teoria e prática para suportar a atividade de desenvolvimento de *software* para plataforma *Big Data* que use o modelo *MapReduce*. A arquitetura do *framework* foi apresentada, assim como alguns metamodelos propostos nela. Apresentou-se também um metamodelo e uma representação gráfica denominados de VisualAlf, que complementa o metamodelo UML e permite definir o comportamento "micro" definido nos métodos das classes de um modelo UML. Finalizando esse capítulo, um processo para utilização do *framework* proposto foi apresentado, assim como, uma discussão dos papéis envolvidos em sua utilização.

A utilização do *framework* proposto permitirá a criação de ferramentas que deem suporte à atividade de desenvolvimento de forma semiautomática, além de preservar os investimentos e facilitar tarefas de documentação, manutenção e evolução utilizando modelos. O *framework* proposto se torna uma contribuição necessária, pois considera-se que soluções que apoiem o processo de *software* como um todo ou apenas de uma de suas atividades são escassas [91].

As características de suportar tarefas de forma semiautomática, de preservar os investimentos e dar suporte na documentação, na manutenção e na evolução foram alcançadas utilizando conceitos e técnicas baseadas em MDE, assim modelos foram utilizados como artefatos em praticamente todo o *framework*.

As contribuições para o contexto de MDE consistem em:

- propostas de metamodelos, que ainda são escassos (por exemplo, até então, ausência de um metamodelo para Alf);
- uma forma de gerar o código fonte de maneira completa, pois o suporte para a descrição de uma lógica de negócio de forma "macro" e "micro" estão asseguradas, pois a extensão de UML, através da inserção do metamodelo e notação gráfica de VisualAlf, assegura a forma "micro", sendo que a forma "macro" já era assegurada por UML;

- proposta de um metamodelo de *Weaving* para permitir o entrelaçamento entre o PIM e o PDM, além de possibilitar a inclusão de comportamento nesse entrelaçamento.

Para a área de *Big Data*, a principal contribuição foi a coesão de todos os conceitos e técnicas utilizadas para que o *framework* proposto pudesse auxiliar no desenvolvimento para *Big Data*. A nossa proposta contribui para a obtenção do código fonte completo a partir de modelos, em contraste com as propostas anteriores [21, 40, 82] que não oferecem soluções para geração de código completo.

No Capítulo 5, uma ferramenta construída com base no *framework* proposto será apresentada, mostrando a viabilidade de nossas propostas.

5 Implementação de um Protótipo para o *Framework* Proposto (F2BD)

Este capítulo apresenta um protótipo de ferramenta que implementa o *Framework* para Desenvolvimento de *Software* para *MapReduce* de *Big Data*. Este protótipo de ferramenta suporta o desenvolvimento de *software*, baseado no modelo *MapReduce* do *framework Spark*. Essa ferramenta foi construída com base no processo proposto apresentado na Seção 4.6, ou seja, ela é uma das possíveis implementações baseadas no *framework* proposto. A ferramenta foi construída com o auxílio do EMF[1], que consiste em um *framework* facilitador da geração de código que são baseados em simples definições de modelos. Utilizou-se também as ferramentas *GenModel* e GMF. A primeira permite a geração de *plug-ins* em *Eclipse* para criação e edição dos modelos que serão manipulados pela ferramenta, e a segunda permite a geração de *plug-ins* em *Eclipse* para criação de editores, mas de forma gráfica.

O capítulo inicia com a apresentação do protótipo e da arquitetura da ferramenta. Logo após, os metamodelos específicos de plataforma para o *framework Spark* serão apresentados, seguidos das definições de transformações necessárias. O capítulo será encerrado com uma discussão sobre a ferramenta.

5.1 Protótipo da Ferramenta (TF2BD)

A ferramenta *Tool based on Framework to Develop Software Systems to MapReduce of Big Data (TF2BD)* suporta o desenvolvimento de *software*, baseado em *MapReduce*, usando o *framework Spark* e a linguagem Java. O *framework Spark* foi escolhido, pois oferece a implementação dos conceitos de *MapReduce* para três linguagens diferentes (*Scala*, *Python* e Java) e por oferecer performance na execução superior que seu maior concorrente o *Hadoop*. A ferramenta só disponibiliza o

[1]EMF é um *framework* de modelagem que permite criar modelos, utilizando XML *Schema*, diagramas UML ou interfaces Java, e gerar classes implementáveis a partir deles explorando as facilidades providas pela *Integrated Development Environment (IDE) Eclipse* [95]

código fonte para linguagem Java, mas podendo ser estendida posteriormente para as linguagens *Python* e *Scala*.

Os principais componentes da ferramenta serão construídos como *plug-ins* para *Eclipse*, onde a ferramenta *TF2BD*, que também será um *plug-in*, funciona como uma agregadora de todos esses componentes. A ferramenta foi implementada como um *plug-in* do tipo *view* da *IDE Eclipse*. Ela é composta das seguintes áreas:

- **Editores**: Essa área é responsável por apresentar os editores que a ferramenta disponibiliza. Cada editor permitirá criar e editar modelos;
- **Área de Edição de Modelos**: Local onde os modelos serão criados ou editados;
- **Propriedade dos Elementos**: Local onde as propriedades dos elementos dos modelos serão definidas.

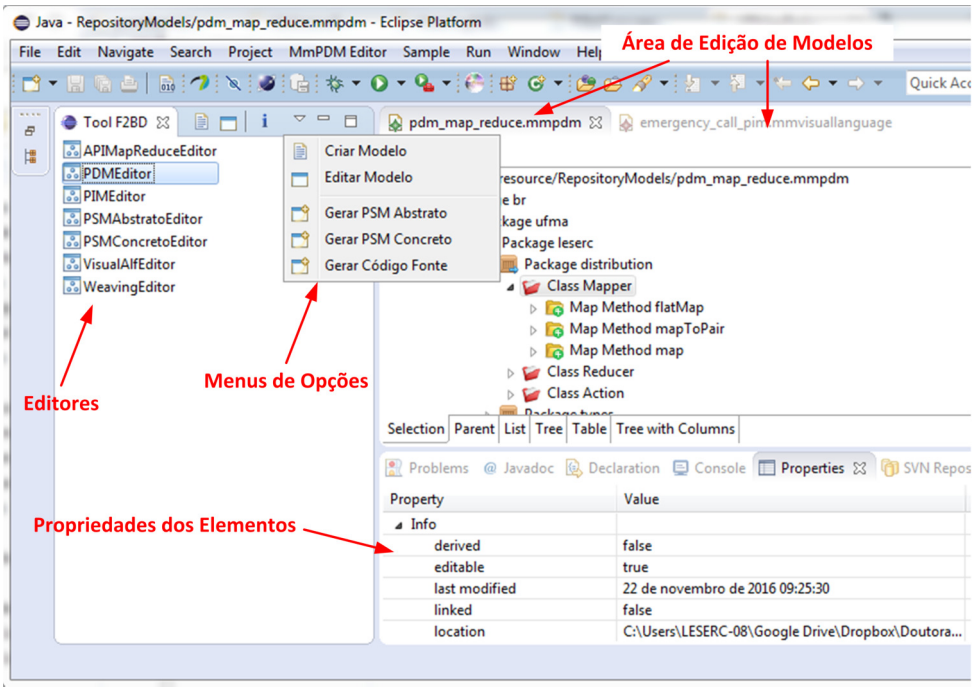


Figura 5.1: Protótipo da Ferramenta que implementa o *Framework* Proposto.

A ferramenta pode ser visualizada através da Figura 5.1. Nessa figura, pode-se visualizar as áreas descritas, assim como dois modelos abertos para edição: um PDM e um PIM, além do menu de opções que a ferramenta disponibiliza. As principais opções são: criar, editar e gerar modelos. A criação e edição são feitas manualmente e a geração é feita automaticamente, através de transformações. Todas as opções existentes nessa ferramenta são de responsabilidade dos usuários de domínio, conforme discussão realizada na Seção 4.6.1.

5.2 Arquitetura do Protótipo da Ferramenta (TF2BD)

A Figura 5.2 apresenta a arquitetura do protótipo que implementa *TF2BD*, que é baseada no processo proposto na Seção 4.6. Essa arquitetura é uma adequação baseada na arquitetura apresentada por Pablo et al. [63].

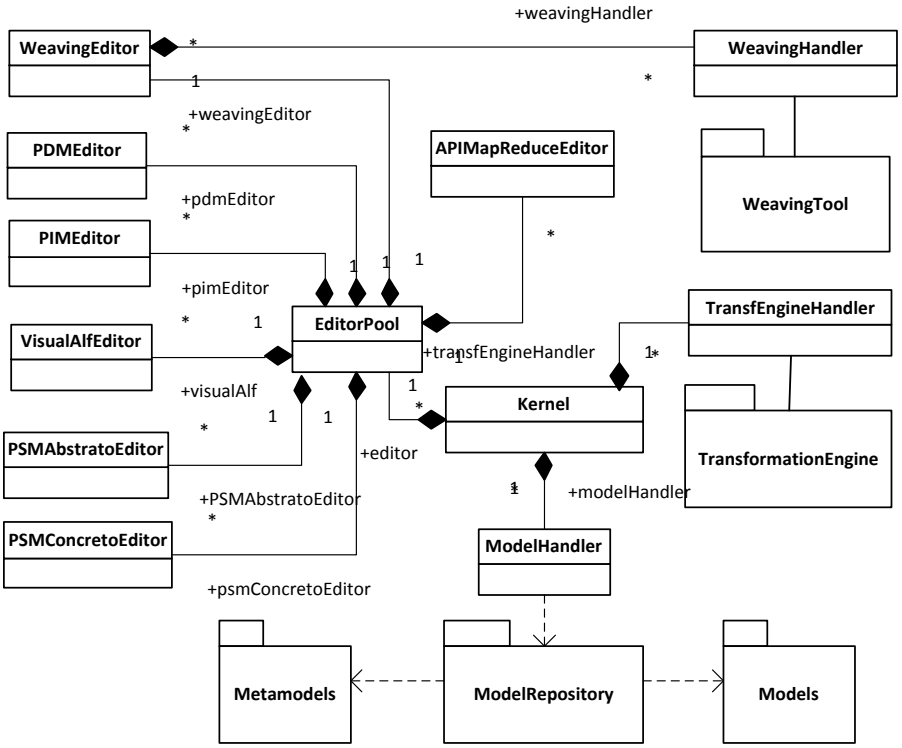


Figura 5.2: Arquitetura da Ferramenta baseada no *Framework* para Desenvolvimento de *Software* para *MapReduce* de *Big Data* (TF2BD)

A classe *Kernel* contém as funcionalidades básicas para inicializar o *plug-in*. Essa classe agrega três outras classes que são fundamentais para criação do *plug-in*:

- **EditorPool**: Essa classe gerencia todos os editores que os usuários de domínio irão precisar para realizar as atividades de desenvolvimento: *PIMEditor*, *VisualAlfEditor*, *PDMEditor*, *WeavingEditor*, *PSMAbstatoEditor*, *APIMapReduceEditor*, *PSMConcretoEditor*. Todos esses editores foram implementados como *plug-ins* e serão apresentados no Apêndice C;

- **ModelHandler**: Essa classe acessa o repositório de modelos (*ModelRepository*), permitindo que o *Kernel* possa manipulá-los;

- **TransfEngineHandler**: Essa classe permite que as definições de transformações criadas sejam executadas por um motor de transformação (*TransformationEngine*).

A arquitetura do protótipo foi construída utilizando os seguintes *software*: *Java SE Development Kit (J2SDK)* versão 7, *Eclipse* IDE versão Luna *Service Release* 2, *Eclipse Modeling Framework* versão 2.10.2 e ATL SDK *Eclipse* versão 3.5. O editor de VisualAlf foi desenvolvido usando *Graphical Modeling Framework (GMF)* versão 3.2.1.

5.3 Metamodelos Específicos de Plataforma

Nesta seção, os metamodelos específicos de plataforma, que são descritos no *framework* proposto, são apresentados. Esses metamodelos não são implementados no *framework* proposto, e sim no ato da criação da ferramenta, pois assim, o *framework* fica flexível e pode ser estendido para a plataforma que se desejar. O usuário responsável pela criação desses metamodelos é o usuário especialista, como anteriormente discutido na Seção 4.6.1.

O primeiro metamodelo específico de plataforma descreve o *framework Spark* de maneira abstrata sem levar em conta questões de implementação na Seção 5.3.1, enquanto o segundo metamodelo descreve *Spark* para a linguagem Java na Seção 5.3.2.

5.3.1 Metamodelo do PSM Abstrato para *Spark*

O *framework Spark* implementa os conceitos de *MapReduce* em três linguagens: *Java*, *Python* e *Scala*. Assim, o metamodelo do PSM abstrato irá descrever os conceitos de *MapReduce*, utilizados por *Spark* sem levar em consideração questões de implementação de nenhuma das linguagens utilizada por ele. A vantagem dessa abordagem é que a partir do PSM abstrato construído a implementação poderá ocorrer para qualquer umas das linguagens que *Spark* suporta.

Um fragmento do metamodelo do PSM abstrato pode ser visto na Figura 5.3. O metamodelo do PSM abstrato proposto contém classes que descrevem o comportamento da API de *Spark*, sem direcionar para nenhuma das linguagens oferecidas por ele.

As principais classes do metamodelo são descritas a seguir:

- **SparkPackage:** permite o agrupamento de vários classificadores, permitindo que a organização do código permaneça;

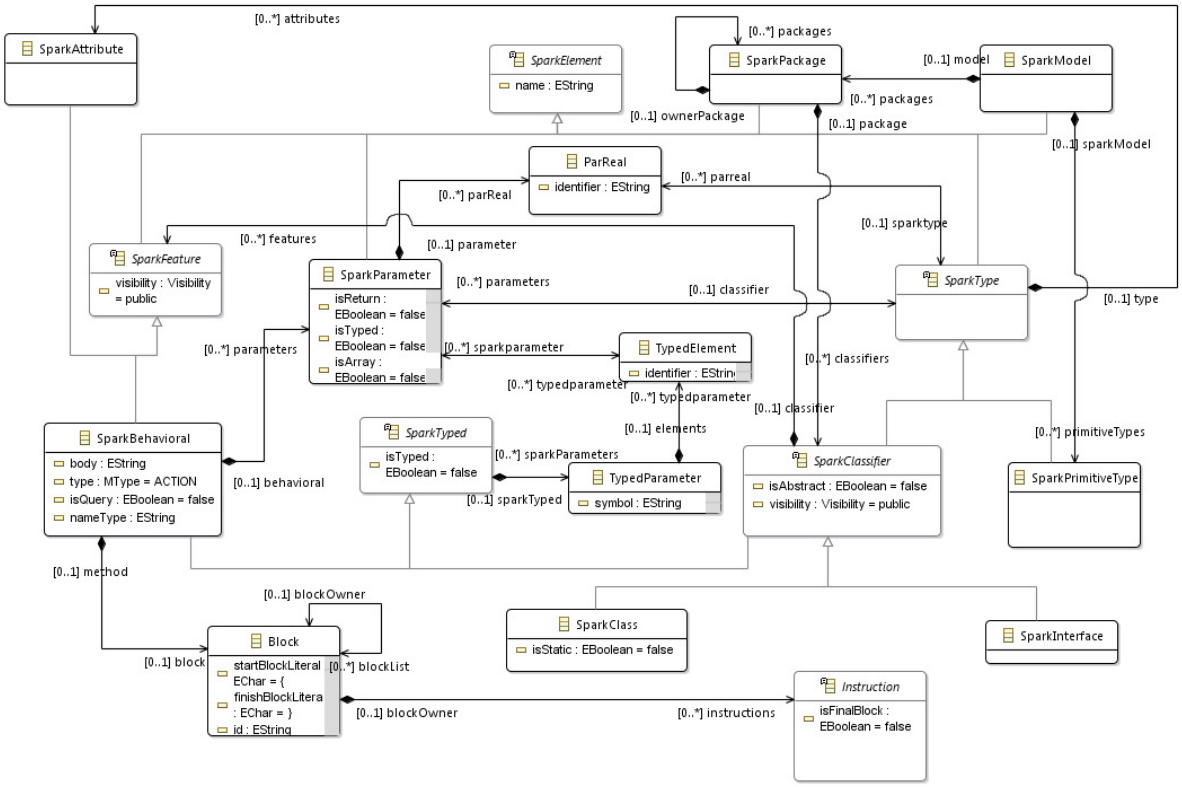


Figura 5.3: Fragmento do Metamodelo do PSM Abstrato para *Spark*

- **SparkClassifier:** Super classe abstrata que define novas classes ou interfaces, através das classes *SparkClass* e *SparkInterface*;
- **SparkFeature:** Super classe abstrata que atribui características a um classificador, por exemplo, atributos (*SparkAttribute*) e métodos (*SparkBehavioral*);
- **SparkBehavioral:** Classe que descreve métodos para um classificador. A instância da classe *SparkBehavioral* pode adicionar comportamento executável, através da classe *Block* e da super classe abstrata *Instruction*.

5.3.2 Metamodelo do PSM Concreto para Java

Nessa ferramenta, foi construído apenas o metamodelo para linguagem Java, para que a API de *Spark* para Java possa ser modelada. Com o modelo da API *MapReduce Spark* criado e com base no metamodelo da linguagem Java, o PSM concreto pode ser gerado.

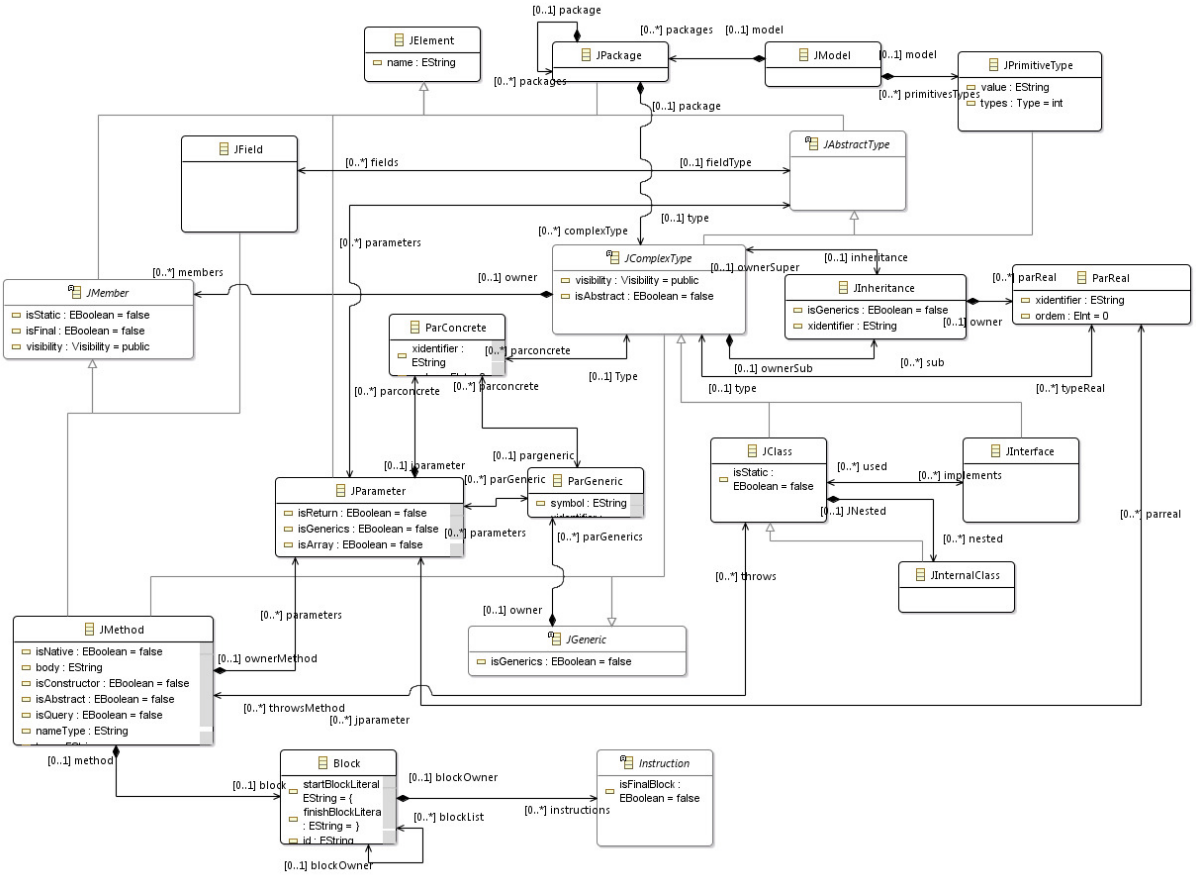


Figura 5.4: Fragmento do Metamodelo do PSM Concreto para Linguagem Java

Um fragmento do metamodelo do PSM concreto, desenvolvido conforme a gramática EBNF da linguagem Java, é apresentado na Figura 5.4. As principais classes são descritas abaixo:

- **JPackage:** Agrupa classes e interfaces para organização do código;
- **JComplexType:** Super classe abstrata que agrupa o comportamento de classificadores da linguagem. As subclasses são *JClass* e *JInterface*;
- **JMember:** Super classe abstrata que agrega características das instâncias das classe que herdam da super classe *JComplexType*. As subclasses diretas são *JAttribute* e *JMethod*;
- **JMethod:** Permite descrever comportamento executável para um classificador, através das classes *Block* e da super classe abstrata *Instruction*.

5.4 Definições de Transformação

Um aspecto fundamental descrito pelo *framework* proposto é a automaticidade da geração do código. Esse aspecto é alcançado através da criação e posterior execução das definições de transformações propostas. Essas transformações estão diretamente relacionadas com os modelos específicos de plataforma, logo só podem ser definidas no ato da implementação da ferramenta. O protótipo *TF2BD* implementa três definições de transformação que vão permitir gerar os modelos específicos de plataforma e, posteriormente, o código fonte. As definições de transformação são as seguintes:

- **Modelos de Entrada para PSM Abstrato:** É a primeira definição de transformação, é do tipo de modelo para modelo, e tem como entrada o PIM, o PDM e o Modelo de *Weaving*, e como saída o PSM abstrato. Esta definição de transformação tem a responsabilidade de começar o processo de diminuição de abstração, ou seja, tornar os modelos independentes de plataforma em um modelo dependente de plataforma;

- **PSM Abstrato para PSM Concreto:** É a próxima definição de transformação, também do tipo modelo para modelo e tem como entrada o PSM abstrato e o modelo da API *MapReduce* de *Spark*, e gera como saída o PSM concreto. O processo de diminuição de abstração continua, pois agora o objetivo da definição de transformação é sair de um modelo que descreve um plataforma de forma abstrata, para um modelo que implementa uma plataforma de forma concreta. O PSM Concreto, resultante dessa transformação, é a representação do código fonte final em forma de modelo;

- **PSM Concreto para Código fonte:** É a última definição de transformação e tem como entrada o PSM concreto, e gera como saída o código fonte do sistema de *software* e outros artefatos, como arquivos de configuração. Dessa forma, esta definição de transformação será de modelo para texto e tem como objetivo disponibilizar o código fonte em formato textual para compilação e execução.

As definições de transformação de modelo para modelo e de modelo para texto, para construção dessa ferramenta, foram todas escritas usando a *Atlas*

Transformation Language (ATL) [31, 39]. ATL foi escolhida por fornecer um meio de produzir uma série de modelos de destino a partir de um conjunto de modelos de origem, por poder realizar transformações de modelo para modelo e de modelo para texto, por ser uma linguagem híbrida e por ter integração com o Eclipse por meio de *plug-in*.

5.4.1 Regras de Transformação

A Listagem do Código 5.1 apresenta um fragmento da definição de transformação dos modelos de entrada para PSM Abstrato com algumas de suas regras de transformação. Estas regras de transformação mostram as correspondências entre os elementos do PIM, do PDM, do modelo de *Weaving* e o PSM Abstrato como a seguir:

- **UNamespace2SparkPackage:** transforma os elementos *UNamespace* do PIM nos elementos *SparkPackage* do PSM abstrato. Está disponível entre as linhas 7 e 17 da Listagem do Código 5.1;
- **UClass2SparkClass:** transforma os elementos *UClass* do PIM nos elementos *SparkClass* do PSM abstrato. Está disponível entre as linhas 18 e 28 da Listagem do Código 5.1;
- **UMethod2SparkBehavioral:** transforma os elementos *UMethod* do PIM nos elementos *SparkBehavioral* do PSM abstrato. Está disponível entre as linhas 29 e 55 da Listagem do Código 5.1.

Também pode-se ver na Listagem do Código 5.1, entre as linhas 29 e 55, que o modelo de *weaving* é consultado para auxiliar na regra de transformação *UMethod2SparkBehavioral*, e que uma regra **lazy rule block** é chamada na linha 52. A regra **lazy rule block** é responsável por transformar as instruções de um modelo para outro. As transformações que lidam com comportamento, neste trabalho, todas têm a característica de **lazy rule**, pois esta regra de transformação permite que o controle das transformações possa ser feito de maneira imperativa. Este comportamento é desejável, para que se possa manter a ordem lógica das instruções que foram modeladas. As demais regras de transformações foram omitidas por questões de limitação de espaço.

Código 5.1: Definição de Transformação escrita em ATL para criar o PSM abstrato a partir dos modelos de entrada

```
module umlVisualandPDM2PSMAbstract;
create OUT: mmPSMAbstractSpark from
    mPDM: mmPDM,
  mUMLVisual: mmUMLVisual,
  mMergeUMLVisual2PDMBigDistribution: mmMerge;

rule UNamespace2SparkPackage {
from
  package:
     mmUMLVisual!UNamespace (
         package.oclIsTypeOf(mmUMLVisual!UNamespace) )
to
  sPackage:
     mmPSMAbstractSpark!SparkPackage (
       name <- package.name,
       ownerPackage <- package.namespace  )
}
rule UClass2SparkClass {
from
  uClass:
    mmUMLVisual!UClass (
      uClass.oclIsTypeOf(mmUMLVisual!UClass) )
to
  sparkClass:
     mmPSMAbstractSpark!SparkClass (
       name <- uClass.name,
       package <- uClass.namespace )
}
rule UMethod2SparkBehavioral {
from
  uMethod:
     mmUMLVisual!UMethod(
       uMethod.oclIsTypeOf(mmUMLVisual!UMethod) and
        mmMerge!WItem.allInstances() ->
         select(m | m.wLeftItem.type = 'UMethod' and
              uMethod.name = m.wLeftItem.name and
          uMethod.owner.name = m.wLeftItem.owner.name)
          -> notEmpty() )
to
  sparkMethod:
      mmPSMAbstractSpark!SparkBehavioral (
        classifier <- uMethod.owner,
      name <- uMethod.name,
      isQuery <- uMethod.isQuery,
      nameType <- sparkMethod.nameType(
         sparkMethod.isMapReduce(uMethod.name).first()),
      type <- sparkMethod.nameItem(
         (sparkMethod.isMapReduce(uMethod.name)).first()),
      parameters <- thisModule.getParametersFromWaving(
         uMethod.name, uMethod.owner.name)
       ->collect(p | thisModule.paramter(p,sparkMethod)),
      block <-  thisModule.block(
        thisModule.getBlockFromWaving(uMethod.name,
         uMethod.owner.name),sparkMethod, OclUndefined) )
}
```

A Listagem do Código 5.2 apresenta um fragmento da regra **lazy block**, que percorre uma coleção com todas as instruções pertencentes ao bloco, com o objetivo de chamar a regra de transformação **lazy block** adequada para cada instrução.

Código 5.2: Fragmento da regra *lazy block* que chama a regra de transformação para cada instrução

```
instructions <-
  if fromBlk <> OclUndefined then
    fromBlk.instructions->collect(i |
   if i.oclIsTypeOf(mmPSMA!VariableDeclaration) then
      thisModule.variableDeclaration(i, toBlk)
   else if i.oclIsTypeOf(mmPSMA!AttributionCommand)  then
       thisModule.attributionCmd(i, toBlk, true, toBlockMetodo)
   else if i.oclIsTypeOf(mmPSMA!forStatement) then
       thisModule.forStatement(i, toBlk)
   else if i.oclIsTypeOf(mmPSMA!ifStatement) then
       thisModule.ifStatement(i, toBlk)
   else if i.oclIsTypeOf(mmPSMA!ReturnInstruction)then
       thisModule.returnInstruction(i, toBlk, toBlockMetodo)
   else if i.oclIsTypeOf(mmPSMA!Invocation) then
       thisModule.resultInvocation( i, toBlk, toBlockMetodo ) --ok no codigo
   else OclUndefined
   endif
   endif
   endif
   endif
   endif
   endif )
  else
 OclUndefined
  endif
```

As demais regras de transformação que compõem as definições de transformação do PSM abstrato para o PSM concreto e do PSM concreto para o código fonte seguem a mesma linha de raciocínio das regras de transformação detalhadas nesta seção, claro que observando os modelos de entrada e os respectivos modelos de saída. Outro aspecto importante, é que essas definições devem ser feitas por um usuário especialista, conforme discutido na Seção 4.6.1.

5.4.2 Discussão

Para Cabot et al. [18], o desenvolvimento de *software*, baseado em modelos, pode ser resumido na seguinte operação: "***modelos + transformações = software***". Partindo-se da afirmação de Cabot et al. [18] e considerando que um *software* é composto de uma visão macro (esqueleto dos métodos) e de uma visão micro (lógica do negócio), então poderia concluir-se que os modelos, nessas abordagens, deveriam descrever tanto a visão macro como a visão micro, visto que as transformações apenas geram outros modelos.

Dessa forma, a lógica de negócio não deve ser preservada apenas como um artefato de documentação de um *software*, mas como um artefato primordial que deve ser incorporado no modelo de forma completa (visão macro e micro), padrão (conforme uma linguagem) e homogênea (no mesmo local) para descrever um *software*. Isso possibilitará a automatização do desenvolvimento no mesmo passo que facilitará a manutenção e a evolução dos *software* construídos, além de preservar os investimentos.

A automaticidade do desenvolvimento, utilizando abordagens baseadas em modelos, é alcançada através das transformações. Porém, muitos trabalhos na literatura apenas oferecem soluções para a visão macro de um *software* utilizando modelos. Neste trabalho, procurou-se oferecer uma solução que permitisse ir além da visão macro, ou seja, oferecer também a visão micro, consultar Seção 4.4. Na Seção 5.4, apresentou-se as definições de transformação utilizadas para garantir que o *software*, em sua visão macro e micro, pudesse ser gerado, enquanto que na Seção 5.4.1 algumas regras de transformação foram detalhadas e nesta seção, uma discussão em termos de desafios e limitações sobre as transformações é apresentada.

Um dos desafios encontrados é que os motores de transformação manipulam as transformações de modelos de forma declarativa, isto quer dizer que eles geram os modelos elemento a elemento. Por exemplo, o motor de transformação gera todas as instâncias dos elementos do tipo “X” de um modelo “A” para todos os elementos do tipo “Y” de um modelo “B”. Quando se pensa em termos de lógica de negócio, a ordem dos elementos é importante. Assim, usar a forma declarativa retira toda a ordem da lógica prescrita no modelo de origem quando a transcreve para o modelo de destino. A solução adotada para superar esse desafio foi usar uma abordagem mista: para os elementos que descrevem a visão macro do modelo foi utilizada a abordagem declarativa, enquanto que para os elementos que representam a lógica de negócio foi utilizada a abordagem imperativa. A abordagem imperativa foi adotada para visão micro, porque permite que as transformações ocorram não elemento a elemento, mas na ordem que as transformações são executadas. Na Seção 5.4.1, regras usando a forma imperativa foram apresentadas.

Desafiador também foi a busca de elementos dentro de um modelo quando a abordagem imperativa era utilizada. A utilização da abordagem imperativa tem um custo, pois a automaticidade das referências entre elementos é perdida e fica a cargo do desenvolvedor encontrá-las. Por exemplo, o tipo de dado de uma variável

no processo de transformação não é trazido automaticamente. Para solucionar essa questão, o elemento proprietário da variável era passado como parâmetro nas regras de transformação, já para os demais elementos foram construídos *helpers*, funções de buscas por tipos de elementos, pois cada um apresentava uma peculiaridade. A solução apresentada para esse desafio é limitada ao nome dos elementos, por isso elementos com o mesmo nome podem ser confundidos. Esse impacto foi diminuído comparando o tipo e o proprietário do dado, além do seu próprio nome. Outra limitação para as transformações implementadas de forma imperativa é que elas não consideram hierarquia de classes, ou seja, os elementos de uma classe pai não são considerados na busca. A questão da hierarquia deve ser acrescida às funções de buscas, quando for conveniente, para que essa limitação seja resolvida.

Outro desafio, foi o mapeamento dinâmico de tipos de dados. Os modelos apresentavam visões independentes e tipos de dados peculiares que só são encontrados quando se chega ao nível de plataforma específica. Como descrever se um tipo de dado usado no modelo independente de plataforma irá mudar para um elemento específico da plataforma? Por exemplo, como saber se um tipo de dado "List" deve ser convertido em um "JavaRDD"? Para solução desse problema, marcadores foram utilizados nos modelos para indicar se a mudança deveria ocorrer e os tipos de dados foram mapeados estaticamente nas regras de transformação. Essa solução é limitada ao número de mapeamentos de tipos realizados nas regras. Caso um novo mapeamento exista, a regra de transformação deve ser atualizada fazendo com que o processo se torne um pouco burocrático.

Mesmo com os desafios e limitações apresentadas, as transformações propiciam uma forma eficaz de automatizar o desenvolvimento completo do *software* utilizando modelos, garantindo que modelo e documentação estejam sempre sincronizados, evitando retrabalho.

5.5 Síntese

Neste capítulo, apresentou-se um protótipo da ferramenta que foi implementada com base no *framework* proposto no Capítulo 4. Para construção da ferramenta foi escolhida a implementação de *MapReduce* utilizada pelo *framework*

Spark. Com base nessa plataforma, dois metamodelos específicos de plataforma foram criados, PSM abstrato e PSM concreto, conforme a arquitetura do *framework* proposto, assim como, alguns trechos das definições de transformação exigidas para que a automaticidade ocorra. Para todos os modelos manipulados pelo protótipo, foram criados editores para facilitar a sua criação, edição ou apenas a visualização.

A ferramenta proposta, i.e. *TF2BD*, suporta a atividade de desenvolvimento de *software* para plataforma *Spark* de forma semiautomática, auxilia na preservação da lógica de negócio, assim como, na documentação, na manutenção e evolução do *software* desenvolvido. A *TF2BD* auxilia de forma semiautomática, pois os modelos de entrada são todos construídos manualmente pelo usuário, enquanto os específicos de plataforma são gerados automaticamente, através de definições de transformação. A *TF2BD* preserva, ainda, a lógica de negócio, pois ela é incorporada no PIM que é independente de plataforma. Este mesmo PIM pode ser relacionado com um PDM, por meio de um modelo de *Weaving* e todos eles serem utilizados para gerar um PSM abstrato, dependente de plataforma. A *TF2BD* auxilia, também, em tarefas de documentação, pois um modelo nada mais é que a especificação daquilo que deve ser construído; auxilia na manutenção e na evolução, já que qualquer ajuste ou melhoria será feito nos modelos e as transformações serão executadas novamente.

Dois exemplos ilustrativos serão apresentados no Capítulo 6 com o intuito de demonstrar que o protótipo da ferramenta proposta, i.e. *TF2BD*, suporta a criação de *software* em *Big Data*.

6 Exemplos Ilustrativos Criados Utilizando a Ferramenta *TF2BD*

Nesse capítulo, dois exemplos ilustrativos serão apresentados. Esses dois exemplos foram construídos com o auxílio da ferramenta *TF2BD*, apresentada no Capítulo 5, que implementa a arquitetura do *framework* proposto na Seção 4.2. O objetivo desses exemplos é demonstrar que a ferramenta construída, com base no *framework* proposto, pode auxiliar nas atividades de desenvolvimento de *software* para o modelo *MapReduce* da plataforma *Big Data*, através da disponibilização de editores e automatização de tarefas. O primeiro exemplo é mais simples e visa demonstrar a ferramenta, enquanto que o segundo exemplo é mais complexo e pretende endossar a utilidade da ferramenta.

Os exemplos serão construídos seguindo as tarefas especificadas no processo proposto na Seção 4.6. Serão observados em cada tarefa: o papel responsável pela mesma, como ela será realizada e o artefato que será produzido.

O capítulo está organizado em duas seções que apresentam, respectivamente, o primeiro e o segundo exemplo ilustrativo. Em cada exemplo, as tarefas do processo serão discutidas separadamente.

6.1 Exemplo 1 - Contagem de Palavras

O primeiro exemplo utilizado é o aplicativo contar palavras (*WordCount*), disponibilizado pelo *framework Spark* na linguagem Java [93]. Esse é um aplicativo simples utilizado pelo *framework Spark* para demonstrar seu funcionamento. Ele simplesmente calcula a frequência com que as palavras contidas em um texto aparecem. Neste trabalho, esse aplicativo, assim como em *Spark*, é utilizado com o propósito de demonstrar o processo e o *framework* propostos. O caso de uso do aplicativo pode ser visualizado na Figura 6.1.

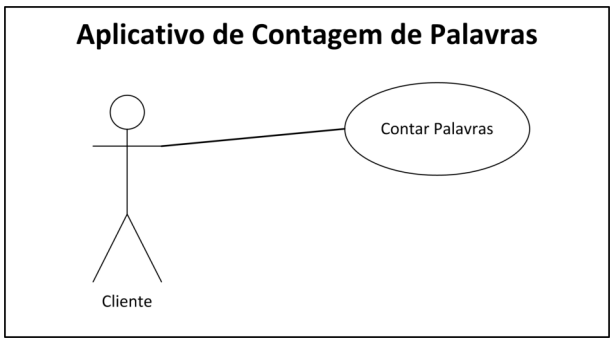


Figura 6.1: Caso de uso do aplicativo contar palavras.

6.1.1 Criação do PIM

Nesta seção, apresenta-se a criação do PIM do aplicativo *WordCount* que é a primeira tarefa do processo, conforme a Figura 4.12. O processo proposto sugere que o PIM deve ser criado em dois momentos. O primeiro momento, na fase de análise e projeto, o PIM deve ser criado pelo usuário que tiver o papel "Analista de sistemas" e deve gerar como artefato o PIM sem comportamento dos métodos definidos. Em um segundo momento, na fase de implementação, o PIM seria complementado pelo usuário que obtiver o papel "Desenvolvedor" e entregaria como artefato o PIM com a lógica de negócio incluída no modelo através da definição dos comportamentos dos métodos.

O PIM desse exemplo é composto de uma classe, *WordCount*, e de cinco métodos: *loadWord*, *splitWord*, *mapWord*, *reduceWord* e *main*. Esses métodos foram construídos conforme o mapeamento BBI proposto na Seção 4.5. A Tabela 6.1 apresenta os métodos classificados conforme o mapeamento BBI.

Tabela 6.1: Métodos do PIM WordCount classificados conforme mapeamento BBI.

Método	Classificação BBI
loadWord	Tradicional
splitWord	Map
mapWord	Map
reduceWord	Reduce
main	Tradicional

O PIM é criado de forma manual pelo usuário, e para apoiar essa criação a *TF2BD* oferece dois editores. O primeiro *PIMEditor* que permite criar o PIM em formato de árvore e o segundo *VisualAlfEditor* que permite criar o PIM de forma gráfica e interativa. A intenção por oferecer essas duas opções de editores é atender perspectivas diferentes. Existem usuários que gostam de trabalhar de

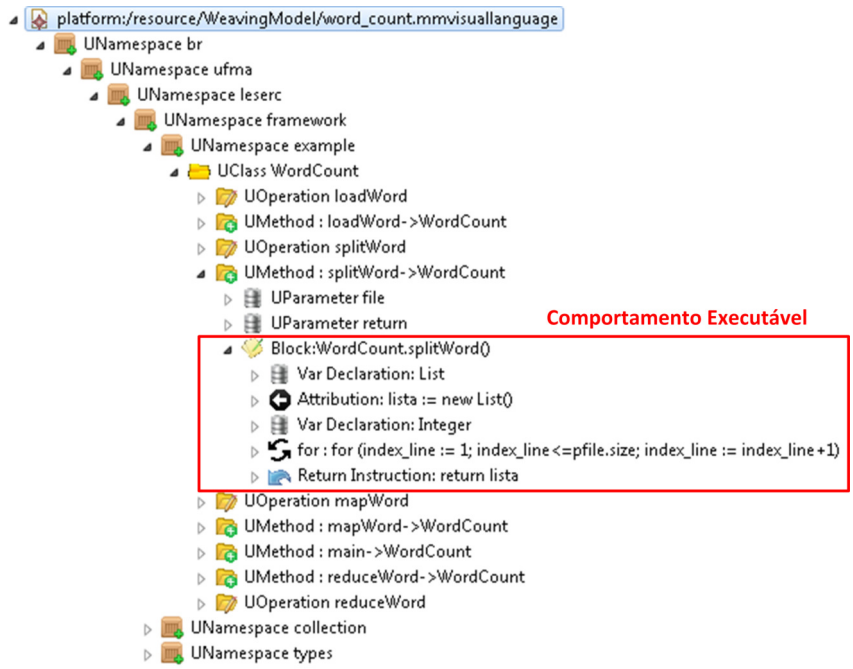


Figura 6.2: PIM criado para o aplicativo *WordCount*.

forma gráfica, enquanto existem usuários que preferem usar uma abordagem mais tradicional, próxima ao texto. Futuramente, outras possibilidades de geração podem ser consideradas para atender outros perfis, por exemplo, uma abordagem mista que oferece a agilidade da forma textual e a facilidade de interpretação oferecida pela forma gráfica.

A Figura 6.2 apresenta o PIM criado, usando o editor *PIMEditor* que gera o PIM em forma de árvore. Ainda na Figura 6.2, pode-se ver que a lógica do negócio para o método *splitWord*, representada pelo comportamento incluído, está em destaque. Isso é possível, porque o metamodelo proposto para o PIM permite que o comportamento do método possa ser incorporado ao modelo preservando, assim, a lógica de negócio.

Representação Visual da Lógica de Negócio

TF2BD também oferece uma forma visual e intuitiva para criação do PIM, através do editor *VisualAlfEditor*. Nesta seção, apresenta-se o método *mapWord* da classe *WordCount* criado, utilizando esse editor que implementa a notação gráfica proposta na Seção 4.4. Esse método é apenas um fragmento do PIM proposto em nosso exemplo ilustrativo e pode ser visualizado na Figura 6.3.

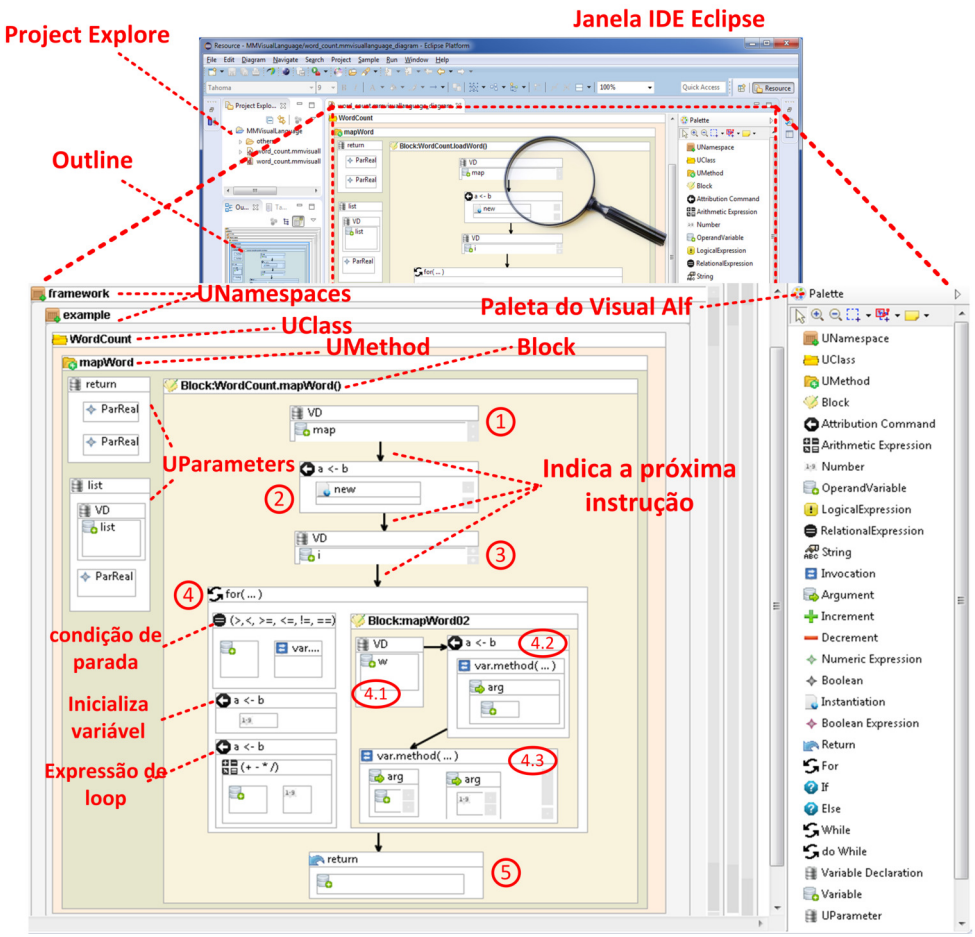


Figura 6.3: Método mapWord feito usando Visual Alf.

Pode-se ver, através da Figura 6.3, que o *namespace* "framework" contém o *namespace* "example" e que o *namespace* "example" contém a classe *WordCount*. Por sua vez, a classe *WordCount* contém o método *mapWord*. Pode-se ver, também, que o método *mapWord* contém parâmetros e um bloco (*Block*). O bloco do método *mapWord* contém cinco instruções para execução: 1 - corresponde a declaração de uma variável, 2 - a inicialização da variável anterior, 3 - a declaração de outra variável, 4 - há um laço de repetição e 5 - retorno do método. É importante destacar que a instrução que está sendo executada indicará, através de uma seta, qual será a próxima instrução da sequência.

A instrução *"for"* possui três instruções padrões: inicializar a variável, condição de parada e laço de repetição, e um bloco. Por exemplo, na Figura 6.3, vê-se as três instruções padrões e o seu bloco (*Block*), que é composto de três instruções: 4.1 - declaração de uma variável, 4.2 - inicialização da variável anterior e 4.3 - invocação de um método. Todos as instruções disponíveis pela notação gráfica estão dispostas na paleta à direita.

VisualAlf é baseada em fluxo de controle, enquanto que em algumas outras linguagens, como a utilizada por *LabVIEW*, o fluxo é baseado em dados. O fluxo de controle foi escolhido, pois é o fluxo utilizado pelas linguagens imperativas para execução de suas instruções. A linguagem Java, que é o foco do nosso trabalho, é um exemplo de linguagem imperativa.

6.1.2 Criação/Alteração do PDM

Nessa seção, apresenta-se o PDM que deve ser analisado na tarefa 2 do processo proposto. O PDM é o modelo que descreve as características de uma plataforma, que nesse trabalho, são as características do modelo *MapReduce* para *Big Data*. Segundo o processo, o PDM é o artefato produzido manualmente pelo papel "Analista de sistemas" na fase de Análise e Projeto. *TF2BD* oferece o editor *PDMEditor* para permitir que esse modelo seja criado ou editado.

Na Figura 6.4, apresenta-se o PDM criado, utilizando o editor *PDMEditor*.

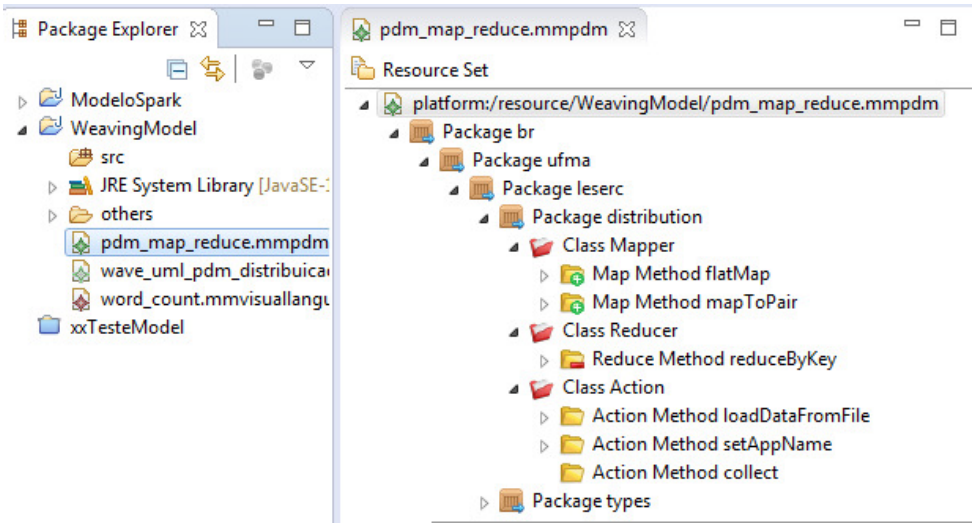


Figura 6.4: PDM para *MapReduce* criado usando *PDMEditor*.

No PDM criado, pode-se ver que dentro do pacote "distribution" existem classes que usam os conceitos de *Map* e *Reduce* adotados em alguns *software Big Data*. A classe *Mapper* tem comportamento de *Map*, a classe *Reducer* tem comportamento de *Reduce* e a classe *Action* tem comportamento tradicional.

Destaca-se que o PDM, dentro do processo proposto, é criado apenas uma vez e utilizado quantas vezes forem necessárias. Assim, se na tarefa de criação do PDM, o usuário que possuir o papel "Analista de sistemas" identificar que o

PDM existe e que não necessita de atualização, mais nada precisará ser feito. Para atualização do PDM, o usuário poderá utilizar o *PDMEditor*.

6.1.3 Criação do Modelo de Weaving

Nesta seção, apresenta-se a criação do modelo de *weaving* conforme, orienta a tarefa 3 do processo proposto.

A criação desse modelo é realizado em duas etapas, assim como, na criação do PIM apresentada na Seção 6.1.1. A criação em duas etapas ocorre, porque esse modelo agregará comportamento *MapReduce* aos métodos do PIM. A primeiro etapa será realizada pelo usuário que obtém o papel "Analista de sistemas" que fará o entrelaçamento entre os modelos do PIM e do PDM na fase de Análise e de Projeto resultando no artefato modelo de *weaving* sem comportamento, e em uma segunda etapa pelo usuário, com o papel de "Desenvolvedor" que complementará a criação do modelo de *weaving*, na fase de implementação, adicionando comportamento aos entrelaçamentos. O comportamento descrito nesse modelo irá determinar o novo comportamento que alguns métodos do PIM irão assumir nas tarefas seguintes.

O editor *WeavingEditor* oferecido pela *TF2BD* é o mecanismo utilizado pelos usuários para a criação manual do modelo de *weaving*.

A Figura 6.5 apresenta o modelo de *weaving*, que foi criado através do entrelaçamento entre os elementos do PIM e os elementos do PDM. O entrelaçamento entre os elementos correspondentes do PIM e do PDM podem ser vistos no centro. As setas que partem da esquerda para o centro e da direita para o centro, que possuem a mesma numeração, indicam os elementos da esquerda e da direita que foram relacionados no centro. O entrelaçamento realizado foi facilitado, pois o PIM foi construído levando em consideração o mapeamento BBI.

O modelo de *weaving* proposto também permite adicionar comportamento. Por exemplo, no terceiro relacionamento, que pode ser visto na Figura 6.5, detalha-se o novo comportamento necessário ao aspecto *mapToPair* do PDM, que o método *mapWord* do PIM deverá possuir quando o PSM abstrato for gerado. No caso, o comportamento desejado é o retorno de uma instância de uma coleção *Map* com dois argumentos: *mapVar*, expressado pela classe *OperandVariable*, e um número literal.

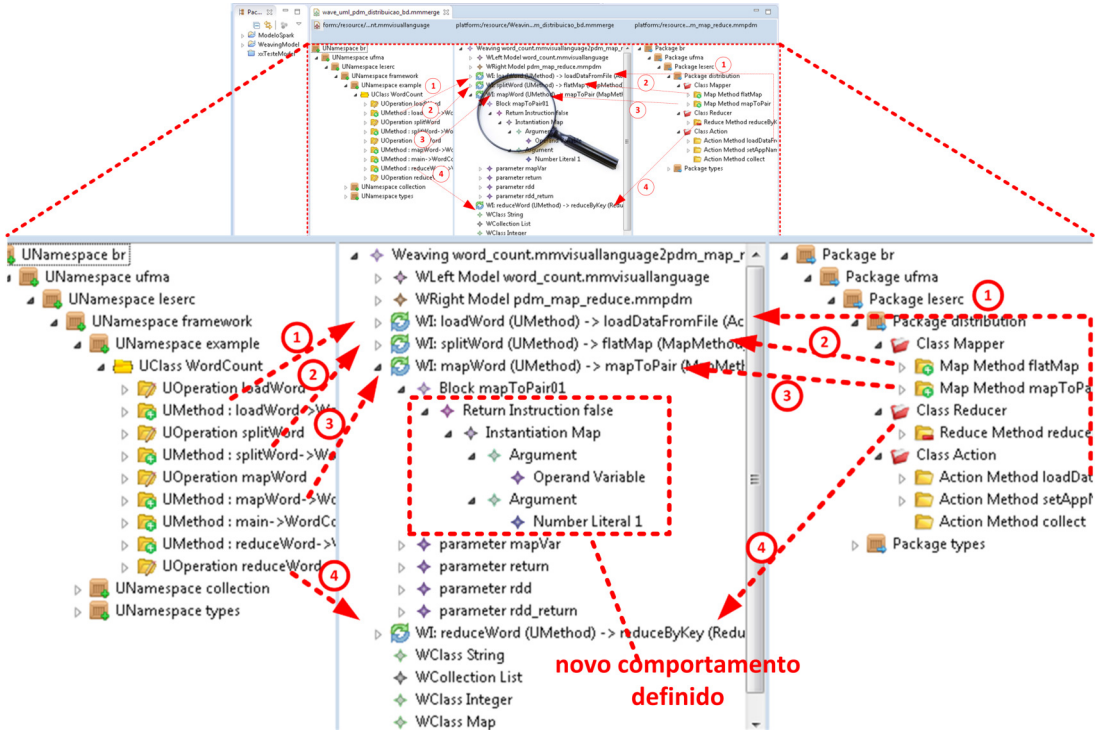


Figura 6.5: Modelo de *Weaving* relacionando o PIM e o PDM através do *WeavingEditor*.

Desse modo, mantêm-se a integridade original do PIM e do PDM ao mesmo tempo que se consegue gerar um PSM abstrato, incorporando aspecto *Big Data* à solução.

É importante ressaltar que o modelo de *weaving*, que pode ser criado nesse trabalho, permite a inserção da lógica de negócio durante o entrelaçamento dos elementos do PIM e do PDM. Diferentemente do que acontece em trabalhos como o de Fabro et al. [35] e de Pablo et al. [63], que apresentam a impossibilidade da inserção de código (textual ou visual) para preservar a lógica do negócio durante o entrelaçamento entre o PIM e o PDM;

6.1.4 Geração do PSM Abstrato

O PIM, o PDM e o modelo de *Weaving* foram criados de maneira manual pelos usuários, seguindo as tarefas de 1 a 3. Agora, a partir da tarefa 4, a automatização das tarefas terá início com a utilização das transformações entre modelos.

A geração do PSM Abstrato deve ser realizada pelo usuário que obtém o papel "Desenvolvedor" de forma automática, através da execução de uma definição de transformação, que receberá como entrada os modelos PIM, PDM e *Weaving*, devendo oferecer como artefato de saída o PSM abstrato, que deve estar conforme o

metamodelo proposto para o *framework Spark*, apresentado na Seção 5.3.1. A execução das transformações é permitida através da *TF2BD* e ocorrem na fase de implementação.

As definições de transformação necessárias para automatização das tarefas propostas pelo processo devem ser criadas e mantidas pelo usuário que tiver o papel "Analista de modelos". Assim, a definição de transformação criada e utilizada nessa tarefa foi elaborada por esse usuário e disponibilizada na ferramenta *Tool F2BD*.

Na Seção 5.4, foram apresentadas as definições de transformações que são utilizadas pela *TF2BD*. A Listagem do Código 5.1 apresenta um fragmento das definições de transformação, que são utilizadas pelo motor de transformação para criar o PSM abstrato, conforme a tarefa 4 da Figura 4.12.

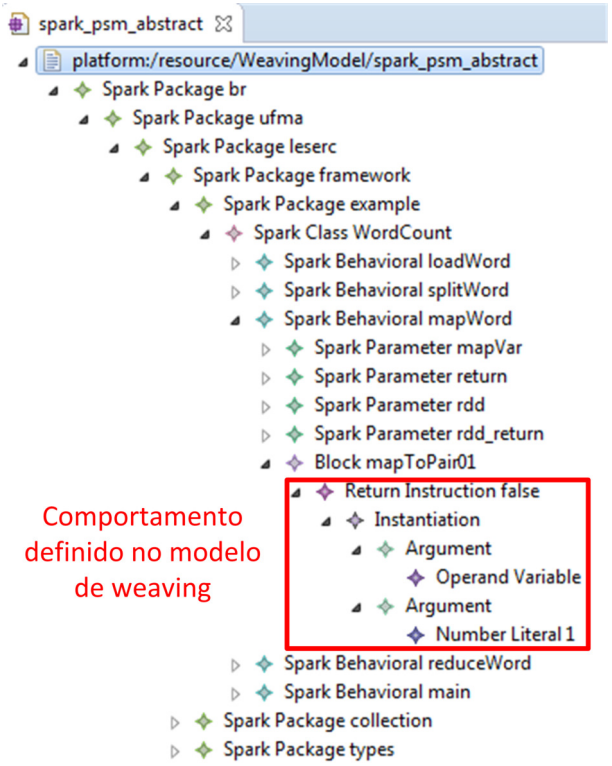


Figura 6.6: Visualização do PSM Abstrato conforme *framework Spark* usando *PSMAbstratoEditor*

O PSM abstrato, resultado de uma transformação, pode ser visualizado na Figura 6.6, utilizando o editor *PSMAbstratoEditor* oferecido pela *TF2BD*. Ainda na Figura 6.6, pode-se ver que o comportamento de "*SparkBehavioral mapWord*" é definido conforme indicado no modelo de *Weaving* da Figura 6.5. Assim, observa-se que o comportamento dos métodos do PIM serão adequados para a realidade definida no modelo de *weaving*, durante a geração do PSM abstrato, para que aspectos *Big Data* possam ser incorporados na solução sem alterar os modelos de entrada.

6.1.5 Geração do PSM Concreto

Nessa seção, apresenta-se a geração do PSM concreto conforme metamodelo apresentado na Seção 5.3.2. Esse modelo descreve os detalhes de implementação de *Spark* para linguagem Java.

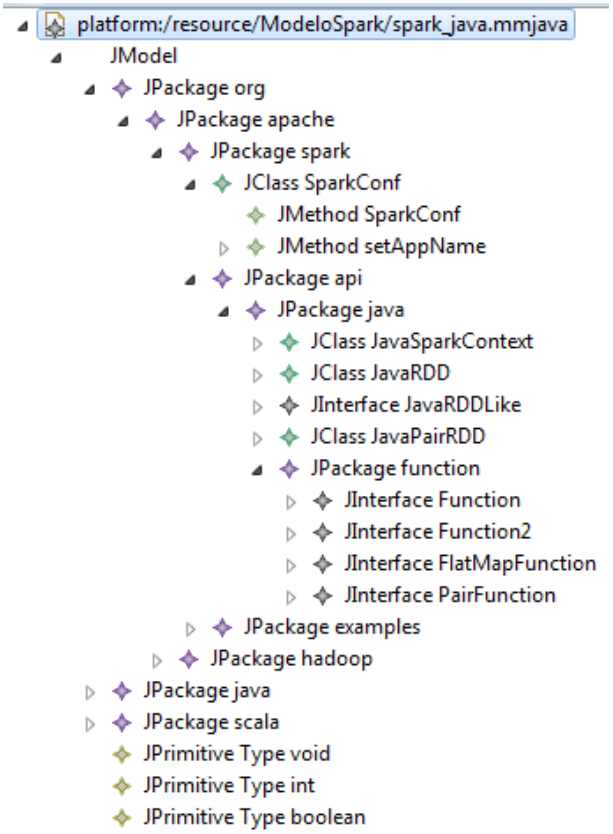


Figura 6.7: Fragmento do modelo da API *Spark* conforme metamodelo Java criado usando *APIMapReduceEditor*.

O usuário com o papel "Desenvolvedor" irá gerar o PSM concreto de forma automática através da execução da definição de transformação que terá como entrada o PSM abstrato e o modelo da API *Spark* para Java, conforme as tarefas 5 e 6 da Figura 4.12. O modelo da API *Spark* deve estar de acordo com o metamodelo utilizado para descrever o PSM concreto. Esse modelo é necessário nessa fase, pois ele irá fornecer as informações da especificação dos métodos usados pelo *framework Spark* que deverão constar no modelo do PSM concreto.

A Figura 6.7 representa um fragmento do modelo da API de *Spark*, conforme metamodelo da linguagem Java que foi construído usando o editor *APIMapReduceEditor*. Esse modelo é de responsabilidade do usuário com papel

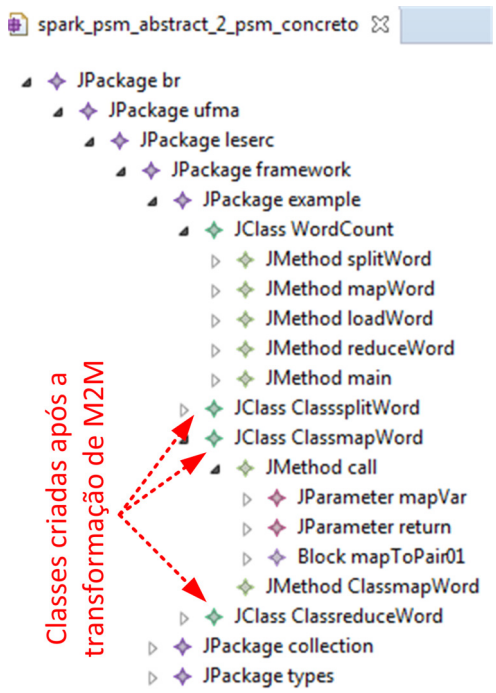


Figura 6.8: PSM concreto baseado em API *Spark* + Java e visualizado através do *PSMConcretoEditor*

"Analista de sistemas". Enquanto, a Figura 6.8 apresenta o PSM concreto gerado, que pode ser visualizado através do editor *PSMConcretoEditor*.

Tanto a manipulação dos modelos, como a execução das transformações podem ser realizadas pelos usuários através da *TF2BD*.

Detalhamento do PSM Concreto

Analisando a Figura 6.8 pode-se ver que outras classes, além da classe *WordCount*, aparecem no PSM concreto, como: *ClasssplitWord*, *ClassmapWord* e *ClassreduceWord*. Essas classes foram criadas após a execução da transformação de modelo para modelo do PSM abstrato para o PSM Concreto.

As novas classes foram geradas usando a seguinte regra: todo método do PIM, que foi relacionado a um elemento *Mapper* ou *Reducer* do PDM, nesta fase do processo, produz uma classe baseada no nome do método do PIM. Por exemplo, o método *splitWord* do PIM cria a classe *ClasssplitWord* no PSM concreto, da mesma forma que o método *mapWord* do PIM irá gerar a classe *ClassmapWord* no PSM concreto, etc.

Código 6.1: Novo comportamento atribuído ao método *splitWord* no PSM concreto

```
1 public JavaRDD<String> splitWord(JavaRDD<String> rdd) {
2   JavaRDD<String> result;
3   ClasssplitWord classsplitWord;
4   classsplitWord =  new ClasssplitWord();
5   result =  rdd.flatMap(classsplitWord);
6   return result;
7 }
```

O comportamento original do método do PIM também muda no PSM concreto. O novo comportamento no PIM instancia a nova classe que foi criada com base no nome do método do PIM. Na nova classe criada, o comportamento do método é o mesmo comportamento que foi definido no modelo de *weaving*. Na Listagem do Código 6.1, pode-se visualizar o novo comportamento atribuído para o método *splitWord* no PSM concreto. Por sua vez, na Listagem do Código 6.2 pode-se ver que o comportamento do método "*call*" da classe *ClasssplitWord* criada assume o comportamento definido no modelo de *weaving*. Essa adequação do comportamento é realizada para que as regras de negócio do PIM possam ser adequadas para a realidade exigida pelo *framework Spark*.

6.1.6 Geração do Código Fonte

Na tarefa 7 da Figura 4.12, uma transformação de modelo para texto é executada. O objetivo dessa transformação é gerar o código fonte de maneira textual para que possa ser compilado.

A geração do código fonte é feita de maneira automática pelo usuário com o papel "Desenvolvedor", através da execução da definição de transformação que terá como entrada o PSM concreto, e gerará como artefato de saída o código fonte. A Listagem do Código 6.2 mostra o código fonte gerado da classe *ClasssplitWord*. Essa transformação também poderá ser realizada utilizando *TF2BD*.

Posteriormente, na tarefa 8 da Figura 4.12, o código fonte é executado de maneira manual para verificar se existem inconsistências entre o código gerado e o que foi modelado. Caso haja inconsistência no código fonte ou a lógica esteja errada, o processo retorna para a tarefa 1 para adequações nos modelos de entrada, e posterior execução de todas as transformações para gerar uma nova versão do código fonte. A ideia é que as correções sejam sempre feitas nos modelos de entrada e que o processo seja executado novamente até a geração do novo código corrigido.

Código 6.2: Exemplo de código fonte gerado.

```
1  package br.ufma.leserc.framework.example;
2
3  import org.apache.spark.api.java.function.FlatMapFunction;
4  import br.ufma.leserc.framework.types.Integer;
5  import br.ufma.leserc.framework.collection.List;
6  import br.ufma.leserc.framework.types.String;
7
8  public class ClasssplitWord implements FlatMapFunction<String,String>  {
9
10  public Iterable<String> call(String word) {
11   Integer index;
12   List lista;
13   lista =  new List();
14   String letter;
15   String sbStr;
16   for (index = 1; index < sbStr.size(); index = index + 1){
17    letter =  word.subString(index,index);
18    if (letter != " "){
19     sbStr = sbStr + letter;
20    } else {
21     lista.add(sbStr);
22     sbStr = "";
23    }
24   }
25   return  lista.iterable();
26  }
27
28  public  ClasssplitWord() {
29
30  }
31  }
```

6.2 Exemplo 2 - Chamada de Emergência

O segundo exemplo, que será apresentado, tem um grau de complexidade maior do que o primeiro. A complexidade mencionada, em questão, faz referência ao quantitativo de classes, métodos e regras de negócio que devem ser incorporadas aos modelos. Assim, o exemplo que será apresentado, nessa seção, será o "Chamadas de Emergência" [53]. Esse exemplo foi adaptado de um blog que traz uma série de exemplos de possíveis aplicações *Big Data* [1]. A Tabela 6.2 apresenta uma comparação entre os dois exemplos ilustrativos, em termos de quantidade de classes, atributos, métodos e entrelaçamentos. O entrelaçamento, mencionado na tabela, faz referência a quantidade de métodos do PIM que vão assumir um comportamento que será descrito no modelo de *weaving*.

Tabela 6.2: Comparação entre os exemplos ilustrativos, em termos de classes, atributos, métodos e entrelaçamento, entre os exemplos 1 e 2.

Exemplo	Qtd. Classe	Qtd. Atributos	Qtd. Métodos	Qtd. Entrelaçamentos
Exemplo 1	1	0	5	4
Exemplo 2	9	14	62	17

Em muitos países, existe uma rede telefônica com um único número de telefone que permite acionar a assistência de serviços de emergência. Geralmente, esse número é de apenas três dígitos para facilitar a memorização e agilizar o processo de chamada. Por exemplo, nos Estados Unidos o número é o 911. Mas em alguns países, esse número de emergência é descentralizado pelo tipo de serviço de emergência oferecido. Por exemplo, no Brasil, existem os números 190 para polícia militar, 193 para bombeiros, 192 para serviço de remoção de doentes (ambulância) entre muitos outros serviços.

O problema "Chamada de Emergência" analisará duas fontes de dados semi-estruturados com informações de chamadas e das localidades na qual as chamadas ocorreram. As fontes de dados oferecidas contêm dados fictícios e é baseada no serviço de chamada dos Estados Unidos. A fonte de dados de chamadas de emergência está estruturada com os seguintes dados: Latitude, Longitude, Descrição da chamada de emergência, *Zip Code* (CEP), Título, Dia e Hora da ocorrência, Distrito e Endereço. Enquanto a fonte de localidade está estruturada com os seguintes dados: *Zip Code* (CEP), Cidade, Estado, Latitude, Longitude e Distrito.

Imagine que a fonte de dados de chamadas pode ser relativamente volumosa devido ao quantitativo de chamadas por hora, assim, a análise visa encontrar uma solução computacional que consiga recuperar as seguintes estatísticas:

- Que tipo de problemas são prevalentes, e em que estado?
- Que tipo de problemas são prevalentes, e em qual cidade?
- Que tipo de problemas são prevalentes, e em que estado e em que mês?

Uma discussão detalhada sobre a aplicação da TF2BD foi apresentada no exemplo 1. Assim, no exemplo 2, será focado apenas os detalhes que são pertinentes a essa solução. Por exemplo, não será mais feito referência a qual editor, papel ou artefato que foi produzido, pois todos esses assuntos foram detalhados cuidadosamente no exemplo anterior e procedem da mesma forma para esse exemplo.

6.2.1 Criação do PIM

As classes que compõe esse PIM foram estruturadas em classes de domínio, negócio, projeto e persistência. Classes de domínio são aquelas que são representações direta do domínio do problema; as classes de negócio são as classes que vão implementar as principais lógicas do negócio; as classes de projeto são aquelas criadas para auxiliar na solução computacional; as classes de persistência são aquelas que irão saber como acessar os dados. A Tabela 6.3 lista todas as classes com seu tipo e uma breve descrição.

Tabela 6.3: Classes do PIM Chamada de Emergência classificadas em: Domínio, Negócio, Projeto e Persistência.

Classe	Tipo	Descrição
Emergency	Domínio	Classe que representa a entidade Chamada de Emergência
ZipCode	Domínio	Classe que representa a entidade Localidade
AnalysisState	Negócio	Contém a regra para recuperar a quantidade de ocorrências por estado
AnalysisCity	Negócio	Contém a regra para recuperar a quantidade de ocorrências por cidade
AnalysisStateMonth	Negócio	Contém a regra para recuperar a quantidade de ocorrências por estado e por mês
EmergencyDAO	Persistência	Recupera as informações da fonte de dado Chamada de Emergência
ZipCodeDAO	Persistência	Recupera as informações da fonte de dado Localidade
EmergencyDataAnalysis	Projeto	Classe criada para interfacear o acesso da interface do usuário com as classes de negócio
Run	Projeto	Classe que representa a interface de usuário, além de ser a responsável por iniciar o processo

A Figura 6.9 apresenta o PIM criado para o exemplo "Chamada de Emergência", usando o editor *PIMEditor*. Nessa figura, é possível observar as nove classes criadas, além do detalhamento do comportamento do método "*numberCallByState*" em destaque, que pertence a classe *AnalysisState*.

É importante também que os métodos do PIM sejam construídos conforme o mapeamento BBI, para que a criação do modelo de *weaving* seja possível. A Tabela 6.4 apresenta a classificação de apenas alguns dos métodos do PIM usando o mapeamento BBI.

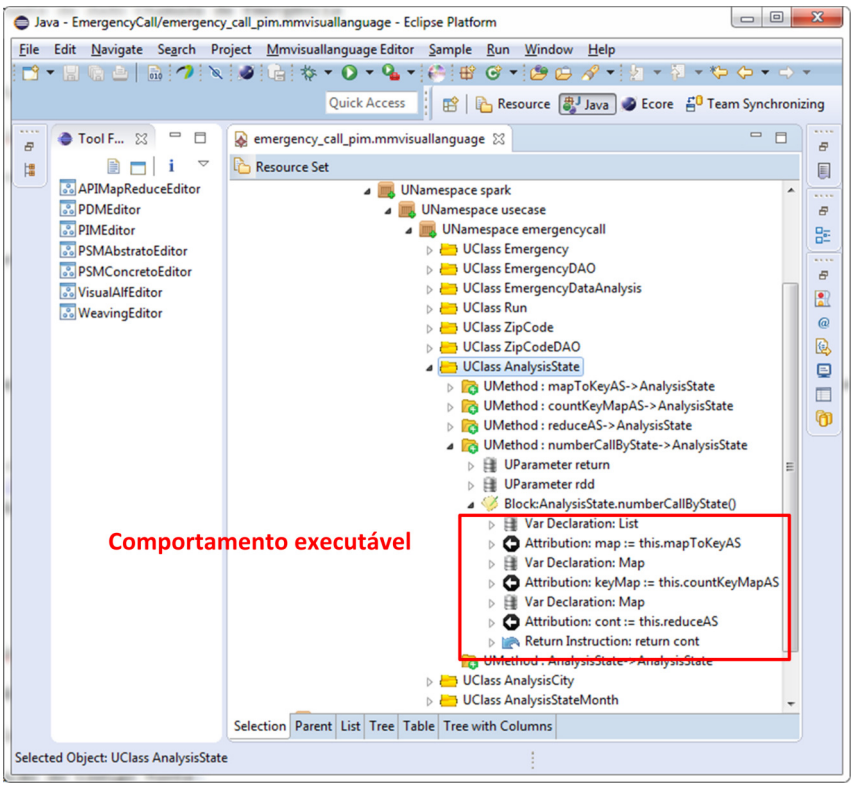


Figura 6.9: PIM para Chamada de Emergência criado usando o editor *PIMEditor*.

Tabela 6.4: Alguns métodos do PIM Chamada de Emergência classificados usando o mapeamento BBI.

Classe	Método	Classificação BBI
AnalysisCity	mapToKeyAC	Map
AnalysisCity	countKeyMapAC	Map
AnalysisCity	reduceAC	Tradicional
EmergencyDataAnalysis	executeJoin	Tradicional
EmergencyDataAnalysis	modifyData	Map
EmergencyDAO	loadFileEmergency	Tradicional
EmergencyDAO	mapEmergency	Map

6.2.2 Criação do PDM

A criação do PDM é a segunda tarefa do processo proposto, mas como ele está criado e não necessita de adequações para a solução desse problema, ele será reutilizado.

6.2.3 Criação do Weaving

A criação do modelo de *weaving* permite que o comportamento de *MapReduce* possa ser atribuído a métodos do PIM sem alterá-los. Na Figura 6.10, é apresentado o modelo de *weaving* criado pelo usuário de forma manual, usando

o editor *WeavingEditor*. Nessa figura, é feito o destaque do entrelaçamento entre o método *countKeyMapAC* do PIM, seta que parte da esquerda ao centro, com o comportamento de *Map*, seta que parte da direita para o centro. Pode-se observar também que o usuário realizou a definição de um novo comportamento para o PIM nesse entrelaçamento no modelo de *weaving*.

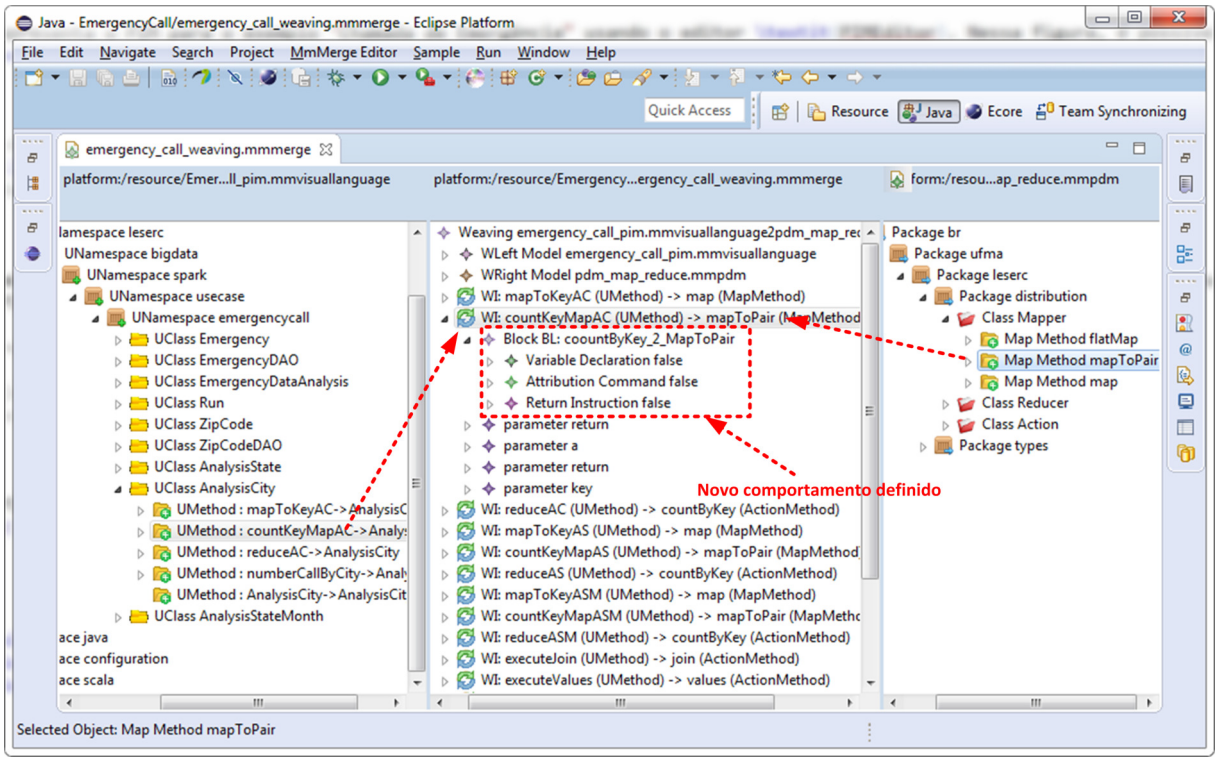


Figura 6.10: Modelo de *Weaving* para Chamada de Emergência, criado usando o editor *WeavingEditor*.

6.2.4 Geração do PSM Abstrato

O PSM abstrato é gerado pelo usuário automaticamente, assim como, descrito na Seção 6.1.4 do exemplo 1. Na Figura 6.11, pode-se observar o PSM abstrato gerado, usando o editor *PSMAbstratoEditor*. Nessa figura, ainda pode-se ver que o comportamento definido no modelo de *weaving* para o método *countKeyMapAC*, destacado na Figura 6.10, foi incorporado no PSM abstrato.

Todas as definições de transformações usadas no exemplo 1 serão as mesmas usadas no exemplo 2.

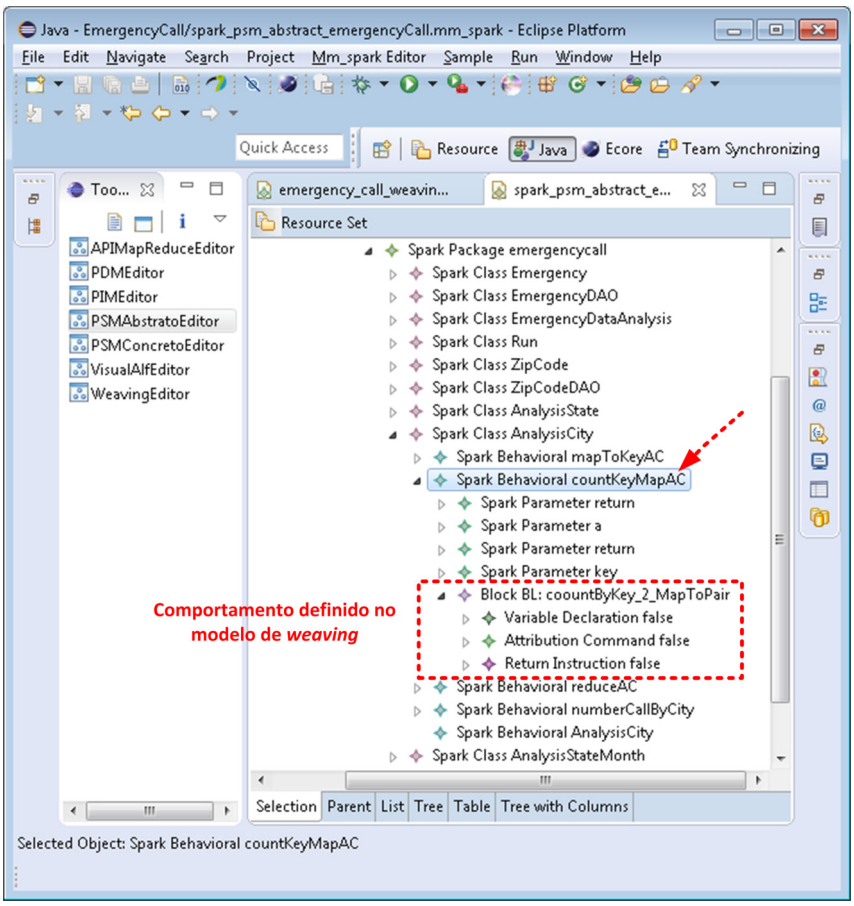


Figura 6.11: Modelo do PSM Abstrato gerado pela definição de transformação e visualizado usando o editor *PSMAbstratoEditor*.

6.2.5 Geração do PSM Concreto

O PSM concreto também é gerado pelo usuário automaticamente, assim como, descrito na Seção 6.1.5 do exemplo 1. O modelo da API de *MapReduce*, usado como uma das entradas para o motor de transformação, é o mesmo utilizado pelo exemplo 1. Na Figura 6.12, pode-se ver o PSM concreto gerado após a execução da definição de transformação. Nessa figura, também pode-se observar que foram geradas novas classes que estão destacadas. Essas classes foram geradas usando a mesma regra definida no detalhamento do PSM concreto que foi descrito na Seção 6.1.5.

Por exemplo, as classes geradas *ClasscountKeyMapAC* e *ClassmapToKeyAC*, destacadas na Figura 6.12, foram geradas como resultado do entrelaçamento dos métodos *countKeyMapAC* e *maptoKeyAC* da classe *AnalysisCity* do PIM com comportamentos de *Map* do PDM. O comportamento definido no método *call* de ambas as classes é o comportamento que foi definido no modelo de *weaving* e o

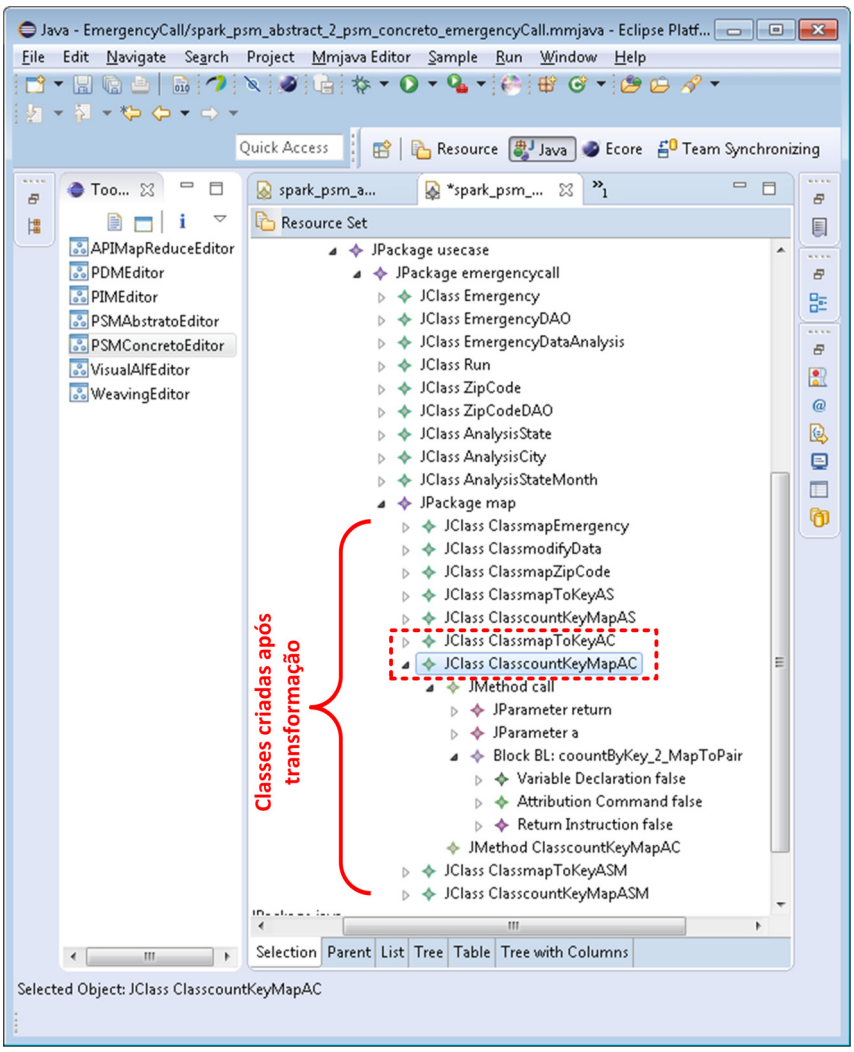


Figura 6.12: Modelo do PSM concreto gerado pela definição de transformação e visualizado, usando o editor *PSMConcretoEditor*.

comportamento definido no método original passa ser a chamada de um método da API *Spark*, passando como parâmetro uma instância dessas classes. As listagens com o código fonte desse exemplo será apresentada na próxima seção.

6.2.6 Geração do Código fonte

O código fonte é gerado através da execução de uma definição de transformação de modelo para texto. O PSM concreto, gerado na tarefa anterior, será agora transformado em arquivos textos para possibilitar que ocorra a compilação desses arquivos.

Por exemplo, a Listagem do Código 6.4 mostra o código fonte gerado para a classe *ClasscountKeyMapAC* usada como exemplo na Seção 6.2.5. A Listagem do

Código 6.5 mostra o código fonte gerado para a classe *ClassmapToKeyAC*. A Listagem do Código 6.3 apresenta o código fonte da classe *AnalysisCity* e pode-se observar que o comportamento dos métodos *countKeyMapAC*, linhas de 39 a 45, e *maptoKeyAC*, linhas de 31 a 37, foram adequados para realizar a chamada dos métodos *Map* da *API de Spark*, passando como parâmetro as suas respectivas classes que podem ser observadas respectivamente nas linhas 43 e 35.

Código 6.3: Código fonte gerado para classe *AnalysisCity*.

```
1  package example.br.ufma.leserc.bigdata.spark.usecase.emergencycall;
2
3  import br.ufma.leserc.bigdata.spark.usecase.emergencycall.map.ClasscountKeyMapAC;
4  import br.ufma.leserc.bigdata.spark.usecase.emergencycall.map.ClassmapToKeyAC;
5  import org.apache.spark.api.java.JavaRDD;
6  import org.apache.spark.api.java.JavaPairRDD;
7  import java.util.Map;
8
9  public class AnalysisCity {
10
11
12   public  Map<String,Object>  reduceAC(JavaPairRDD<String,Object> red) {
13     Map<String,Object>  cont;
14     cont =  red.countByKey();
15     return cont;
16   }
17
18   public  Map<String,Object>  numberCallByCity(JavaRDD<Emergency> rdd) {
19     JavaRDD<String>  map;
20     map =  this.mapToKeyAC(rdd);
21     JavaPairRDD<String,Integer>  keyMap;
22     keyMap =  this.countKeyMapAC(map);
23     Map<String,Object>  cont;
24     cont =  this.reduceAC(keyMap);
25     return cont;
26   }
27
28   public  void AnalysisCity() {
29   }
30
31   public  JavaRDD<String>  mapToKeyAC(JavaRDD<Emergency> rdd) {
32     JavaRDD<String>  result;
33     ClassmapToKeyAC classmapToKeyAC;
34     classmapToKeyAC =  new ClassmapToKeyAC();
35     result =  rdd.map(classmapToKeyAC);
36     return result;
37   }
38
39   public  JavaPairRDD<String,Integer>  countKeyMapAC(JavaRDD<String> rdd) {
40     JavaPairRDD<String,Integer>  result;
41     ClasscountKeyMapAC classcountKeyMapAC;
42     classcountKeyMapAC =  new ClasscountKeyMapAC();
43     result =  rdd.mapToPair(classcountKeyMapAC);
44     return result;
45   }
46
47   }
```

Código 6.4: Código fonte gerado em função do método countKeyMapAC da classe *AnalysisCity*.

```
1  package example.br.ufma.leserc.bigdata.spark.usecase.emergencycall.map;
2
3  import org.apache.spark.api.java.function.PairFunction;
4  import scala.Tuple2;
```

```
5
6  public class ClasscountKeyMapAC implements PairFunction<String,String,Integer>  {
7
8   public Tuple2<String,Integer>  call(String a) {
9     Tuple2<String,Integer>  tuple;
10    tuple =  new Tuple2(a,1);
11    return tuple;
12  }
13
14  public  ClasscountKeyMapAC() {
15  }
16  }
```

Código 6.5: Código fonte gerado em função do método mapToKeyAC da classe *AnalysisCity*.

```
1  package example.br.ufma.leserc.bigdata.spark.usecase.emergencycall.map;
2
3  import org.apache.spark.api.java.function.Function;
4  import java.lang.String;
5  import example.br.ufma.leserc.bigdata.spark.usecase.emergencycall.ZipCode;
6  import java.lang.Integer;
7
8  public class ClassmapToKeyAC implements Function<Emergency,String>  {
9
10  public  String call(Emergency emergency) {
11    String title;
12    title =  emergency.getTitle();
13    String subStr;
14    ZipCode zipC;
15    String city;
16    Integer index;
17    index =  title.indexOf(":");
18    subStr =  title.subString(0,index);
19    zipC =  emergency.getZipCode();
20    city =  zipC.getCity();
21    return subStr + "-->" + city;
22  }
23
24  public  ClassmapToKeyAC() {
25  }
26  }
```

A última tarefa do processo é executar o código fonte gerado para verificação de inconsistências. Caso haja alguma inconsistência, o processo começa novamente, através da análise e correção dos modelos e posterior execução das definições de transformações para obter a nova versão do código fonte.

6.3 Síntese

Nesse capítulo, dois exemplos ilustrativos foram apresentados para demonstrar que a ferramenta proposta, i.e. *TF2BD*, suporta as atividades de desenvolvimento de *software* para *Big Data*. Sendo assim, demonstra-se que o *framework* usado para implementar a *TF2BD* é viável. Essa ferramenta foi construída para auxilar na construção de *software* para *Big Data* que usem o *framework Spark*.

O capítulo começa apresentando o primeiro exemplo ilustrativo escolhido, contar palavras, que foi bem simples com apenas uma classe, mas foi utilizado para apresentar, com uma riqueza de detalhes, as principais características oferecidas pela *TF2BD*. Em seguida, o capítulo é encerrado com a apresentação do segundo exemplo ilustrativo escolhido que foi o chamadas de emergência. Esse é um problema real, mais útil e bem mais complexo quando comparado ao primeiro. Possui um total de nove classes, mas foi apresentado de forma mais simplificada quando se refere a utilização da ferramenta, para que o texto não ficasse repetitivo e massante.

7 Experimento para avaliar TF2BD

A experimentação é a forma mais confiável de verificar teorias a fim de que elas possam ser comprovadas e corrigidas [41], além de ser um processo sistemático que envolve algumas etapas como observação do problema, formulação das questões de pesquisa, validação de hipóteses, execução do experimento, análises estatísticas para estabelecer relações e, finalmente, formular conclusões [71].

Trabalhos que oferecem soluções para suportar desenvolvimento, baseado em *MapReduce* de *Big Data*, são encontrados na literatura, alguns foram apresentados no Capítulo 3, mas nenhum realizou um estudo experimental, com base em hipóteses comprovadas estatisticamente, que analisasse suas teorias. Desta forma, este capítulo apresenta um estudo experimental para analisar o suporte às atividades de desenvolvimento utilizando a TF2BD, apresentada no Capítulo 5, e construída usando o *framework* F2BD, apresentado no Capítulo 4. O experimento foi baseado no trabalho de Oliveira et al. [71], pois seu experimento apresenta semelhanças no contexto (desenvolvimento de *software*) e prioriza o controle da execução, além da medição.

Todas as seções deste capítulo, se concentram em descrever o estudo experimental, com exceção da Seção 7.13, que faz uma análise comparativa deste trabalho com os trabalhos relacionados.

7.1 Definição do Objetivo

O objetivo deste experimento é analisar o suporte às atividades de desenvolvimento de *software* para *Big Data* com ou sem o uso da TF2BD. Sendo assim, será avaliado o efeito desses dois suportes com respeito ao esforço, à eficácia e à percepção da qualidade nas tarefas de desenvolvimento, manutenção e evolução. Para tanto, o experimento contará com a participação de analistas e técnicos em tecnologia da informação, estudantes de graduação, estudantes de pós-graduação em Ciência da

Computação. Também, será analisada, a percepção sobre o esforço necessário para manter a documentação e o código fonte sincronizados.

7.2 Questões e Métricas

O estudo foi planejado com o objetivo de responder as questões relacionadas ao esforço, à eficácia e à percepção de qualidade, assim como no trabalho de Oliveira et al. [71].

7.2.1 Questões de Pesquisa Relacionadas ao Esforço

As Questões de Pesquisa (QP) relacionadas ao esforço são as seguintes:

- QP1: O esforço necessário para realizar a tarefa de desenvolvimento de um *software Big Data* com a utilização da TF2BD é menor que o esforço exigido para desenvolver o mesmo *software Big Data* sem a utilização da TF2BD?
- QP2: O esforço necessário para realizar a tarefa de manutenção de um *software Big Data* com a utilização da TF2BD é menor que o esforço exigido para manter o mesmo *software Big Data* sem a utilização da TF2BD?
- QP3: O esforço necessário para realizar a tarefa de evolução de um *software Big Data* com a utilização da TF2BD é menor que o esforço exigido para evoluir o mesmo *software Big Data* sem a utilização da TF2BD?
- QP4: O esforço necessário para manter a documentação de um *software Big Data* sincronizada com o código fonte, gerado com a utilização da TF2BD, é menor que o esforço exigido para manter sincronizada a documentação do mesmo *software Big Data* sem a utilização da TF2BD?

Utilizou-se a medida de tempo em minutos como métrica para avaliar o esforço aplicado no desenvolvimento, manutenção e evolução de *software Big Data*. Quanto maior o tempo, maior o esforço aplicado. Enquanto, para aferir a precisão da sincronização da documentação foi utilizada uma escala de valor que vai de 1 a 5, sendo que 1 é o menor esforço empregado e 5 é o maior esforço empregado.

7.2.2 Questões de Pesquisa Relacionadas à Eficácia

As questões de pesquisa relacionadas a eficácia são as seguintes:

- QP5: A eficácia no suporte à tarefa de desenvolvimento de *software Big Data* implementados, utilizando TF2BD, é maior que a eficácia no suporte ao desenvolvimento de *software Big Data* implementados sem a utilização da TF2BD?
- QP6: A eficácia no suporte à tarefa de manutenção de *software Big Data* mantidos, utilizando TF2BD, é maior que a eficácia no suporte à manutenção de *software Big Data* mantidos sem a utilização da TF2BD?
- QP7: A eficácia no suporte à tarefa de evolução de *software Big Data*, utilizando TF2BD, é maior que a eficácia no suporte à evolução de *software Big Data* evoluídos sem a utilização da TF2BD?

Para avaliar a eficácia no suporte ao desenvolvimento, a manutenção e a evolução, foi utilizada a métrica inverso da quantidade de erros de programação $(1/(quantidade\_erros + 1))$, utilizada por Oliveira et al. [71].

7.2.3 Questões de Pesquisa Relacionadas à Percepção de Qualidade

As questões de pesquisa relacionadas à percepção de qualidade são as seguintes:

- QP8: A percepção da qualidade do suporte oferecido ao desenvolvimento de *software Big Data* implementados, utilizando TF2BD, é maior que a percepção de qualidade no suporte ao desenvolvimento de *software Big Data* implementados sem a utilização da TF2BD?
- QP9: A percepção da qualidade do suporte oferecido à manutenção de *software Big Data* mantidos, utilizando TF2BD, é maior que a percepção de qualidade no suporte à manutenção de *software Big Data* mantidos sem a utilização da TF2BD?
- QP10: A percepção da qualidade do suporte oferecido à evolução de *software Big Data* evoluídos, utilizando TF2BD, é maior que a percepção de qualidade no suporte à evolução de *software Big Data* evoluídos sem a utilização da TF2BD?

- QP11: A percepção da qualidade do suporte oferecido à documentação de *software Big Data* documentados, utilizando TF2BD, é maior que a percepção de qualidade no suporte à documentação de *software Big Data* documentados sem a utilização da TF2BD?

Considerando a subjetividade que a natureza dessa questão levanta, questionários para aferir a satisfação dos voluntários foram utilizados.

7.3 Hipóteses de Pesquisa

As hipóteses de pesquisa levantadas neste estudo experimental são relacionadas ao esforço, à eficácia e à percepção de qualidade, assim como no trabalho de Oliveira et al. [71].

7.3.1 Hipóteses Relacionadas ao Esforço

As hipóteses de pesquisa relacionadas ao esforço são as seguintes:

- Hipótese nula, H_0 *esforço desenvolvimento*: Não há diferença no esforço, medido em termos de tempo em minutos, para desenvolver *software Big Data* com e sem o uso da TF2BD. H_0 *esforço desenvolvimento*: Tempo para desenvolver (TF2BD) = Tempo para desenvolver (Sem suporte da TF2BD). Hipótese alternativa, H_1 *esforço desenvolvimento*: Tempo para desenvolver (TF2BD) $\neq$ Tempo para desenvolver (Sem suporte da TF2BD);

- Hipótese nula, H_0 *esforço manutenção*: Não há diferença no esforço, medido em termos de tempo em minutos, para manter *software Big Data* com e sem o uso da TF2BD. H_0 *esforço manutenção*: Tempo para manter (TF2BD) = Tempo para manter (Sem suporte da TF2BD). Hipótese alternativa, H_1 *esforço manutenção*: Tempo para manter (TF2BD) $\neq$ Tempo para manter (Sem suporte da TF2BD);

- Hipótese nula, H_0 *esforço evolução*: Não há diferença no esforço, medido em termos de tempo em minutos, para evoluir *software Big Data* com e sem o uso da TF2BD. H_0 *esforço evolução*: Tempo para evoluir (TF2BD) = Tempo para evoluir

(Sem suporte da TF2BD). Hipótese alternativa, H_1 *esforço evolução*: Tempo para evoluir (TF2BD) $\neq$ Tempo para evoluir (Sem suporte da TF2BD);

- Hipótese nula, H_0 *esforço documentação*: Não há diferença na percepção do esforço para manter a documentação sincronizada com o código fonte de *software Big Data* com e sem o uso da TF2BD. H_0 *esforço documentação*: Percepção do esforço (TF2BD) = Percepção do esforço (Sem suporte da TF2BD). Hipótese alternativa, H_1 *esforço documentação*: Percepção do esforço (TF2BD) $\neq$ Percepção do esforço (Sem suporte da TF2BD);

7.3.2 Hipóteses Relacionadas à Eficácia

As hipóteses de pesquisa relacionadas à eficácia são as seguintes:

- Hipótese nula, H_0 *eficácia desenvolvimento*: Não há diferença na eficácia para desenvolver *software Big Data* com e sem o uso da TF2BD. H_0 *eficácia desenvolvimento*: Eficácia para desenvolver (TF2BD) = Eficácia para desenvolver (Sem suporte da TF2BD). Hipótese alternativa, H_1 *eficácia desenvolvimento*: Eficácia para desenvolver (TF2BD) $\neq$ Eficácia para desenvolver (Sem suporte da TF2BD);

- Hipótese nula, H_0 *eficácia manutenção*: Não há diferença na eficácia para manter *software Big Data* com e sem o uso da TF2BD. H_0 *eficácia manutenção*: Eficácia para manter (TF2BD) = Eficácia para manter (Sem suporte da TF2BD). Hipótese alternativa, H_1 *eficácia manutenção*: Eficácia para manter (TF2BD) $\neq$ Eficácia para manter (Sem suporte da TF2BD);

- Hipótese nula, H_0 *eficácia evolução*: Não há diferença na eficácia para evoluir *software Big Data* com e sem o uso da TF2BD. H_0 *esforço evolução*: Eficácia para evoluir (TF2BD) = Eficácia para evoluir (Sem suporte da TF2BD). Hipótese alternativa, H_1 *eficácia evolução*: Eficácia para evoluir (TF2BD) $\neq$ Eficácia para evoluir (Sem suporte da TF2BD);

7.3.3 Hipóteses Relacionadas à Percepção de Qualidade

As hipóteses de pesquisa relacionadas à percepção de qualidade são as seguintes:

- Hipótese nula, H_0 *qualidade desenvolvimento*: Não há diferença na percepção da qualidade do suporte oferecido ao desenvolvimento de *software Big Data* com e sem o uso da TF2BD. H_0 *qualidade desenvolvimento*: Qualidade desenvolvimento (TF2BD) = Qualidade desenvolvimento (Sem suporte da TF2BD). Hipótese alternativa, H_1 *qualidade desenvolvimento*: Qualidade desenvolvimento (TF2BD) $\neq$ Qualidade desenvolvimento (Sem suporte da TF2BD);

- Hipótese nula, H_0 *qualidade manutenção*: Não há diferença na percepção da qualidade do suporte oferecido à manutenção de *software Big Data* com e sem o uso da TF2BD. H_0 *qualidade manutenção*: Qualidade manutenção (TF2BD) = Qualidade manutenção (Sem suporte da TF2BD). Hipótese alternativa, H_1 *qualidade manutenção*: Qualidade manutenção (TF2BD) $\neq$ Qualidade manutenção (Sem suporte da TF2BD);

- Hipótese nula, H_0 *qualidade evolução*: Não há diferença na percepção da qualidade do suporte oferecido à evolução de *software Big Data* com e sem o uso da TF2BD. H_0 *qualidade evolução*: Qualidade evolução (TF2BD) = Qualidade evolução (Sem suporte da TF2BD). Hipótese alternativa, H_1 *qualidade evolução*: Qualidade evolução (TF2BD) $\neq$ Qualidade evolução (Sem suporte da TF2BD);

- Hipótese nula, H_0 *qualidade documentação*: Não há diferença na percepção da qualidade do suporte oferecido à documentação de *software Big Data* com e sem o uso da TF2BD. H_0 *qualidade documentação*: Qualidade documentação (TF2BD) = Qualidade documentação (Sem suporte da TF2BD). Hipótese alternativa, H_1 *qualidade documentação*: Qualidade documentação (TF2BD) $\neq$ Qualidade documentação (Sem suporte da TF2BD);

7.4 Definição de Variáveis para o Experimento

As variáveis consideradas neste experimento são as seguintes:

- **Variáveis independentes**: São aquelas que são disponibilizadas como entrada para o experimento. Essas variáveis podem ser manipuladas durante o experimento. Nesse experimento, TF2BD e API de *Spark*, utilizando a IDE Eclipse para Java, ambas utilizadas para suportar o desenvolvimento de *software Big Data*, são consideradas as variáveis independentes, além de *software* como ArgoUML para auxiliar na documentação.

- **Variáveis dependentes**: São aquelas resultantes após a execução do experimento. Elas oferecem os resultados referentes aos tratamentos aplicados no experimento. As variáveis dependentes consideradas nesse experimento, para analisar o suporte às tarefas de desenvolvimento, manutenção, evolução e documentação, são a eficácia, o esforço e a percepção de qualidade dos participantes.

7.5 Objetos de Estudo

Os objetos de estudo são os meios utilizados para verificar a relação de causa-efeito nas hipóteses. Durante a execução do experimento, os tratamentos com e sem o uso da TF2BD são aplicados aos objetos. Nesse experimento, considerou-se como objeto o exemplo ilustrativo *WordCount*, apresentado na Seção 6.1. Esse objeto foi escolhido para que o experimento não se tornasse muito longo e dificultasse a adesão de participantes.

Por meio desse objeto, visa-se demonstrar o relacionamento causa-efeito entre o suporte às tarefas de desenvolvimento, de manutenção, de evolução e de documentação com e sem a utilização da TF2BD.

7.6 Seleção dos Participantes

A seleção dos participantes é uma fase importante para o experimento, porque os participantes escolhidos devem possibilitar uma generalização dos resultados obtidos. Assim como, no trabalho de Oliveira et al. [71], a escolha dos participantes não foi feita de forma aleatória, como se espera de um experimento, mas conforme a disponibilidade encontrada. Esse tipo de amostragem é conhecida como amostragem de conveniência.

Os participantes alvos deste experimento são analistas e técnicos com formação profissional em Ciência da Computação ou áreas afins ou que apenas tenham experiência em desenvolvimento de *software* com o paradigma de programação orientado a objeto. Foram considerados também alunos de graduação ou de pós-graduação de cursos em Ciência da Computação ou áreas afins que também tenham experiência em programação com orientação a objetos.

Um questionário foi aplicado para caracterização dos participantes. O propósito foi descobrir o perfil de cada um dos participantes em termos acadêmicos, profissionais e de conhecimento do modelo *MapReduce*. Esse questionário pode ser encontrado no Apêndice D.

No processo de seleção, 21 (vinte e um) questionários foram preenchidos, mas apenas 18 participantes concluíram todas as etapas do experimento. Destes, 61% são profissionais que atuam no mercado, 22% são apenas estudantes e 17% estudam e trabalham. Em termos de formação, o perfil dos participantes é equilibrado. O maior percentual encontrado é de mestres 45% (28% de mestres + 17% de doutorandos) contra 39% de graduados (22% de graduados + 17% de mestrandos). A Figura 7.1 apresenta os percentuais em termos de formação e tipo de experiência dos participantes.

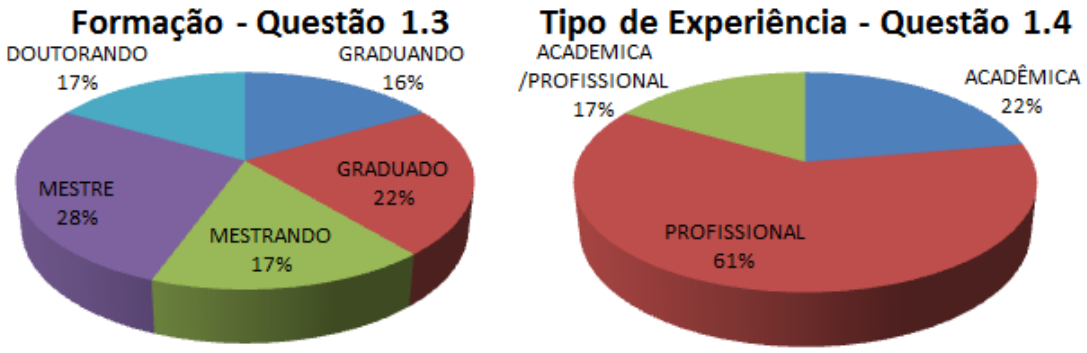


Figura 7.1: Perfil dos participantes em termo de formação e tipo de experiência.

A respeito da experiência com programação orientada a objetos, 61,11% dos participantes afirmaram ter experiência moderada, 33,33% experiência avançada e apenas 5,56% experiência básica. Assim como, quando questionados sobre a importância da documentação para o processo de desenvolvimento 88,89% afirmaram que a documentação é essencial, mas apenas 16,67% afirmaram que sempre que o código é atualizado também têm o cuidado de atualizar a documentação. A Figura 7.2 apresenta três gráficos com os percentuais relacionados à experiência com orientação

a objetos, à importância da documentação e à frequência que os participantes atualizam documentação de um *software*. O termo alternativa descrito na legenda dos gráficos corresponde às alternativas que os participantes poderiam selecionar nas suas respectivas questões, por exemplo a alternativa 1 sobre experiência em orientação a objetos corresponde à primeira alternativa da questão 2.1 do formulário e assim sucessivamente.

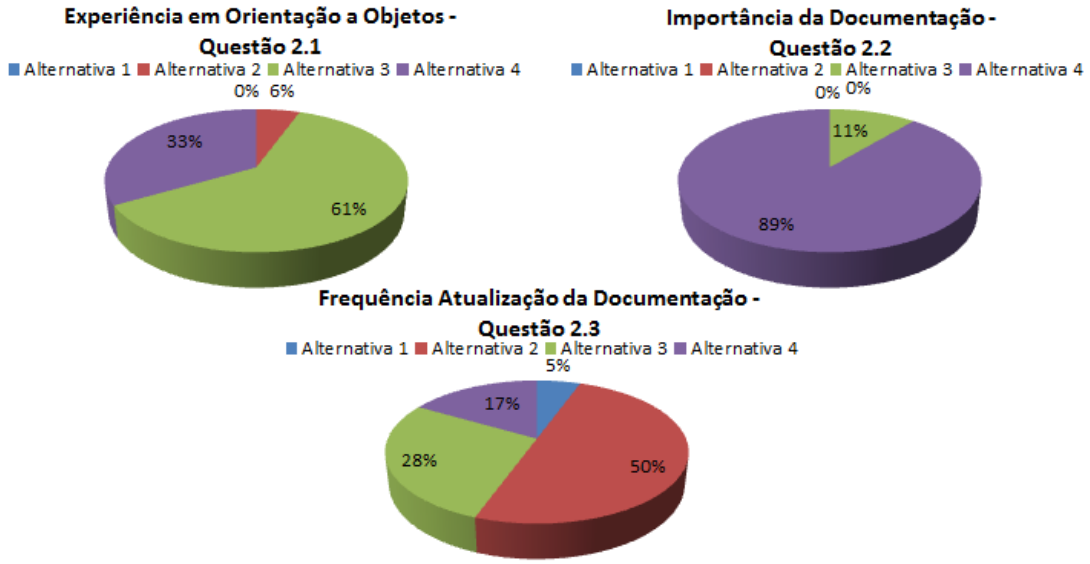


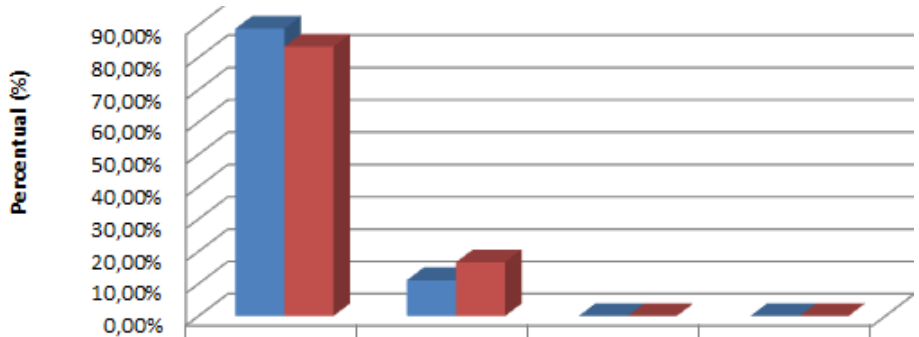
Figura 7.2: Perfil dos participantes em termo de experiência em orientação a objetos, importância e frequência da atualização da documentação.

Em termos de conhecimento, mais de 80% dos participantes afirmaram desconhecer o modelo *MapReduce* e soluções aplicadas para o desenvolvimento, utilizando-o. A Figura 7.3 apresenta um gráfico com o nível de conhecimento associado para o modelo *MapReduce*, assim como soluções que suportem o desenvolvimento, utilizando-o. Esse desconhecimento faz com que um treinamento deva ser considerado antes da execução do experimento.

7.7 Contexto e Instrumentação

O contexto do experimento é *in-vitro*, com participantes com perfis mistos entre estudantes e profissionais para resolver problemas do mundo real, mas com dados fictícios de forma específica para *MapReduce* de *Big Data*.

A instrumentação do experimento foi composta de materiais que foram disponibilizados para os participantes de forma impressa, como o roteiro para



	Desconheço	Básico	Moderado	Avançado
Conhecimento sobre MapReduce	88,89%	11%	0%	0%
Conhecimento de ferramentas/frameworks para MapReduce	83,33%	16,67%	0,00%	0,00%

Figura 7.3: Perfil dos participantes em termo de conhecimento do modelo *MapReduce* e de soluções que suportem o desenvolvimento de *software* utilizando-as.

construção do objeto proposto no experimento e os mecanismos utilizados para registrar os resultados das medições. Por exemplo, para mensurar o tempo foram utilizados formulários que foram preenchidos pelos participantes a cada tarefa concluída.

7.8 Projeto do Experimento

O projeto do experimento demonstra como o experimento deve ser conduzido, além de determinar como os tratamentos são aplicados aos participantes utilizando os objetos [41].

Neste estudo, a estratégia experimental apresentada por Oliveira et al. [71] foi aplicada. Esta estratégia utiliza um fator com dois tratamentos sem *crossover*. Oliveira et al. [71] aplica essa estratégia com o intuito de reduzir algumas ameaças relacionadas a validade interna. Para Amaral et al. [41], à validade interna é a que possui maior relevância dentro do contexto de experimentos para engenharia de *software*.

O experimento realizado neste trabalho considera mais de um fator. Os fatores considerados são os seguintes: suporte à tarefa de desenvolvimento, de manutenção e de evolução de *software MapReduce*. Para utilização da estratégia experimental, aplicada por Oliveira et al. [71], cada um dos fatores considerados neste

experimento foi analisado separadamente, ou seja, para cada fator foi aplicado os tratamentos com e sem o suporte da TF2BD sem *crossover*.

A estratégia de experimentação usada por Oliveira et al. [71] foi aplicada neste experimento pela semelhança apresentada entre o contexto (desenvolvimento de *software*) e por priorizar o controle da execução e medição. A Tabela 7.1 apresenta um resumo do projeto do experimento de Oliveira et al. [71] adequado à realidade deste trabalho.

Tabela 7.1: Estratégia de experimentação aplicada neste experimento.

	Suporte ao Desenvolvimento, a Manutenção, a Evolução e a Documentação de Software MapReduce de BigData			
	1 Etapa		**2 Etapa**	
	1 Sessão	**2 Sessão**	**3 Sessão**	**4 Sessão**
Grupos	Abordagem com TF2BD		Abordagem Manual	
Grupo Único	Treinamento (2h)	Aplicação do Objeto	Treinamento (2h)	Aplicação do Objeto

O objeto foi aplicado aos participantes em duas etapas. Na primeira etapa, os participantes desenvolveram, mantiveram e evoluíram o objeto através do tratamento com TF2BD. Em seguida, na segunda etapa fizeram o mesmo, mas com o tratamento sem TF2BD (manualmente). Os fatores foram aplicados individualmente e na seguinte ordem: desenvolver, manter e evoluir. A documentação foi analisada durante a execução desses fatores.

Um treinamento com os tratamentos foi realizado antes de cada etapa. Esse treinamento foi de até duas horas e simulou o que os participantes deveriam executar durante o experimento.

7.9 Operação

Foi realizada, antes da execução do experimento, uma apresentação esclarecendo sobre o contexto que levou a sua realização, assim como, uma visão geral de como esse experimento ia ser conduzido e esclarecimentos sobre algumas questões que poderiam influenciar nos resultados alcançados. Durante essa apresentação, tomou-se o cuidado para que as hipóteses da pesquisa não fossem reveladas.

A primeira etapa do experimento consiste em duas sessões. Na primeira sessão, um treinamento sobre o que é *Big Data* foi realizado e como funciona o desenvolvimento para *Big Data*, usando *MapReduce* e o suporte para *MapReduce* de *Big Data*, usando a TF2BD. Durante o treinamento, o pesquisador pôde interagir com os participantes para esclarecer dúvidas e auxiliar nas tarefas a serem executadas. A ideia do treinamento é permitir que os participantes executem as mesmas tarefas do experimento, mas com um objeto diferente do que foi aplicado no estudo.

Na segunda sessão, da primeira etapa do experimento, os participantes executaram o experimento, utilizando o objeto proposto na Seção 7.5. Nessa sessão do experimento, os participantes não puderam interagir com os pesquisadores. Todo o ambiente da execução foi preparado antecipadamente, de tal forma que os participantes se preocupassem apenas em utilizá-lo. As tarefas que os participantes executaram foram: desenvolver[1], manter[2], evoluir[3] e finalmente documentar[4] o objeto.

A segunda etapa do experimento também é composta de duas sessões. Na primeira sessão, da segunda etapa, um treinamento foi realizado utilizando os instrumentos descritos na Seção 7.7. O treinamento realizado na segunda etapa utilizou o mesmo objeto e as mesmas tarefas aplicadas no treinamento realizado na primeira etapa, diferenciando apenas no tratamento utilizado. Durante esse treinamento, os participantes puderam interagir com os pesquisadores.

Na segunda sessão, da segunda etapa, a execução do experimento ocorreu sem a utilização da TF2BD, ou seja, apenas com o uso das soluções apresentadas durante o treinamento da segunda etapa. Semelhante à execução da primeira etapa, os participantes não puderam interagir com os pesquisadores e executaram as mesmas tarefas aplicadas durante a segunda sessão da primeira etapa.

Após a execução das duas etapas, os participantes foram convidados a responderem o questionário com questões referentes ao experimento e sobre a qualidade do suporte oferecido pela TF2BD.

[1]por desenvolver se entende por construir a solução. A especificação das classes e dos métodos foram passadas como roteiro para os participantes.

[2]por manter se entende por fazer qualquer alteração no código original. A manutenção exigida foi modificar o nome de todas as classes e métodos de inglês para português.

[3]por evoluir subtende-se por adicionar novas funcionalidades ou mudar o comportamento original do *software*. A evolução exigida foi a modificação do comportamento de um método.

[4]por documentar subtende-se sincronizar o código fonte com os modelos criados. Para a primeira etapa, a tarefa de documentação é transparente, enquanto para a segunda etapa essa tarefa será feita com auxílio da ferramenta ArgoUML.

O experimento deveria contar com 21 participantes, mas três acabaram sendo excluídos. Os três excluídos eram profissionais, sendo que dois tiveram que abandonar o experimento por questões relativas a trabalho e um não compareceu à primeira etapa do experimento. O experimento foi executado em três momentos para que se pudesse alcançar o número de 18 participantes. Essa divisão ocorreu por questões de agenda dos participantes e por questões de limitação do ambiente para execução. Todos os momentos foram executados conforme o projeto experimental descrito na Seção 7.8 e detalhado nesta seção. Dessa forma, o grupo único mencionado no projeto experimental representa como a amostra foi analisada.

7.10 Resultados do Experimento

Os resultados do experimento estão estruturados em três seções. A Seção 7.10.1 apresenta os resultados relacionados ao esforço obtido em cada etapa do experimento, enquanto, a Seção 7.10.2 apresenta os resultados relacionados à eficácia em função dos erros obtidos pelo participantes durante as etapas do experimento e, finalmente, a Seção 7.10.3 apresenta os resultados relacionados à percepção da qualidade. Os resultados são apresentados em forma de tabelas ou em forma de gráficos com a intenção de possibilitar uma melhor compreensão para o leitor.

7.10.1 Resultados Relacionados ao Esforço

A Tabela 7.2 apresenta os resultados relacionados ao esforço com o uso de dois tratamentos: com suporte da TF2BD (1ª Etapa) e sem o suporte da TF2BD (2ª Etapa) durante o desenvolvimento, a manutenção e a evolução do objeto proposto (*WordCount*). O esforço é apresentado em minutos e detalhado por participante e pela aplicação dos dois tratamentos para cada fator. Por exemplo, o participante 1 levou 24 minutos para desenvolver, 1 minuto para manter e 9 minutos para evoluir com o suporte da TF2BD, enquanto o mesmo participante sem o suporte da TF2BD levou 55 minutos para desenvolver, 4 minutos para manter e 6 minutos para evoluir.

Foram encontrados 1 *outlier*[5] na segunda etapa do desenvolvimento, 2 *outliers* na manutenção, sendo um na primeira etapa e o outro na segunda e mais dois

[5]*outlier* são valores atípicos e que apresentam grande afastamento das demais observações.

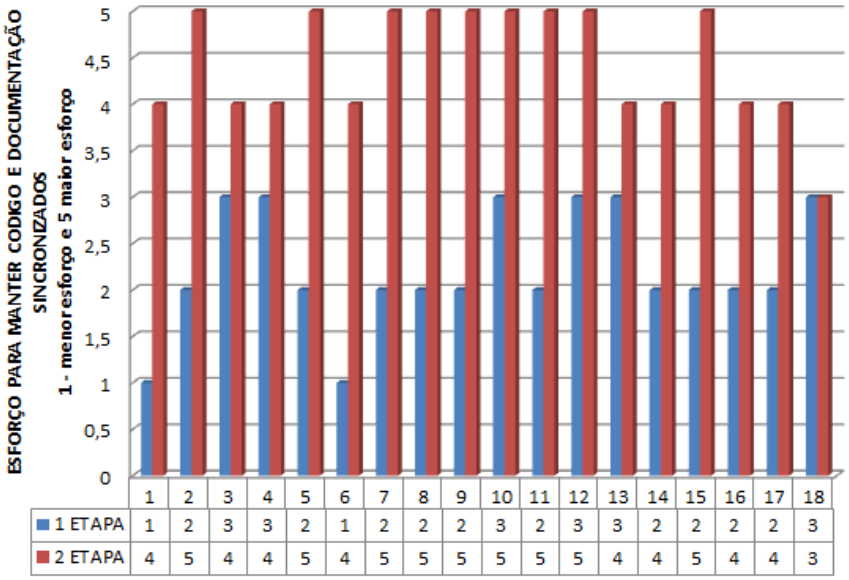
Tabela 7.2: Resultados do experimento quanto ao esforço em minutos para desenvolver, manter e evoluir com e sem a TF2BD.

	ESFORÇO AFERIDO EM MINUTOS					
	DESENVOLVIMENTO		MANUTENÇÃO		EVOLUÇÃO	
Part.	1ª ETAPA	2ª ETAPA	1ª ETAPA	2ª ETAPA	1ª ETAPA	2º ETAPA
1	24	55	1	4	9	6
2	25	48	2	6	6	5
3	40	45	1	7	6	7
4	46	49	1	5	11	8
5	22	40	2	4	10	5
6	26	23	2	4	7	6
7	42	44	2	5	10	17
8	62	56	1	4	13	6
9	55	52	4	7	10	10
10	34	43	28	8	6	8
11	28	46	3	6	10	10
12	60	68	0,5	10	12	9
13	33	41	0,5	5	10	10
14	54	41	6	17	14	5
15	30	48	3	6	10	11
16	31	50	2	3	28	10
17	33	47	1	6	11	10
18	41	46	2	6	15	10

na evolução, sendo um na primeira etapa e o outro na segunda. A Tabela 7.2 apresenta os *outliers* destacando-os na cor vermelha. Esses *outliers* podem ser justificados pela baixa frequência do uso de ferramentas de modelagem pelo participantes 7, 10, 12 e 14 e pela questão do tempo do treinamento ter sido considerada insuficiente pelos participantes 10 e 16. Os *outliers* foram mantidos na amostra, da mesma forma que em Oliveira et al. [71], já que esses valores não foram ocasionados por erros de medição ou condução do experimento.

O esforço para manter o código e a documentação sincronizados também foi aferido. A Figura 7.4 apresenta um gráfico com o resultado da percepção do esforço por participante para manter esse sincronismo com e sem o suporte da TF2BD. As barras em azul representam a percepção do esforço, usando TF2BD, e as barras em vermelho representam o esforço sem TF2BD. A percepção do esforço foi mensurada pelos participantes através da atribuição de uma escala de valores discretos que variava de 1 a 5, sendo o valor 1 considerado o menor esforço e o valor 5, o maior esforço possível.

Na Seção 7.11.1, os resultados sobre o esforço, apresentados nesta seção, serão analisados e discutidos.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1 ETAPA	1	2	3	3	2	1	2	2	2	3	2	3	3	2	2	2	2	3
2 ETAPA	4	5	4	4	5	4	5	5	5	5	5	5	4	4	5	4	4	3

Figura 7.4: Percepção do esforço para manter código e documentação sincronizados na 1ª Etapa e na 2ª Etapa do experimento

7.10.2 Resultados Relacionados à Eficácia

A Tabela 7.3 apresenta os resultados relacionados à eficácia com o uso de dois tratamentos: com suporte da TF2BD (1ª Etapa) e sem o suporte da TF2BD (2ª Etapa) durante o desenvolvimento, a manutenção e a evolução do objeto proposto (*WordCount*). A eficácia esta sendo aferida em função dos erros que são apresentados[6], utilizando a equação $(1/(quantidade\_erros + 1))$, e detalhados por participante e pela aplicação dos dois tratamentos para cada fator.

A equação utilizada para apresentar os resultados representa o inverso da quantidade de erros, ou seja, um valor igual a 1 indica que não houve erros e que um valor próximo a 0 indica muitos erros. Dessa forma, quanto mais próximo de 0 for o valor mais erros ocorreram e quanto mais próximo de 1 menos erros ocorreram. Por exemplo, o participante 6 conseguiu desenvolver, manter e evoluir em ambas as etapas praticamente sem erros, pois todos os seus valores foram 1, com exceção da segunda etapa do desenvolvimento. Já o participante 15 apresentou pelo menos um erro em praticamente todas as etapas, com exceção da primeira etapa de manutenção.

Foram encontrados 3 *outlier* na primeira etapa e 3 na segunda etapa do desenvolvimento, além de 2 *outliers* na manutenção, sendo ambos na primeira etapa. A Tabela 7.3 apresenta os *outliers* destacando-os na cor vermelha. No caso da eficácia

[6]Os erros foram contabilizados com base em um código de referência criado pelos pesquisadores para encontrar erros de lógica e de documentação. Os erros de sintaxe foram apontados pelo compilador.

Tabela 7.3: Resultados do experimento quanto a eficácia para desenvolver, manter e evoluir com e sem a TF2BD.

	EFICÁCIA AFERIDA PELO INVERSO DA QUANTIDADE DE ERROS					
	DESENVOLVIMENTO		MANUTENÇÃO		EVOLUÇÃO	
Part.	1ª ETAPA	2ª ETAPA	1ª ETAPA	2ª ETAPA	1ª ETAPA	2º ETAPA
1	0,50	0,33	1	0,25	1	0,2
2	**1**	0,16	1	0,33	1	0,33
3	0,33	**1**	1	0,5	1	0,14
4	0,33	0,25	1	1	0,5	0,33
5	0,50	0,5	1	0,33	1	1
6	**1**	0,5	1	1	1	1
7	0,16	0,33	1	0,33	1	0,33
8	0,20	0,5	**0,5**	0,25	0,5	0,33
9	0,50	0,5	1	1	0,5	1
10	0,33	0,33	**0,5**	1	1	1
11	0,50	0,33	1	1	1	0,33
12	0,33	0,125	1	1	0,5	0,33
13	0,50	**1**	1	0,33	1	1
14	0,10	**1**	1	1	0,5	1
15	0,50	0,5	1	0,2	0,33	0,5
16	**1**	0,33	1	0,5	0,33	0,5
17	0,33	0,5	1	0,25	0,5	0,5
18	0,16	0,25	1	0,33	0,2	0,5

relacionada ao desenvolvimento, os *outliers* encontrados são justificáveis pelo fato de que somente os participantes 2, 6 e 16 concluíram sem erros a primeira etapa e somente os participantes 3, 13 e 14 concluíram sem erros a segunda etapa. Enquanto os *outliers* relacionados à manutenção, indicam que somente os participantes 8 e 10 apresentaram erros na primeira etapa.

A taxa elevada de erros apresentadas para desenvolver, manter e evoluir pode ser justificada, tendo em vista o primeiro contato dos participantes com a abordagem, a apresentação de novos conceitos e as alegações de alguns participantes quanto ao tempo de treinamento ser insuficiente para o experimento. Os *outliers* não foram desconsiderados pelas mesmas questões apresentadas na Seção 7.10.1.

Na Seção 7.11.2, os resultados sobre a eficácia, apresentados nesta seção, serão analisados e discutidos.

7.10.3 Resultados Relacionados a Percepção de Qualidade

A percepção da qualidade por ser algo subjetivo foi aferida por meio da aplicação de um questionário que pode ser encontrado no Apêndice E. Este

questionário foi utilizado também para obter a percepção do usuário quanto à execução do experimento. Os resultados das questões objetivas estão em forma de gráficos.

O primeiro gráfico pode ser visto na Figura 7.5. Por meio dele, é possível observar a percepção dos participantes em relação à execução do experimento. A maioria dos participantes, 61%, classifica o experimento como bem executado, 39%, como razoável e 0%, como mal executado. Os comentários realizados com relação ao experimento foram direcionados à questão do tempo do treinamento e à execução do experimento. Dos 18 participantes do experimento, 10 fizeram comentários neste contexto.

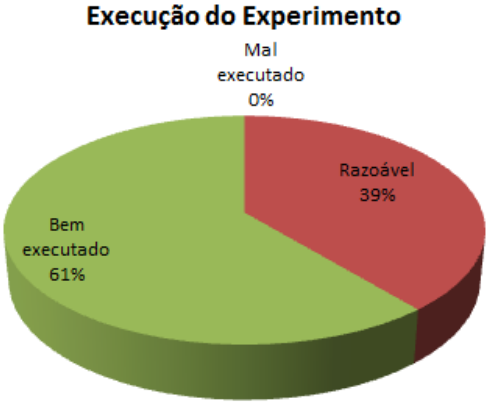
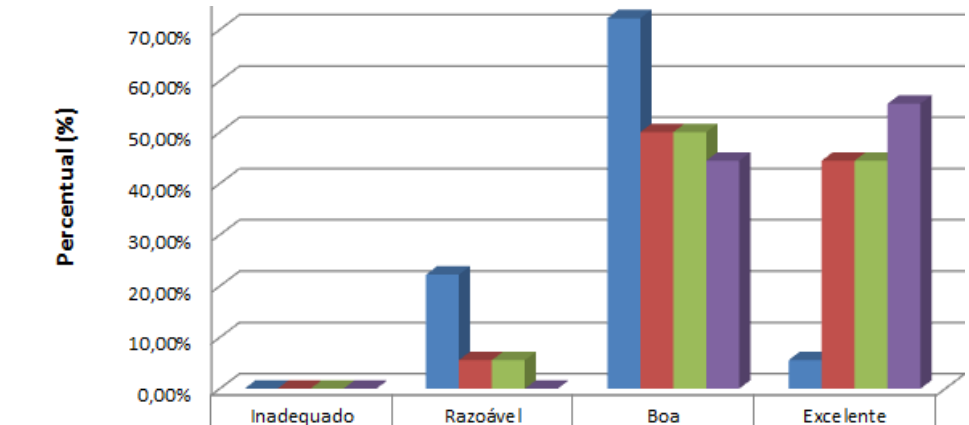


Figura 7.5: Percepção de satisfação com a execução do experimento

A Figura 7.6 apresenta o gráfico que avalia o suporte da TF2BD para as tarefas de desenvolvimento, manutenção, evolução e documentação. Para o suporte de cada tarefa, os participantes poderiam classificá-lo como inadequado, razoável, bom ou excelente. O gráfico em barras exibe o suporte à tarefa de desenvolvimento por meio da barra azul, na qual 72,22% dos participantes considerou o suporte bom. O suporte a tarefa de manutenção, representado pela barra vermelha, foi considerado bom por 50% e excelente por 44%, da mesma forma que o suporte para a tarefa de evolução representado pela barra verde. O suporte a tarefa de documentação representado pela barra roxa foi considerado bom por 44,44% e excelente por 55,56% dos participantes.

O resultado relacionado à questão do uso pode ser visto pelo gráfico apresentado na Figura 7.7. Este gráfico apresenta a forma como os usuários utilizariam a TF2BD. Primeiro os usuários foram questionados quanto ao uso da ferramenta conforme ela foi apresentada, questão representada pela barra azul, e posteriormente os usuários foram questionados se a TF2BD oferecesse outros mecanismos de interação



	Inadequado	Razoável	Boa	Excelente
DESENVOLVIMENTO	0,00%	22,22%	72,22%	5,56%
MANUTENÇÃO	0,00%	5,56%	50,00%	44,44%
EVOLUÇÃO	0,00%	5,56%	50,00%	44,44%
DOCUMENTAÇÃO	0,00%	0,00%	44,44%	55,56%

Figura 7.6: Avaliação da TF2BD em relação ao suporte oferecido ao desenvolvimento, a manutenção, a evolução e a documentação.

para gerar os modelos como eles a usariam, questão representada pela barra vermelha. No primeiro caso, 61,11% optou em usá-la apenas para projetos acadêmicos, enquanto, no segundo caso, 72,22% utilizaria tanto para projetos acadêmicos quanto para projetos comerciais. Alguns comentários feitos sobre essas questões apontaram a necessidade de se construir uma forma mais interativa para construção dos modelos.

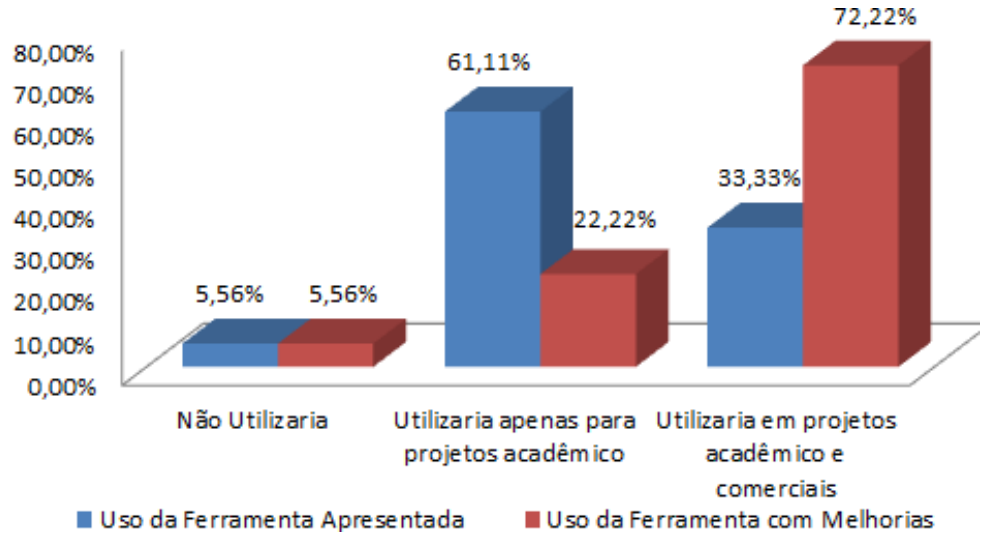


Figura 7.7: Avaliação da ferramenta em relação ao uso.

Na Seção 7.11.3, os resultados, sobre a percepção da qualidade apresentados nesta seção, serão analisados e discutidos.

7.11 Análise e Interpretação dos Resultados

Os resultados apresentados na Seção 7.10 serão analisados e interpretados por meio de observação e testes estatísticos. A análise e interpretação dos resultados serão distribuídas em seções, assim como, ocorreu na apresentação dos resultados. A primeira Seção 7.11.1 apresentará a análise sobre os resultados relacionados ao esforço, a Seção 7.11.2 apresentará a análise dos resultados relacionados à eficácia, e a Seção 7.11.3 apresentará a análise sobre a percepção de qualidade.

7.11.1 Análise e Interpretação dos Resultados Relacionados ao Esforço

A Figura 7.8 apresenta a média do esforço dos participantes por etapa e por tarefa. Analisando as médias dos esforços apresentadas nessa figura, pode-se observar que o esforço realizado na 1ª Etapa, representado pela barra azul, foi menor que o esforço da 2ª Etapa, representado pela barra vermelha, para os suportes às tarefas de desenvolvimento e manutenção e foi maior para o suporte a tarefa de evolução. Esse resultado foi considerado bom, visto que a TF2BD é apenas um protótipo, e obteve tempos melhores ou próximos quando comparada a uma solução comercial[7]. É importante ressaltar que a TF2BD desenvolvida neste trabalho não tem cunho comercial, pois seu propósito é demonstrar que o *framework* proposto no Capítulo 4 permite ser estendido para desenvolver ferramentas que suportem atividades de desenvolvimento, manutenção, evolução e documentação de *software Big Data*, baseados em *MapReduce*.

A simples utilização da média como teste, não é uma forma confiável para determinar se as amostras são estatisticamente diferentes. Dessa forma, as métricas obtidas na primeira e na segunda etapa foram analisadas utilizando testes estatísticos. A ferramenta OriginPro 9.1 foi utilizada como apoio para realização dos testes estatísticos, bem como no trabalho de Oliveira et al. [71].

Os dados de cada etapa e tarefa foram submetidos a um teste de normalidade antes da escolha do teste estatístico. Esse teste é importante para

[7]A solução comercial utilizada foi o combo do ambiente eclipse, API de *Spark* para Java e o *software* de documentação ArgoUML.

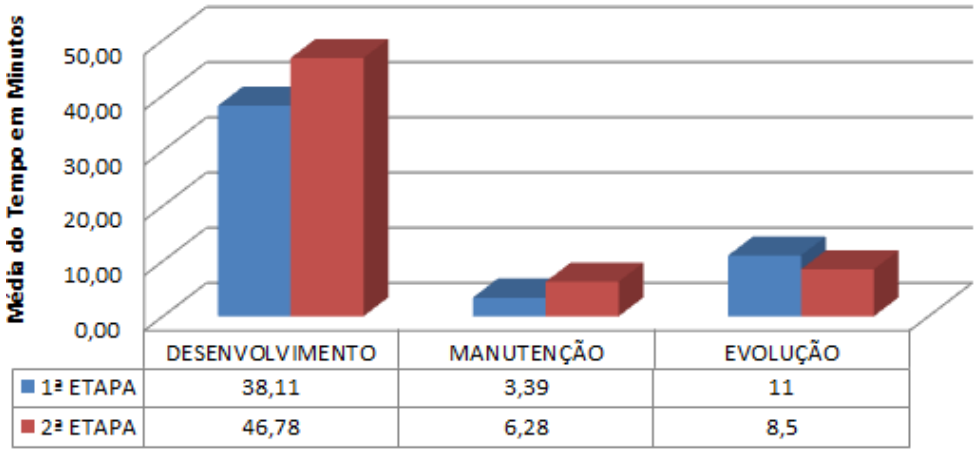


Figura 7.8: Média do esforço dos participantes por etapa e por tarefa.

determinar se os dados seguem uma distribuição normal ou não. Este tipo de informação é necessária, pois alguns testes estatísticos tem como premissa que os dados da amostra constituam uma distribuição normal. A Tabela 7.4 apresenta o resultado da execução do teste de normalidade *Anderson-Darlyng* para os dados referentes ao esforço. Pode-se observar que apenas as amostras relacionadas ao desenvolvimento apresentam uma distribuição normal, assim um teste estatístico que não presuma que as amostras estão distribuídas normalmente é necessário para analisar os dados referentes ao esforço.

Tabela 7.4: Teste de normalidade aplicada as amostras de esforço por etapa e por tarefa.

Teste de Normalidade *Anderson-Darlyng* para esforço				
Etapa	**Tarefa**	α	$\rho - value$	**Normal**
1	Desenvolvimento	0,05	0,12792	Sim
2	Desenvolvimento	0,05	0,06022	Sim
1	Manutenção	0,05	1,83E-10	Não
2	Manutenção	0,05	3,74E-04	Não
1	Evolução	0,05	6,64E-04	Não
2	Evolução	0,05	3,50E-02	Não

Com base no teste de normalidade, foi escolhido o teste não-paramétrico de *Wilcoxon* com amostras pareadas para avaliar as hipóteses relacionadas ao esforço neste experimento. A Tabela 7.5 apresenta os resultados da aplicação do teste em relação ao esforço despendido em cada tarefa com o apoio dos diferentes tratamentos. Pode-se observar que os testes se demonstraram estatisticamente diferentes somente para os fatores dos suportes às tarefas de desenvolvimento e manutenção. Dessa forma, as hipóteses H_0 *esforço desenvolvimento* e H_0 *esforço manutenção* foram rejeitas e as questões

QP1 e QP2 foram respondidas de forma positiva, ou seja, os esforços para desenvolver e manter podem ser considerados menores utilizando TF2BD do que não a utilizando.

Tabela 7.5: Teste não-paramétrico de *Wilcoxon* aplicado as amostras de esforço por etapa e tarefa.

Teste não-paramétrico de *Wilcoxon* com amostras pareadas para o esforço						
Trat. 1	**Trat. 2**	**Objeto**	**Tarefa**	α	$\rho - value$	**Estatisticamente Diferentes**
TF2BD	Manual	WordCount	Desenvolvimento	0,01	0,00541	Sim
TF2BD	Manual	WordCount	Manutenção	0,01	0,00172	Sim
TF2BD	Manual	WordCount	Evolução	0,01	0,0426	Não

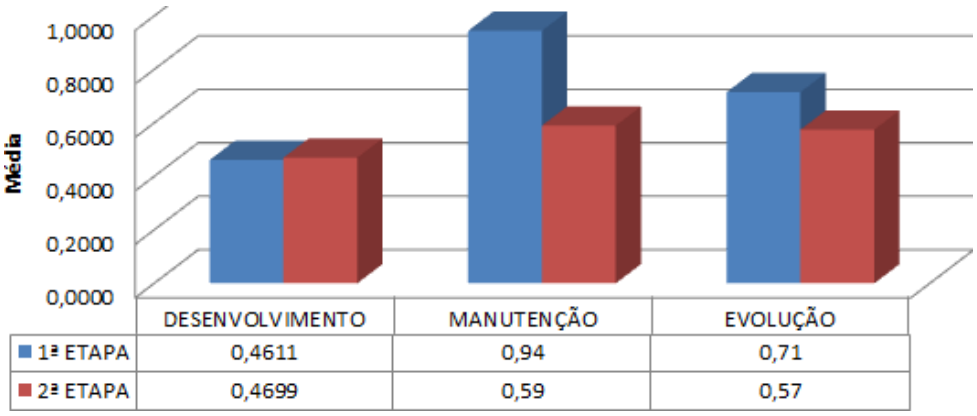
No entanto, a hipótese H_0 *esforço evolução* não pode ser rejeitada, pois as amostras não são estatisticamente diferentes. Isso não quer dizer que as amostras são estatisticamente iguais, mas que não há evidências, com as amostras apresentadas, que comprovem que elas são estatisticamente diferentes. Dessa forma, a QP3 foi respondida de forma negativa, ou seja, o esforço utilizando TF2BD para a tarefa de evolução não pode ser considerado menor que o esforço de evoluir sem a utilização da TF2BD. Mesmo os esforços não se demonstrando estatisticamente diferentes, os participantes obtiveram um esforço médio próximo. Se for levado em consideração que a TF2BD é apenas um protótipo e que as limitações apontadas pelos participantes podem ser resolvidas provavelmente esse esforço será reduzido.

Ainda sobre o esforço, a hipótese sobre H_0 *esforço documentação* foi analisada com base na percepção do esforço de manter a documentação e o código fonte sincronizados. Com base nos resultados, observou-se que, utilizando a TF2BD 55,55% dos participantes considerou que o esforço é baixo e 33,33% que o esforço é médio. Enquanto, sem a utilização da TF2BD 44,44% dos participantes considerou o esforço de sincronização difícil e 50% considerou o esforço de sincronização muito difícil[8]. Dessa forma, a hipótese H_0 *esforço documentação* foi rejeitada e a questão QP4 foi respondida de forma positiva, ou seja, o esforço para manter a documentação e o código fonte sincronizados é menor utilizando a TF2BD do que o esforço sem a utilização da TF2BD.

[8]Os percentuais foram alcançados usando a seguinte classificação de esforço: muito baixo para valores iguais a 1, baixo para valores iguais a 2, médio para valores iguais a 3, difícil para valores iguais a 4 e muito difícil para valores iguais a 5

7.11.2 Análise e Interpretação dos Resultados Relacionados à Eficácia

A Figura 7.9 apresenta a média do inverso da quantidade de erros dos participantes por etapa e por tarefa. Pode-se observar que no desenvolvimento a média de erros é praticamente igual com e sem o suporte a TF2BD, enquanto na manutenção e na evolução a média dos erros são menores com a utilização da TF2BD do que sem a sua utilização. Lembrando que quanto mais próximo de 1 menor é a quantidade de erros e quanto mais próximo de zero maior é a quantidade de erros.



	DESENVOLVIMENTO	MANUTENÇÃO	EVOLUÇÃO
1ª ETAPA	0,4611	0,94	0,71
2ª ETAPA	0,4699	0,59	0,57

Figura 7.9: Média do inverso da quantidade de erros dos participantes por etapa e por tarefa.

A taxa de erros elevada nos dois tratamentos decorre por razões distintas. Usando a TF2BD, os participantes foram apresentados a uma nova abordagem de desenvolvimento que não pode ser desvinculada da documentação e do protótipo da ferramenta (TF2BD) que suporta essas tarefas. Por a TF2BD ser ainda um protótipo, a falta de alguns itens como validação em tempo de compilação dos modelos e uma interatividade mais adequada com o usuário elevou a quantidade de erros. Enquanto sem a utilização da TF2BD, a complexidade dos conceitos associados a *MapReduce* surge, além do fato do usuário ter que utilizar duas ferramentas, uma para documentar e outra para adequar e compilar o código desenvolvido.

Da mesma forma que a análise realizada na Seção 7.11.1, a simples análise da média não é suficiente para determinar se as amostras são estatisticamente diferentes. Assim, testes estatísticos devem ser aplicados, mas antes a premissa da normalidade da distribuição deve ser analisada. Para isso, o teste de *Anderson-Darlyng* mais uma vez foi aplicado às amostras. Os resultados da aplicação do teste são apresentados na Tabela 7.6.

Tabela 7.6: Teste de normalidade aplicada as amostras de erros por etapa e por tarefa.

Teste de Normalidade *Anderson-Darlyng* para eficácia				
Etapa	**Tarefa**	α	$\rho - value$	**Normal**
1	Desenvolvimento	0,05	0,00271	Não
2	Desenvolvimento	0,05	0,00114	Não
1	Manutenção	0,05	1,05E-14	Não
2	Manutenção	0,05	2,57E-05	Não
1	Evolução	0,05	2,24E-05	Não
2	Evolução	0,05	2,18E-04	Não

Pode-se observar por meio da Tabela 7.6 que nenhuma das amostras obedecem uma distribuição normal, dessa forma o teste não-paramétrico de *Wilcoxon* com amostras pareadas também foi escolhido para analisar as amostras referentes à eficácia neste experimento. A Tabela 7.7 apresenta os resultados da aplicação do teste em relação as amostras de erros por tarefa.

Tabela 7.7: Teste não-paramétrico de *Wilcoxon* aplicado as amostras de erros por tarefa.

Teste não-paramétrico de *Wilcoxon* com amostras pareadas para eficácia						
Trat. 1	**Trat. 2**	**Objeto**	**Tarefa**	α	$\rho - value$	**Estatisticamente Diferentes**
TF2BD	Manual	WordCount	Desenvolvimento	0,01	0,98901	Não
TF2BD	Manual	WordCount	Manutenção	0,01	0,00244	Sim
TF2BD	Manual	WordCount	Evolução	0,01	0,21191	Não

Analisando a Tabela 7.7, pode-se observar que apenas o suporte a tarefa de manutenção se demonstrou estatisticamente diferente. Dessa forma, é possível afirmar que a hipótese H_0 *eficácia manutenção* pode ser rejeitada e a questão QP6 pode ser respondida de forma positiva, ou seja, a eficácia na manutenção se demonstrou maior utilizando TF2BD do que a eficácia sem utilizar TF2BD. No entanto, as hipóteses H_0 *eficácia desenvolvimento* e H_0 *eficácia evolução* não puderam ser rejeitadas, pois não se mostraram estatisticamente diferentes. Dessa forma, as questões QP5 e QP7 não puderam ser respondidas positivamente, ou seja, a eficácia no desenvolvimento e na evolução não é maior utilizando TF2BD do que a eficácia sem utilizar TF2BD.

Considerou-se que o resultado alcançado foi positivo mesmo que as hipóteses nulas tenham sido rejeitadas nos casos do desenvolvimento e da evolução. Essa consideração é justificável pelo foto de que a TF2BD é apenas um protótipo, que as médias dos resultados alcançados pela ferramenta são semelhantes aos resultados

da ferramenta comercial utilizada e que a TF2BD pode evoluir de um protótipo para um produto comercial.

7.11.3 Análise e Interpretação dos Resultados Relacionados à Percepção de Qualidade

Finalizando a análise do experimento, os resultados relacionados à percepção da qualidade vão ser considerados. Os resultados vão ser apresentados por suporte oferecido às tarefas de desenvolvimento, de manutenção, de evolução e de documentação. Posteriormente, os resultados relacionados à utilização da ferramenta vão ser apresentados.

O suporte à tarefa de desenvolvimento foi considerado pelos participantes razoável por 22,22%, bom por 72,22% e excelente por 5,56%. Dessa forma, a hipótese H_0 *qualidade desenvolvimento* foi rejeitada, e a questão QP8 foi respondida positivamente, ou seja, a percepção da qualidade do desenvolvimento é maior com a TF2BD do que sem a TF2BD. Por sua vez, o suporte as tarefas de manutenção e de evolução apresentaram os mesmos percentuais. Foram considerados razoáveis por 5,56%, bons por 50% e excelentes por 44,44%. Assim, as hipóteses H_0 *qualidade manutenção* e H_0 *qualidade evolução* foram rejeitadas, e as perguntas QP9 e QP10 foram respondidas positivamente, ou seja, a percepção da qualidade da manutenção e da evolução, utilizando a TF2BD é maior que sem a TF2BD. A respeito do suporte a tarefa de documentação 44,44% dos participantes o considerou bom, e 55,56% o considerou excelente, nesse contexto, a hipótese H_0 *qualidade documentação* foi rejeitada, e a questão QP11 foi respondida de forma positiva, ou seja, a percepção da qualidade da documentação é maior, utilizando TF2BD do que sem utilizar TF2BD.

Os participantes também foram questionados a respeito da utilização da ferramenta. Considerando que a ferramenta é um protótipo e apresenta algumas limitações de cunho de usabilidade, que foram apontadas pelos participantes, apenas 33% afirmou que utilizaria a TF2BD em projetos acadêmicos e comerciais e 61,11% afirmou que a utilizaria apenas em projetos acadêmicos. Essa resposta era esperada, pois além da abordagem ser diferente do seu dia a dia (código fonte completo nos modelos) as limitações do protótipo poderiam induzir a não utilização. Nesse sentido, os participantes foram questionados, considerando que a TF2BD apresentasse novas

formas de interação. Mais uma vez, o resultado foi como esperado: 22,22% afirmou que a usaria apenas para projetos acadêmicos e 72,22% afirmou que a usaria para projetos acadêmicos e comerciais.

O protótipo da TF2BD se demonstrou aceitável, mas para que possa ser considerado comercialmente deve apresentar melhorias de usabilidade e validação.

7.12 Validade

Uma grande questão sobre os resultados obtidos em um experimento é sua confiabilidade, ou seja, até que ponto eles podem ser considerados válidos [41, 102]. Aspectos relacionados à validade devem ser previstos durante a fase de planejamento de um experimento para tentar evitar situações que possam invalidá-lo [102]. Nesse contexto, como o experimento apresentado está seguindo a mesma linha de Oliveira et al. [71]; as ameaças à validação do experimento e à forma de mitigá-las são semelhantes. As validações comumente utilizadas na literatura são: de conclusão, de construção, interna e externa [41].

A validade de conclusão procura garantir que conclusões corretas sobre os resultados possam ser alcançadas. Alguns dos problemas mais comuns recorrentes às conclusões que foram consideradas no experimento de Oliveira et al. [71] foram apresentados juntamente com a forma de mitigá-los:

- **Usar um teste com baixo poder estatístico:** Testes estatísticos podem revelar padrões em um conjunto de dados. Logo, o teste estatístico deve ser escolhido conforme o desenho do experimento para não aumentar a probabilidade de conclusões erradas. A escolha do teste estatístico utilizado seguiu os mesmos critérios utilizados por Oliveira et al. [71];

- **Não considerar premissas dos testes estatísticos:** Os testes estatísticos possuem premissas que devem ser consideradas para sua correta utilização. Caso essas premissas não sejam consideradas, os resultados dos testes podem induzir a conclusões erradas. Assim como em Oliveira et al. [71], os dados foram analisados previamente para garantir que eles sigam uma distribuição normal, pois esta é uma premissa necessária para aplicação de muitos testes estatísticos;

- **Confiabilidade das medições:** Procura garantir a confiabilidade do processo de medição. O ideal é que a medição não utilize julgamento humano. Por isso, medições objetivas que não necessitam de interpretação humana são mais confiáveis. Neste trabalho, duas medidas objetivas (esforço e eficácia) e outra medida subjetiva (percepção da qualidade) foram utilizadas;

- **Confiabilidade da aplicação dos tratamentos:** Deve garantir que todos os participantes utilizem os mesmos tratamentos. Assim como, no trabalho de Oliveira et al. [71], no qual foram utilizados dois tratamentos com apenas uma implementação para cada tratamento. Dessa forma, se descartou a possibilidade de aplicações diferentes, pois todos os participantes foram submetidos a mesma abordagem;

- **Distúrbio na configuração do experimento:** Elementos externos ao estudo podem interferir nos resultados. Para mitigar essa ameaça, o experimento foi executado em um ambiente tranquilo e com acesso controlado para favorecer a concentração dos participantes.

A validade interna procura garantir que, durante o experimento, só haja interferência controlada entre o tratamento aplicado e os resultados alcançados, ou seja, se os resultados foram gerados pelos tratamentos aplicados sem interferência controlada. Uma lista de ameaças à validade interna, assim como tratá-las serão apresentadas:

- **História**: Acontecimentos históricos podem interferir nos resultados, pois podem influenciar diretamente os participantes. Por exemplo, indisposição depois de um final de semana e/ou euforia depois de grandes eventos. Para tentar mitigar esses problemas, a data para a realização do experimento foi marcada antecipadamente, conforme disponibilidade dos participantes, tentou-se prever um dia sem feriados, finais de semanas e eventos culturais e sociais que pudessem afetar os participantes do experimento;

- **Maturação**: Essa ameaça está relacionada ao comportamento ou reação dos participantes com o passar do tempo. Os participantes podem não se empolgar com as tarefas previstas no experimento ou podem aprender por repetição agindo positivamente. Assim como no trabalho de Oliveira et al. [71], a maturação foi

utilizada contra a TF2BD pela definição da utilização da TF2BD como o primeiro tratamento a ser aplicado, evitando que o *crossover* tivesse que ser utilizado. Dessa forma, o tratamento sem o uso da TF2BD é privilegiado, pois já existe um certo aprendizado sobre os objetos envolvidos no experimento;

- **Mortalidade**: Diz respeito à quantidade de participantes que por desistência não concluem o experimento. Foi tomado o cuidado de analisar se as desistências representaram um número substancial da amostra, além de identificar as razões;
- **Ameaças Sociais**: Permite garantir que os resultados não sofram interferência de fatores sociais como rivalidade, amizade ou inimizade. Para evitar esses problemas, foi incluído um breve resumo sobre o objetivo do experimento na ficha de identificação do perfil do participante. Esse resumo ressaltava que o experimento não se tratava de competição ou de uma avaliação pessoal do participante. Um reforço sobre essas questões, em forma de uma apresentação, foi realizado depois que os participantes foram selecionados.

A validade de construção consiste em garantir se o tratamento reflete a causa de maneira satisfatória, assim como se o resultado reflete o efeito de forma satisfatória. Esta validação está ligada diretamente aos aspectos relevantes do projeto do experimento do ponto de vista dos pesquisadores e dos participantes. Ameaças como a expectativa do pesquisador e as possíveis suposições dos participantes, quanto aos objetivos, devem ser consideradas. Para mitigá-las, os pesquisadores não puderam auxiliar os participantes durante o experimento e os participantes foram instruídos a seguir o experimento de maneira imparcial para que sua opinião pessoal não interferisse nos resultado.

A validade externa trata da generalização dos resultados do experimento. Alguns problemas e como foram tratados são apresentados a seguir:

- **Seleção dos participantes**: Pode ser um problema, pois a amostra de participantes selecionada pode não ser representativa frente à população desejada. Para tentar ampliar a capacidade de generalização dos resultados, foram utilizados alunos de graduação, pós-graduação e alguns profissionais;
- **Ferramentas inadequadas**: O problema ocorre na escolha de ferramentas que não sejam adequadas e de alguma maneira favoreçam alguns dos tratamentos

executados. As ferramentas selecionadas são amplamente conhecidas e utilizadas tanto para atividades acadêmicas como profissionais;

- **História**: Esse problema está relacionado com o dia da execução do experimento. Fatores externos podem afetar os resultados, como por exemplo, se for escolhido algum dia que, por algum evento único (manifestação que atrapalhe a chegada dos participantes), leve os participantes a saírem da sua rotina. Essa mudança de rotina pode influenciar os resultados. Para diminuir esse risco, a data do experimento foi escolhida de forma cautelosa a evitar situações atípicas.

- **Generalização para indústria**: Capacidade de generalizar os resultados para a prática industrial. Apesar das principais ameaças a validade externa terem sido consideradas, o experimento proposto apresenta uma baixa generalização de resultados e isto prejudica a validade externa, visto que não foi possível aplicar mais de um objeto durante o experimento.

A ordem de prioridade das validações é outro aspecto interessante, relacionado à validação, que deve ser considerado. Essa ordem é determinada em função do objetivo do experimento. A ordem de prioridade, para o tipo de experimento que foi realizado, deve ser: validade interna, validade externa, validade de construção e validade de conclusão [71]. Apesar da validade externa ter sido prejudicada por questões de generalização da prática industrial, as demais validações foram bem tratadas.

7.13 Análise dos Trabalhos Relacionados

Os exemplos ilustrativos desenvolvidos foram utilizados para demonstrar a aplicação do protótipo da ferramenta criada (*TF2BD*), utilizando a arquitetura do *framework* proposto. Assim, os itens avaliados (conforme Seção 3.1.8) entre os trabalhos relacionados são retomados para comparação e análise com a proposta deste trabalho. Esses itens acrescentam contribuições, assim como alguns dos objetivos especificados no início do trabalho. Na Tabela 7.8, pode-se observar que foi acrescentada uma nova coluna, em azul, que descreve as características do trabalho proposto, ao mesmo tempo que realiza a comparação com os trabalhos relacionados.

A construção dos exemplos ilustrativos foi feita de forma detalhada, seguindo as tarefas sugeridas pelo processo proposto na Seção 4.6. Os modelos criados através dos editores apresentados e das definições de transformação executadas comprovam que *TF2BD* apoiou o usuário em todas as tarefas da atividade de desenvolvimento de *software* (análise, projeto e implementação) de maneira prática, ou seja, de uma maneira concreta e demonstrada por meio de um estudo experimental apresentado neste capítulo. Esse estudo permitiu que os usuários pudessem utilizar e se expressar com relação ao suporte oferecido pela TF2BD. O apoio oferecido pela TF2BD se deu de maneira semiautomática, porque alguns modelos foram gerados manualmente pelo usuário, através do apoio de editores para facilitar sua criação, edição ou visualização, e outros foram gerados automaticamente, através do apoio das definições de transformação. Os trabalhos de Rajbhoj et al. [82], Chen et al. [21] e Guerriero et al. [40] são os únicos trabalhos que oferecem soluções práticas. Os trabalhos de Chen et al. [21] e Guerriero et al. [40] oferecem soluções para as fases de análise e projeto, enquanto o trabalho de Rajbhoj et al. [82] oferece soluções para as fases de projeto e implementação. No entanto, o trabalho de Guerriero et al. [40] também oferece um nível de automaticidade semelhante ao oferecido no protótipo da ferramenta desenvolvida neste trabalho para a fase de projeto.

Os códigos fontes gerados de forma textual e apresentados demonstram que a ferramenta permite gerar código de maneira completa e não somente parte dele como muitas ferramentas comerciais apresentam em outros contextos. A geração desse código completo só foi possível, porque os modelos manipulados pela *TF2BD* foram descritos de forma a permitir que a lógica de negócio fosse incorporada neles. Essa característica permite afirmar que a ferramenta auxilia na sincronização da documentação com o código fonte preservando a lógica de negócio de forma, independente de plataforma. Essa afirmação foi atestada durante o experimento, quando 88% dos participantes afirmou que o esforço para manter o código e a documentação sincronizados é baixo (55%) ou médio (33%). Apenas o trabalho de Rajbhoj et al. [82] ofereceu uma solução para implementação, mas a solução é limitada para a implementação do modelo *MapReduce* da *Hadoop* e gera somente o esqueleto do código fonte. A lógica de negócio deveria ser incorporada posteriormente.

É possível afirmar também, com base na percepção de qualidade dos participantes do experimento, que a ferramenta criada auxilia nas tarefas de

documentação, manutenção e evolução. Essa percepção é justificável, pois como a lógica de negócio é preservada nos modelos, a realização dessas tarefas se torna menos repetitiva. A ferramenta auxilia na documentação, pois os modelos gerados são uma documentação em si. Os modelos utilizados pela ferramenta só podem ser gerados conforme as especificações que os descrevem fazendo com que os mesmos se tornem confiáveis. A ferramenta auxilia na manutenção, através de correções realizadas nos modelos com o apoio dos editores fornecidos e pela facilidade da geração do código fonte atualizado. O mesmo ocorre com a tarefa de evolução, novas características podem ser adicionadas nos modelos através do uso dos editores e o código fonte pode ser gerado novamente através das transformações. Como essas duas tarefas ocorrem manipulando os modelos, é possível afirmar que o código fonte textual, gerado sempre, será um representação fiel dos modelos, fazendo com que a tarefa de documentação se torne praticamente transparente.

Os trabalhos de Rajbhoj et al. [82], Chen et al. [21] e Guerriero et al. [40] podem auxiliar na documentação, visto que usam modelos para realizar as fases de análise e/ou projeto, mas não chegam ao mesmo nível de confiabilidade, gerado pela ferramenta desenvolvida nesse trabalho, visto que eles não apresentam uma solução para geração do código fonte completo, enquanto na ferramenta proposta esse código é gerado automaticamente em função dos modelos criados. A falta de uma solução para geração de código fonte completo em seus trabalhos pode comprometer a documentação, assim como dificultar as tarefas de manutenção/evolução, pois as mesmas serão feitas de maneira manual e desvinculadas de forma direta da documentação.

Os resultados obtidos no estudo experimental comprova que a TF2BD suporta as atividades de desenvolvimento, manutenção, evolução e documentação para *software Big Data* baseados em *MapReduce*. O foco da TF2BD é a preservação da lógica de negócio e a automaticidade de tarefas usando modelos. As características apresentadas para *TF2BD*, nesta seção, são inerentes do *framework* proposto (*F2BD*), visto que essa ferramenta é fruto da utilização do mesmo. Dessa forma, pode-se concluir que a arquitetura do *framework*, baseada na abordagem MDE, é viável e que outras ferramentas podem ser construídas usando a sua descrição. Nesse sentido, pode-se afirmar também que os metamodelos, modelos e definições de transformação

comentados no trabalho também foram validados, pois foram comprovadamente utilizados pelos participantes durante o experimento.

A base da arquitetura do *framework* são os conceitos de *weaving* de modelo e o processo de desenvolvimento em Y, pois essa arquitetura foi pensada para que o *framework* possa evoluir através da adição de novos PDMs, oferecendo uma possível solução para o problema de integração de *frameworks*, levantado por Madhavji [60]. A ferramenta apresentada também atende a questão de produtividade levantada por DeLine [27], pois a ferramenta oferece parte do processo feito de maneira automática, e atende os anseios de Anderson [3] por ferramentas que apoiem a atividade de desenvolvimento.

Muitos *software* para *Big Data* estão sendo desenvolvidos, assim soluções que visem flexibilizar e ao mesmo tempo preservar a lógica de negócio desses *software* são necessárias. Dessa forma, o *framework* proposto é uma solução que vem contribuir de maneira favorável para o contexto do desenvolvimento de *software* para *Big Data* que ainda está em pleno processo de evolução.

7.14 Síntese

Neste capítulo, um estudo experimental foi apresentado para analisar o suporte ao desenvolvimento, manutenção, evolução e documentação com e sem o uso da TF2BD. A avaliação foi feita em função do esforço, eficácia e percepção da qualidade dos participantes do experimento.

A TF2BD, mesmo sendo um protótipo, apresentou um esforço menor para desenvolver, manter e documentar. Enquanto em termos de eficácia, o estudo apontou que não houve diferenças significativas para desenvolver, manter, evoluir e documentar. Esse resultado foi considerado positivo, visto que a TF2BD é apenas um protótipo e foi comparada a ferramentas comerciais. Por meio da percepção da qualidade, observou-se que a proposta da TF2BD foi aceita principalmente se houver melhorias em termos de usabilidade e questões de validações.

O capítulo é encerrado com a discussão dos resultados alcançados e com a comparação da nossa proposta para suportar a criação de *software* para *Big Data* com

Tabela 7.8: Comparação e análise dos trabalhos relacionados

	Trabalhos Relacionados							Proposta
Itens Avaliados	Research Directions for Engineering Big Data Analytics Software [78]	Embrace the Challenges: Software Engineering in a Big Data World [3]	Research Opportunities for the Big Data Era of Software Engineering [27]	Big Picture of Big Data Software Engineering: With Example Research Challenges [60]	Early Experience with Model-driven Development of MapReduce based Big Data Application [82]	Big Data System Development: An Embedded Case Study with a Global Outsourcing Firm [21]	Towards a Model-driven Design Tool for Big Data Architectures [40]	Framework Proposto (F2BD)
Trabalho Teórico ou Prático	T	T	T	T	P	P	P	P
Abordagem Principal	Não oferece	Não oferece	Não oferece	Não oferece	Baseado em MDD	Baseado em ADD	Baseado em MDE	Baseado em MDE
Usa modelo como artefato de desenvolvimento	Não	Não	Não	Não	Sim	Sim	Sim	Sim
Auxilia na fase de Análise	Não	Não	Não	Não	Não	Manual	Manual	Manual
Auxilia na fase de Projeto	Não	Não	Não	Não	Manual	Manual	Automático	Automático
Auxilia na fase de implementação	Não	Não	Não	Não	Automático	Não	Não	Automático
Gera o código fonte completo	Não	Não	Não	Não	Não	Não	Não	Sim
Auxilia na Manutenção/ Evolução	Não	Não	Não	Não	Não	Não	Não	Sim
Auxilia na Documentação	Não	Não	Não	Não	Manual	Manual	Manual	Automático
Realizou estudo experimental	Não	Não	Não	Não	Não	Não	Não	Sim

os trabalhos relacionados (i.e. aqueles trabalhos que foram apresentados no Capítulo 3).

8 Considerações Finais

Nesse trabalho, apresentou-se um *framework*, baseado em MDE e *Weaving*, para suportar o desenvolvimento de forma semiautomática da atividade de desenvolvimento de *software*, para plataforma de *Big Data*, que usem o modelo *MapReduce* (F2BD). O *F2BD* inclui uma arquitetura que utiliza conceitos de MDE, *Weaving* e Desenvolvimento de *Software* baseado em Y. O *F2BD* é baseado em modelos em todos os estágios de desenvolvimento, e a informação de um estágio para outro estágio é passado através de transformações. Para demonstrar a viabilidade da arquitetura do *F2BD*, a ferramenta *TF2BD* foi construída com base no modelo *MapReduce*, utilizado por *Spark*. A *TF2BD* foi construída como um *plug-in* para o IDE *Eclipse*, possuindo editores de modelos PIM, PDM, modelo de *Weaving*, PSM abstrato e PSM concreto, e reutiliza ferramentas como o editor e motor de transformação de ATL.

Um processo que orienta a utilização do *F2BD* também foi apresentado. O processo proposto, assim como, a *TF2BD* foram testados através da criação de dois exemplos ilustrativos: Contar Palavras e Chamada de Emergência. Nesses exemplos, mostrou-se como a *TF2BD* apoia nas tarefas sugeridas pelo processo proposto. Os editores do PIM, do PDM, do modelo de *Weaving* e dos PSMs também foram apresentados, assim como os papéis e os artefatos produzidos em cada tarefa do processo proposto. O PIM (modelo utilizado para descrever a lógica de negócio de cada exemplo ilustrativo) foi criado e editado pelo editor fornecido, assim como o PDM (modelo utilizado para descrever as características relacionadas a *MapReduce* de *Big Data*) e o modelo *Weaving* (utilizado para entrelaçar os elementos do PIM e do PDM). Apresentou-se, também, algumas definições de transformação, que, quando executadas pelo motor de transformação, geraram novos modelos, como por exemplo, PSM abstrato e PSM concreto, ambos gerados para os dois exemplos ilustrativos. O PSM abstrato foi gerado pela execução de uma definição de transformação que teve como entradas o PIM, PDM e modelo de *Weaving*. O PSM concreto foi gerado por outra definição de transformação que teve como entrada o PSM abstrato e o modelo da API *Spark*. A última transformação teve como entrada o PSM concreto para gerar o

código fonte em Java do sistema de *software* baseado na API de *Spark* para plataforma de *Big Data*.

A implementação dos exemplos ilustrativos, i.e. Contar Palavras e Chamada de Emergência, permitiu descrever como a *TF2BD* auxilia nas tarefas da atividade de desenvolvimento de *software* (análise, projeto e implementação), além de auxiliar em tarefas de documentação, manutenção e evolução do *software* construído. A manutenção e a evolução podem ser feitas com correções e/ou adições de novas funcionalidades nos modelos, através do uso dos editores fornecidos e da execução das definições de transformações propostas. Uma documentação é feita de maneira transparente, visto que o código fonte sempre será um espelho dos modelos criados, pois o mesmo é gerado através desses modelos.

O estudo experimental aplicado nesse trabalho demonstrou que a *TF2BD* realmente oferece suporte às tarefas de desenvolvimento, manutenção, evolução e documentação como descrito durante a implementação dos exemplos ilustrativos, mas ela precisa melhorar em termos de usabilidade e questões de validação.

A maioria dos trabalhos sobre MDE encontrados na literatura gera apenas o esqueleto do código, sendo que *body* dos métodos devem ser desenvolvidos pelos programadores. Por outro lado, o surgimento de Alf não trouxe grande expectativa para definição do corpo (*body*) dos métodos, pois ele é mais uma linguagem textual. Para contornar essas limitações com UML e Alf, a proposta incluiu a linguagem *VisualAlf*, que introduz uma notação gráfica para o Alf e um metamodelo para *VisualAlf*, que pode ser incluído em outros metamodelos. A proposta feita é introduzir o metamodelo do *VisualAlf* no metamodelo de UML e no metamodelo do PSM abstrato. Desta forma, assegurou-se o suporte para geração do código de forma mais completa, pois o algoritmo executado pelos métodos podem ser descritos no PIM (e.g. PIM conforme UML mais *VisualAlf*) e propagado e aperfeiçoado no PSM abstrato e transformado para o PSM concreto (e.g. em Java ou C#). Sendo assim, o *VisualAlf* foi proposto como uma solução viável para que UML (i.e. metamodelo de UML estendido com *VisualAlf*) possa dar suporte de forma mais completa à descrição do comportamento dos métodos. Assim, nesse contexto, pode-se afirmar que a *TF2BD* também permite que a lógica de negócio seja preservada no modelo; afirmação comprovada pelos participantes do experimento realizado, que criaram modelos que incorporavam a lógica neles.

As características discutidas e atribuídas à *TF2BD*[1] só puderam ser criadas, pois essa ferramenta foi construída com base na arquitetura do *F2BD*. Assim, pode-se concluir que o *F2BD* é viável e pode ser utilizado para a construção de outras ferramentas. Pode-se concluir, também, que a arquitetura do *F2BD* pode evoluir através da inclusão de outros PDM, pois ela foi construída com base nos conceitos de *weaving* de modelos e do processo de desenvolvimento baseado em Y.

O desenvolvimento para *Big Data* ainda é emergente e muitos desafios ainda existem. Muitas mudanças ainda estão por vir, por isso, uma solução que possa dar suporte ao desenvolvimento de *software* na plataforma de *Big Data*, que usem *MapReduce*, e que minimizem os impactos das mudanças, preservando a lógica de negócio, é essencial neste momento de incerteza. O *framework* proposto, como apresentado neste trabalho, é uma solução que visa minimizar o impacto destas mudanças, preservando a lógica do negócio, além de auxiliar na atividade de desenvolvimento de *software*, automatizando algumas tarefas e facilitando outras.

8.1 Contribuições Científicas e Tecnológicas

A abordagem proposta apresenta as seguintes contribuições científicas e tecnológicas.

As contribuições científicas são:

- Estudo e aplicabilidade da Engenharia Dirigida por Modelos para desenvolver *software* para *MapReduce* de *Big Data*;
- Proposta de um *framework*, baseado em MDE, para desenvolvimento de *software* para *MapReduce* de *Big Data*;
- Proposta de um metamodelo e notação gráfica para VisualAlf.

As contribuições tecnológicas são:

- Desenvolvimento de uma ferramenta que implementa o *framework* proposto como *plug-in* para *Eclipse*;

[1] as características atribuídas a *TF2BD* são: suporte a atividade de desenvolvimento de *software* para plataforma *Spark* de forma semiautomática, auxilia na preservação da lógica de negócio, assim como, na documentação, na manutenção e evolução do *software* desenvolvido.

- Proposta de metamodelo de PDM para *Big Data*, de PSM Abstrato para *Spark* e extensão do metamodelo de UML para permitir que a lógica de negócio possa ser acoplada diretamente ao modelo;
- Geração automática do código completo e não só de seu esqueleto através de transformações;
- Proposta de definições de transformações específicas para *Spark*, que permitam gerar o código completo.

8.2 Desafios e Limitações da Solução Proposta

A arquitetura do *framework* proposto, por utilizar o Processo de Desenvolvimento em Y, pode ser estendida por meio da inclusão de vários PDMs que vão ser entrelaçados com o PIM, através da operação de *weaving* entre modelos de forma serial, ou seja, para cada PDM adicionado um modelo específico de *weaving* será criado em função desse PDM e do PIM. Recentemente, Stefanello et al. [94] apresenta uma forma paralela de realizar o entrelaçamento entre o PIM e vários PDMs. Segundo Stefanello et al. [94], essa forma de entrelaçar agiliza o desenvolvimento, pois irá gerar apenas um modelo de *weaving* e não vários como na forma serial, utilizada no *framework* proposto. O trabalho proposto neste documento, por focar especificamente no aspecto *MapReduce* para *Big Data*, testou a arquitetura somente com um PDM. Dessa forma, um desafio interessante seria realizar um estudo com vários PDMs, analisando qual abordagem, a forma serial, utilizada neste trabalho, ou a forma paralela, proposta por Stefanello et al. [94], oferece um melhor desempenho para o contexto de desenvolvimento para *Big Data*. Em termos de limitação, a arquitetura só permite a geração do modelo de *weaving* de forma manual e não oferece uma solução para especificação dos requisitos e automatização dos testes de *software*.

A automaticidade proposta pelo *F2BD*, feita por meio das transformações entre modelos, também apresentaram desafios e limitações que foram discutidos na Seção 5.4.2. Uma limitação que deve ser ressaltada, é que as transformações foram construídas levando em consideração apenas o modelo de *weaving*, que entrelaça o PIM com o PDM, e aspectos específicos da API do *framework Spark*. Desse modo, se

novos aspectos forem incorporados as transformações devem ser modificadas para se adequar a eles.

A limitação em termo de usabilidade foi observada pelos participantes do experimento. Não foi realizado um teste específico de usabilidade, mas os comentário são pertinentes, pois o perfil dos participantes era, em sua maioria, de profissionais que atuam no mercado. Durante o experimento, muitos participantes se queixaram da usabilidade dos editores oferecidos. Sugestões como a melhoria de nomes e ícones identificadores foram feitas, além da sugestão de novas formas de interagir com os modelos. Algumas funcionalidades, como validação em tempo de compilação do modelo gerado e mais mensagens para os usuários, também foram sugeridas, para que o suporte oferecido pela *TF2BD* seja mais adequado.

Essas questões, de certa maneira, eram esperadas, visto que a *TF2BD* foi construída apenas como um protótipo para mostrar que o *framework* proposto possibilita o suporte às tarefas de desenvolver, de manter, de evoluir e de documentar *software*, baseados em *MapReduce* de *Big Data*. Dessa forma, o protótipo oferecia apenas o mínimo necessário para que as tarefas pertinentes ao desenvolvimento de *software* pudessem ser realizadas. Nesses termos, outro desafio a ser considerado é: realizar um estudo para determinar a melhor maneira de incorporar a lógica de negócio nos modelos de forma mais ágil e interativa.

Em termos de avaliação, a interface gráfica do PIM proposta na Seção 4.4 não foi validada durante o experimento por questões de tempo e disponibilidade dos participantes do experimento. O experimento também foi considerado limitado por questões como quantitativo de participantes, tempo de aplicação e pelo objeto aplicado no estudo.

Mesmo com todo trabalho realizado, pode-se observar que algumas melhorias e estudos precisam ser realizados. Essa constatação é boa, pois representa que há espaço para novos trabalhos.

8.3 Trabalhos Futuros

Dentre os trabalhos futuros, pode-se destacar:

- Estender a arquitetura do *framework* para permitir que ele atenda às demais fases de um processo de *software*, como por exemplo, especificação de requisitos e a geração de casos de teste de maneira automática;
- Propor novas características para a abordagem, através da proposta de novos PDMs;
- Realizar estudo comparativo entre a abordagem serial e paralela para realizar entrelaçamento entre modelos no contexto do *framework* proposto;
- Aperfeiçoar as transformações para que elas manipulem outras estruturas e conceitos utilizados em modelos como sobrecarga e/ou sobreposição, além de permitir analisar vários modelos de *weaving* simultaneamente;
- Propor uma possível automatização da geração do modelo de *weaving*. Dotando a ferramenta proposta de alguma inteligência que possa suportar a criação automática ou semiautomática de modelos de *weaving*;
- Propor novas formas de incorporar lógica de negócio aos modelos;
- Realizar estudos específicos de usabilidade para encontrar problemas dentro da ferramenta proposta;
- Realizar estudo avaliativo mais aprofundado com um exemplo ilustrativo mais complexo;
- Realizar estudo de caso em empresas.

8.4 Publicações

Um artigo foi publicado:

O artigo intitulado "*Developing software systems to Big Data platform based on MapReduce model: An approach based on Model Driven Engineering*", sobre este trabalho, foi publicado na revista *Information and Software Technology*. Essa revista possui Fator de Impacto de 2.694 e seu ISSN é 0950-5849. O trabalho pode ser encontrado pela URL "https://doi.org/10.1016/j.infsof.2017.07.006" [48].

Um artigo está em fase de submissão:

O artigo intitulado "*An Approach to Embed Action Language in Models: VisualAlf*", sobre este trabalho, foi submetido à revista *Information and Software Technology*. Essa revista possui Fator de Impacto de 2.694 e seu ISSN é 0950-5849.

Referências Bibliográficas

[1] ACADGILD. Acadgild official blog on big data, hadoop, devops & android. Disponível em: < https://acadgild.com/blog/ >. Acessado em: 29 Ago. 2016.

[2] ALMEIDA, J. P., DIJKMAN, R., VAN SINDEREN, M., AND PIRES, L. F. On the notion of abstract platform in mda development. In *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International* (Sept 2004), pp. 253–263.

[3] ANDERSON, K. M. Embrace the challenges: Software engineering in a big data world. In *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on* (May 2015), pp. 19–25.

[4] APACHE. Apache cassandra. Disponível em: < http://cassandra.apache.org/ >. Acessado em: 21 Jul. 2016.

[5] APACHE. Apache hbase. Disponível em: < http://hbase.apache.org/ >. Acessado em: 21 Jul. 2016.

[6] APACHE. Apache hive. Disponível em: < http://hive.apache.org/ >. Acessado em: 21 Jul. 2016.

[7] APACHE. Apache spark: Lightning-fast cluster computing. Disponível em: < http://spark.apache.org/ >. Acessado em: 21 Jul. 2016.

[8] APACHE. Spark 0.9.0. Disponível em: < https://spark.apache.org/docs/0.9.0/index.html >. Acessado em: 20 Jul. 2016.

[9] BAGHERI, E., KONTOGI, K., MADHAVJ, N., AND MIRANSKYY, A. Special issue on software engineering for big data. Disponível em: < http://wikicfp.com/cfp/servlet/event.showcfp?eventid=48608&copyownerid=81646 >. Acessado em: 24 Nov. 2016.

[10] BARESI, L., MENZIES, T., AND METZGER, A. Special issue on big data software engineering. Disponível em: < http://static.springer.com/sgw/documents/1550278/application/pdf/bigDataCfP+%281%29.pdf >. Acessado em: 24 Nov. 2016.

[11] BELANGOUR, A., BÉZIVIN, J., AND FREDJ, M. Towards a new software development process for mda. In *Proc. of the European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA)* (2006).

[12] BERNSTEIN, P. Applying model management to classical meta data problems. In *Proceedings of the 2003 CIDR Conference* (2003).

[13] BÉZIVIN, J., AND GERARD, S. A preliminary identification of mda components. In *In Generative Techniques in the context of Model Driven Architecture* (2002).

[14] BÉZIVIN, J., HAMMOUDI, S., LOPES, D., AND JOUAULT, F. Applying mda approach to b2b applications: A road map. In *Workshop on Model Driven Development (WMDD 2004) at ECOOP 2004* (2004), SpringerVerlag.

[15] BÉZIVIN, J., HAMMOUDI, S., LOPES, D., AND JOUAULT, F. *Knowledge Sharing in the Integrated Enterprise: Interoperability Strategies for the Enterprise Architect*. Springer US, Boston, MA, 2005, ch. B2B Applications, BPEL4WS, Web Services and .NET in the Context of MDA, pp. 225–236.

[16] BHARDWAJ, V., AND JOHARI, R. Big data analysis: Issues and challenges. In *Electrical, Electronics, Signals, Communication and Optimization (EESCO), 2015 International Conference on* (Jan 2015), pp. 1–6.

[17] BOLLATI, V. A., VARA, J. M., JIMÉNEZ, A., AND MARCOS, E. Applying {MDE} to the (semi-)automatic development of model transformations. *Information and Software Technology 55*, 4 (2013), 699 – 718.

[18] BRAMBILLA, M., CABOT, J., AND WIMMER, M. *Model-Driven Software Engineering in Practice*, 2 ed. Morgan & Claypool, 2017.

[19] CARVALHO, M. V., LOPES, D., AND ABDELOUAHAB, Z. *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*. Springer International Publishing, Cham, 2015, ch. A Framework Based on Model Driven Engineering to Support Schema Merging in Database Systems, pp. 397–405.

[20] CENTER, I. I. Planning guide getting started with big data. Disponível em: < http://www.intel.com/content/www/us/en/big-data/getting-started-with-big-data-planning-guide.html >. Acessado em: 15 Ago. 2016.

[21] CHEN, H. M., KAZMAN, R., HAZIYEV, S., AND HRYTSAY, O. Big data system development: An embedded case study with a global outsourcing firm. In *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on* (May 2015), pp. 44–50.

[22] CHOO, J., AND PARK, H. Customizing computational methods for visual analytics with big data. *IEEE Computer Graphics and Applications 33*, 4 (July 2013), 22–28.

[23] CORPORATION, N. I. Labview. Disponível em: < http://www.ni.com/labview/pt/ >. Acessado em: 24 Jun. 2016.

[24] CUADRADO, J. S., IZQUIERDO, J. L. C., AND MOLINA, J. G. Applying model-driven engineering in small software enterprises. *Science of Computer Programming 89, Part B* (2014), 176 – 198. Special issue on Success Stories in Model Driven Engineering.

[25] DA SILVA, A. R. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures 43* (2015), 139 – 155.

[26] DAVIES, J., GIBBONS, J., WELCH, J., AND CRICHTON, E. Model-driven engineering of information systems: 10 years and 1000 versions. *Science of Computer Programming 89, Part B* (2014), 88 – 104. Special issue on Success Stories in Model Driven Engineering.

[27] DELINE, R. Research opportunities for the big data era of software engineering. In *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on* (May 2015), pp. 26–29.

[28] DIMITRIOS, K., RICHARD, P., AND FIONA, P. *The Epsilon Transformation Language*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 46–60.

[29] DONG, X., LI, R., HE, H., ZHOU, W., XUE, Z., AND WU, H. Secure sensitive data sharing on a big data platform. *Tsinghua Science and Technology 20*, 1 (Feb 2015), 72–80.

[30] EBACH, M. C., MICHAEL, M. S., SHAW, W. S., GOFF, J., MURPHY, D. J., AND MATTHEWS, S. Big data and the historical sciences: A critique. *Geoforum 71* (2016), 1 – 4.

[31] ECLIPSE. Atlas definition language. Disponível em: < https://eclipse.org/atl/ >. Acessado em: 10 Jul. 2016.

[32] EMANI, C. K., CULLOT, N., AND NICOLLE, C. Understandable big data: A survey. *Computer Science Review 17* (2015), 70 – 81.

[33] FABRO, M., BÉZIVIN, J., JOUAULT, F., AND VALDURIEZ, P. Applying generic model management to data mapping. In *Proc. of Base de Données Avancées (BDA 2005)* (2005).

[34] FABRO, M., BÉZIVIN, J., AND VALDURIEZ, P. Weaving models with the eclipse amw plugin. In *In Eclipse Modeling Symposium, Eclipse Summit Europe* (2006).

[35] FABRO, M., JEAN, B., FRÉDÉRIC, J., AND ERWAN, B. Amw: A generic model weaver. In *Premieres Journées sur lÍngénierie Dirigée par les Modeles* (2005).

[36] FAHAD, A., ALSHATRI, N., TARI, Z., ALAMRI, A., KHALIL, I., ZOMAYA, A. Y., FOUFOU, S., AND BOURAS, A. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Transactions on Emerging Topics in Computing 2*, 3 (Sept 2014), 267–279.

[37] FANG, H., ZHANG, Z., WANG, C. J., DANESHMAND, M., WANG, C., AND WANG, H. A survey of big data research. *IEEE Network 29*, 5 (September 2015), 6–9.

[38] GIL, D., AND SONG, I. Modeling and management of big data: Challenges and opportunities. *Future Generation Computer Systems 63* (2016), 96 – 99. Modeling and Management for Big Data Analytics and Visualization.

[39] GROUP, A., LINA, AND INRIA. *Atlas Transformation Language - ATL: User Manual version 0.7*, February 2006.

[40] GUERRIERO, M., TAJFAR, S., TAMBURRI, D. A., AND DI NITTO, E. Towards a model-driven design tool for big data architectures. In *Proceedings of the 2Nd International Workshop on BIG Data Software Engineering* (New York, NY, USA, 2016), BIGDSE '16, ACM, pp. 37–43.

[41] GUROV, D., AND AMARAL, E. Introdução à engenharia de software experimental. Tech. rep., Universidade Federal do Rio de Janeiro, 2002.

[42] GUTIÉRREZ, J., ESCALONA, M., AND MEJÍAS, M. A model-driven approach for functional test case generation. *Journal of Systems and Software 109* (2015), 214 – 228.

[43] HAMMOUDI, S., ALOUINI, W., LOPES, D., AND HUCHARD, M. Towards A semi-automatic transformation process in MDA: architecture, methodology and first experiments. *IJISMD 1*, 4 (2010), 48–76.

[44] HOMES, A. *Hadoop in Practice*. Manning Publications Co, 2012.

[45] JAGADISH, H. Big data and science: Myths and reality. *Big Data Research 2*, 2 (2015), 49 – 52. Visions on Big Data.

[46] JAIN, R., SARKAR, P., AND SUBHRAVETI, D. Gpfs-snc: An enterprise cluster file system for big data. *IBM Journal of Research and Development 57*, 3/4 (May 2013), 5:1–5:10.

[47] JIAMIN, L., AND JUN, F. A survey of mapreduce based parallel processing technologies. *China Communications 11*, 14 (Supplement 2014), 146–155.

[48] JUNIOR, O. S. S., LOPES, D., SILVA, A. C., AND ABDELOUAHAB, Z. Developing software systems to big data platform based on mapreduce model: An approach based on model driven engineering. *Information and Software Technology 92*, Supplement C (2017), 30 – 48.

[49] KARAU, H. *Fast Data Processing with Spark*. Packt Publishing, 2013.

[50] KEIM, D., QU, H., AND MA, K. L. Big-data visualization. *IEEE Computer Graphics and Applications 33*, 4 (July 2013), 20–21.

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

[52] KOZIEL, S., LEIFSSON, L., LEES, M., KRZHIZHANOVSKAYA, V., DONGARRA, J., SLOOT, P. M., BRUEL, J.-M., COMBEMALE, B., OBER, I., AND RAYNAL, H. Mde in practice for computational science. *Procedia Computer Science 51* (2015), 660 – 669.

[53] KRISHINA, K. Spark sql use case (911) - emergency helpline number data analysis. Disponível em: < https://acadgild.com/blog/spark-sql-use-case-911-emergency-helpline-number-data-analysis/ >. Acessado em: 29 Ago. 2016.

[54] KRUCHTEN, P. *Introdução Ao Rup - Rational Unified Process*. Ciência Moderna, 2003.

[55] LEAVITT, N. Storage challenge: Where will all that big data go? *Computer 46*, 9 (September 2013), 22–25.

[56] LEE, J.-G., AND KANG, M. Geospatial big data: Challenges and opportunities. *Big Data Research 2*, 2 (2015), 74 – 81. Visions on Big Data.

[57] LIN, J. In defense of mapreduce. *IEEE Internet Computing 21*, 3 (2017), 94–98.

[58] LIU, J., LIU, F., AND ANSARI, N. Monitoring and analyzing big traffic data of a large-scale cellular network with hadoop. *IEEE Network 28*, 4 (July 2014), 32–39.

[59] LUBLINSKY, AND BORIS. *Professional Hadoop Solutions*. John Wiley & Sons, 2013.

[60] MADHAVJI, N. H., MIRANSKYY, A., AND KONTOGIANNIS, K. Big picture of big data software engineering: With example research challenges. In *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on* (May 2015), pp. 11–14.

[61] MARTÍNEZ-GARCÍA, A., GARCÍA-GARCÍA, J., ESCALONA, M., AND PARRA-CALDERÓN, C. Working with the hl7 metamodel in a model driven engineering context. *Journal of Biomedical Informatics 57* (2015), 415 – 424.

[62] MATOS, P. Um framework de segurança baseado em engenharia dirigida por modelos para plataformas de computação em nuvem: Uma abordagem para modelos saas. Master's thesis, Universidade Federal do Maranhão, 2015.

[63] MATOS, P., LOPES, D., AND ABDELOUAHAB, Z. A model driven engineering approach to support the development of secure software as a service. *JSW 11*, 2 (Feb 2016), 118–132.

[64] MATTURDI, B., XIANWEI, Z., SHUAI, L., AND FUHONG, L. Big data security and privacy: A review. *China Communications 11*, 14 (Supplement 2014), 135–145.

[65] MEDIA, O. R. *Big Data Now*. O' Reilly Media, 2014.

[66] MELLOR, S., KENDALL, S., UHL, A., AND WEISE, D. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 2004.

[67] NEVO, D., NEVO, S., KUMAR, N., BRAASCH, J., AND MATHEWS, K. Enhancing the visualization of big data to support collaborative decision-making. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on* (Jan 2015), pp. 121–130.

[68] OLDEVIK, J. *MOFScript User Guide - version 0.9*, 2nd ed. Eclipse Project, http://www.eclipse.org/gmt/mofscript/doc/MOFScript-User-Guide-0.9.pdf, November 2006.

[69] OLIVEIRA, J. Uma abordagem baseada em engenharia dirigida por modelos para suportar o teste de sistemas de software na plataforma de computação em nuvem. Master's thesis, Universidade Federal do Maranhão, 2012.

[70] OLIVEIRA, J., LOPES, D., ABDELOUAHAB, Z., CLARO, D., AND HAMMOUDI, S. *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*. Springer International Publishing, Cham, 2015, ch. Model Driven Testing for Cloud Computing, pp. 297–304.

[71] OLIVEIRA, P. Athena: uma arquitetura e uma ferramenta para auxiliar o desenvolvimento de sistemas baseados em inteligência computacional. Master's thesis, Universidade Federal do Piauí, 2015.

[72] OMG. Action language for foundational uml (alf) - concrete syntax for a uml action language. Disponível em: < http://www.omg.org/spec/ALF/1.0.1 >. Acessado em: 15 Mar. 2016.

[73] OMG. Meta objectfacility (mof) specification - omg document number (2002-04-03). Disponível em: < http://www.omg.org/spec/MOF/1.4/PDF >. Acessado em: 15 Ago. 2016.

[74] OMG. Model driven architecture (mda)- document number ormsc/2001-07-01. Disponível em: < http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01 >. Acessado em: 15 Jan. 2016.

[75] OMG. Omg unified modeling language specification. Disponível em: < http://www.omg.org/spec/UML/1.5 >. Acessado em: 15 Mar. 2016.

[76] OMG. Semantics of a foundational subset for executable uml models (fuml). Disponível em: < http://www.omg.org/spec/FUML/1.1 >. Acessado em: 15 Mar. 2016.

[77] OMG. *MOF 2.0 - Query/View/Transformation Specification*. http://www.omg.org/cgi-bin/doc?ptc/2007-07-07, July 2007.

[78] OTERO, C. E., AND PETER, A. Research directions for engineering big data analytics software. *IEEE Intelligent Systems 30*, 1 (Jan 2015), 13–19.

[79] PARREIRAS, F. S. *Model-Driven Engineering Foundations*. Wiley-IEEE Press, 2012, pp. 9–20.

[80] PENCHIKALA, S. Big data com apache spark - parte 1: Introdução. Disponível em: < https://www.infoq.com/br/articles/apache-spark-introduction >. Acessado em: 29 Jul. 2016.

[81] PEREIRA, F. Big data e data analysis: Visualização de informação. Master's thesis, Unversidade do Minho - Escola de Engenharia, 2015.

[82] RAJBHOJ, A., KULKARNI, V., AND BELLARYKAR, N. Early experience with model-driven development of mapreduce based big data application. In *2014 21st Asia-Pacific Software Engineering Conference* (Dec 2014), vol. 1, pp. 94–97.

[83] RODRIGUES, A. W. O., GUYOMARCH, F., AND DEKEYSER, J. L. An mde approach for automatic code generation from uml/marte to opencl. *Computing in Science Engineering 15*, 1 (Jan 2013), 46–55.

[84] RUSCIO, D. *SPECIFICATION OF MODEL TRANSFORMATION AND WEAVING IN MODEL DRIVEN ENGINEERING*. PhD thesis, Universita di L Aquila, 2007.

[85] RUSCIO, D. D., PAIGE, R. F., PIERANTONIO, A., HUTCHINSON, J., WHITTLE, J., AND ROUNCEFIELD, M. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming 89* (2014), 144 – 161.

[86] SANDRYHAILA, A., AND MOURA, J. M. F. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE Signal Processing Magazine 31*, 5 (Sept 2014), 80–90.

[87] SCHMIDT, D. C. Guest editors introduction: Model-driven engineering. *Computer 39*, 2 (Feb 2006), 25–31.

[88] SEI. Add (attribute-driven design method). Disponível em: < http://www.sei.cmu.edu/architecture/tools/define/add.cfm >. Acessado em: 20 Mai. 2016.

[89] SENER, S., SARIDOGAN, E., STAUB, S., OZKOSE, H., ARI, E. S., AND GENCER, C. Yesterday, today and tomorrow of big data. *Procedia - Social and Behavioral Sciences 195* (2015), 1042 – 1050.

[90] SINGH, S., AND LIU, Y. A cloud service architecture for analyzing big monitoring data. *Tsinghua Science and Technology 21*, 01 (Feb 2016), 55–70.

[91] SOMMERVILLE, I. *Engenharia de Software, 8 edição*. Pearson Addison-Wesley, 2007.

[92] SOUSA, H., LOPES, D., ABDELOUAHAB, Z., CLARO, D. B., AND HAMMOUDI, S. An approach for model driven testing: framework, metamodels and tools. *Comput. Syst. Sci. Eng. 26*, 4 (2011).

[93] SPARK. Java word count. Disponível em: < https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/JavaWordCount.java >. Acessado em: 11 Jan. 2016.

[94] STEFANELLO, D. Um framework baseado em mde e weaving para suporte ao desenvolvimento de sistemas de software sensíveis ao contexto. Master's thesis, Universidade Federal do Maranhão, 2017.

[95] STEINBERG, D., BUDINSKY, F., PATERNOSTRO, M., AND MERKS, E. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2009.

[96] TAN, Z., NAGAR, U. T., HE, X., NANDA, P., LIU, R. P., WANG, S., AND HU, J. Enhancing big data security with collaborative intrusion detection. *IEEE Cloud Computing 1*, 3 (Sept 2014), 27–33.

[97] TANKARD, C. Big data security. *Network Security 2012*, 7 (2012), 5 – 8.

[98] VARAJAO, J., CUNHA, M., BJORN-ANDERSEN, N., TURNER, R., WIJESEKERA, D., MARTINHO, R., RIJO, R., AND JOKONYA, O. Towards a big data framework for the prevention and control of hiv/aids, tb and silicosis in the mining industry. *Procedia Technology 16* (2014), 1533 – 1541.

[99] VIJAYAKUMAR, V., NEELANARAYANAN, V., ARCHENAA, J., AND ANITA, E. M. Big data, cloud and computing challenges a survey of big data analytics in healthcare and government. *Procedia Computer Science 50* (2015), 408 – 413.

[100] WHITE, T. *Hadoop: The Definitive Guide, Third Edition*. O' REILLY, 2012.

[101] WHITTLE, J., HUTCHINSON, J., AND ROUNCEFIELD, M. The state of practice in model-driven engineering. *IEEE Software 31*, 3 (May 2014), 79–85.

[102] WOHLIN, C., RUNESON, P., HOST, M., OHLSSON, M., REGNELL, B., AND WESSLEN, A. *Experimentation in Software Engineering*. Springer, 2012.

[103] XU, L., JIANG, C., WANG, J., YUAN, J., AND REN, Y. Information security in big data: Privacy and data mining. *IEEE Access 2* (2014), 1149–1176.

[104] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 2–2.

[105] ZHOU, Z. H., CHAWLA, N. V., JIN, Y., AND WILLIAMS, G. J. Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum]. *IEEE Computational Intelligence Magazine 9*, 4 (Nov 2014), 62–74.

A Trabalhos sobre MDE e *Big Data*

Este anexo tem como objetivo descrever alguns trabalhos sobre MDE e *Big Data* nas suas mais diversas aplicabilidades. Com a apresentação desses trabalhos, deseja-se demonstrar ao leitor a maturidade, a potencialidade e a versatilidade da abordagem MDE e como *Big Data* ainda é uma plataforma emergente.

Primeiramente, o foco será nos trabalhos voltados para MDE. Em seguida, serão abordados os trabalhos sobre *Big Data*. Por fim, uma discussão será apresentada sobre MDE e *Big Data*.

A.1 Trabalhos sobre MDE

Davies et al. [26] reportam mais de dez anos de experiência em desenvolvimento e aplicação de tecnologias dirigidas por modelo. Davies et al. [26] explicam a tecnologia *Booster*, que foi construída tendo como influência métodos formais. Esta tecnologia foi utilizada para desenvolver diversos sistemas de informação computacionais. Três sistemas desenvolvidos, usando *Booster*, foram escolhidos para apresentar as lições, que foram aprendidas sobre aplicação de técnicas dirigidas por modelos ao longo destes anos. Algumas das lições enumeradas por Davies et al. [26] são: Na engenharia dirigida por modelos, novas ferramentas serão sempre necessárias; Mudar o sistema é fácil, mas o mais importante são os impactos das mudanças e Tudo o que muda com o modelo deve mudar automaticamente.

Whittle et al. [101] identificaram que um estudo voltado para a aplicação de MDE na indústria, com foco nas experiências de sucesso e fracasso ainda não tinha sido realizado. Eles relatam que os trabalhos encontrados que aplicam MDE para indústria, apenas abrangem aspectos como a modelagem, por exemplo. Nesse trabalho, pesquisaram mais de 450 utilizadores de MDE e entrevistaram de maneira mais detalhada 22 participantes, sendo destes 17 representantes de empresas diferentes e de 9 áreas industriais diferentes. Uma série de conclusões foram obtidas após a análise dos dados coletados. Algumas das conclusões foram: a difusão do uso da

MDE é uma realidade em vários nichos industriais; a aplicação da MDE foi um sucesso utilizando linguagens próprias e onde, como e com quem aplicar são fatores relevantes para usar MDE.

Cuadrado et al. [24] relatam a falta de estudos da aplicação de MDE para pequenas empresas. Eles afirmam que existem diversos estudos de casos da aplicação de MDE, mas para grande empresas. Apesar de admitir que, em ambos os casos, os desafios associados à engenharia de *software* são semelhantes, eles alertam que as soluções devem ser adequadas ao tamanho e a natureza da empresa. Alertam, também, que empresas de pequeno porte são mais flexíveis e responsivas, além de geralmente não terem recursos suficientes para custear a construção de soluções internas. Assim, para pequenas empresas é sugerido utilizar técnicas de MDE para automatizar certos problemas de desenvolvimento como forma de melhorar a produtividade da empresa.

Hutchinson et al. [85] levantam a questão de que não havia sido realizado um trabalho sistemático e multidisciplinar para estudar a eficácia da MDE em termos gerais. Por exemplo, não existem atualmente pesquisas generalizadas e sistemáticas do uso de MDE na indústria. Para Hutchinson et al. [85] existem três problemas chaves: a falta de conhecimento de como MDE é usada na indústria; a falta de entendimento de como fatores sociais afetam o uso de MDE e a falta de avaliação dos aspectos de MDE, além de UML. Eles abordam o problema de duas maneiras diferentes, mas complementares. Primeiro realizam uma pesquisa *online* sobre a implantação e experiência de MDE para prover medidas quantitativas sobre a prática de MDE na indústria; em seguida, eles complementam a pesquisa a partir de alguns dados qualitativos, obtidos através de entrevistas com profissionais da área de MDE. Algumas conclusões alcançadas foram: linguagens específicas de domínio estão ganhando espaço, apenas produtividade não é justificativa suficiente para aplicar MDE, a aplicabilidade de MDE depende do comprometimento organizacional e MDE deve ser usada para o negócio.

Gutierrez et al. [42] garantem que a qualidade do *software* é um dos passos mais críticos no processo de desenvolvimento de *software*. Assim, muitos trabalhos vêm propondo técnicas para garantir esta qualidade na fase de testes do processo de desenvolvimento ao mesmo tempo que tentam reduzir os custos associados. Gutierrez et al. [42] garantem que MDE pode ser uma solução viável e que, no contexto de testes,

ela é conhecida como *Model-Driven Testing (MDT)*. Nesse trabalho, Gutierrez et al. [42] propõem uma abordagem baseada em MDT para um tipo bem definido de testes, os testes funcionais. Eles garantem que a abordagem proposta define um conjunto de modelos, transformações e processos que permitam de uma forma sistemática aplicar testes funcionais a partir de requisitos funcionais para reduzir tempo, assegurar a qualidade e diminuir custos.

Para Pablo et al. [63], na literatura, soluções de segurança para computação em nuvem estão concentradas apenas no código fonte final. No entanto, há uma ausência de discussão sobre abordagens que levem em conta a segurança em *Software* como um Serviço (SaaS) desde a concepção até o código fonte. A abordagem proposta por Pablo et al. [63], em seu trabalho, leva em consideração questões de segurança desde a modelagem do negócio. Para tanto, a abordagem é baseada em MDE e em um mecanismo de segurança para apoiar o desenvolvimento de *software* como serviço.

Rodrigues et al. [83] identificam que os avanços nas engenharias e na comunidade científica vêm proporcionando uma necessidade cada vez maior por processamento paralelo para resolver problemas complexos de larga escala. Apesar das experiências de alto nível, os desenvolvedores deste nicho têm dificuldades para implementar suas aplicações paralelas de forma eficaz. Assim, Rodrigues et al. [83] propõem uma abordagem baseada em MDE para especificar, projetar e gerar aplicações *OpenCL*. As contribuições determinadas por Rodrigues et al. [83] são: Permite que os usuários modelem de forma simples aspectos de distribuição de memória: transferência de dados e alocação de memória; Permite que os usuários modelem a plataforma e executem os modelos de *OpenCL*; Capacidade de transformações inteligentes podem determinar níveis de otimização em comunicação de dados e de acesso a dados.

A ciência computacional auxilia na resolução de problemas complexos. Para Bruel et al. [52], esta área vem se beneficiando cada vez mais dos progressos computacionais, mas as soluções para os problemas atuais estão exigindo novos progressos, entre eles, Bruel et al. [52] destacam, a elevação do nível de abstração. Assim, a questão é como se pode estruturar o conhecimento de especialistas de domínio. Eles lembram que na comunidade de engenharia de *software*, pesquisas com foco em *Domain Specific Modeling Languages (DSML)* são realizadas para desenvolver ferramentas e linguagens de desenvolvimento que permitam especialistas de domínio

desenvolverem soluções de sistemas de forma eficiente. Assim, baseado em experiências concretas, eles acreditam que a *DSML* pode ser a solução entre a lacuna existente entre os cientistas que modelam a solução do problema e os programadores que a implementam. Para tanto, Bruel et al. [52] apresentam como *workbenches* DSML, usados para modelos de engenharia de *software*, podem ser úteis para encerrar a lacuna entre a modelagem e a implementação, através de duas experiências: a primeira uma aplicação de ciência computacional e a segunda no domínio de sistemas agrícolas. Bruel et al. [52] concluem através das investigações que o aumento do nível de abstração, através da utilização de um abordagem baseada em modelo, oferece um verdadeiro valor que deve ser considerado como uma técnica para ser usada mais abrangentemente em aplicações científicas. Bruel et al. [52] concluem, também, que a importância de ferramentas para suportar de maneira amigável a abordagem é fundamental para garantir o sucesso.

Martinez et al. [61] apresentam *Health Level 7 (HL7)* internacional que é uma organização que define padrões associados com sistemas de informação para saúde. Seus membros desenvolvem normas relacionadas com o intercâmbio e modelo de informações de saúde com o objetivo de apoiar a prática clínica, a gestão de desenvolvimento e a avaliação nos serviços da saúde. Este conjunto de normas é conhecido como HL7. A proposta de Martinez et al. [61] é proporcionar uma maneira que os engenheiros de *software* possam modelar suas soluções, usando o metamodelo de UML e o metamodelo de HL7 de forma automática, através do uso de ferramentas de MDE. Desta forma, os engenheiros de *software*, sem experiência em HL7, poderiam modelar problemas do mundo real, através da definição dos requisitos do *software* em UML, que seriam compatíveis com os modelos de domínio HL7 de forma transparente.

A.2 Trabalhos sobre *Big Data*

Jagadish [45] reconhece, em seu trabalho, que a aplicação de *Big Data* vem impactando diversos aspectos de nossa sociedade. Ele afirma que, devido a corrida pela utilização do termo *Big Data*, alguns mitos foram criados. Assim, nesse trabalho, ele desmistifica seis mitos sobre *Big Data*: Tamanho é o que importa; O desafio central de *Big Data* é criar novas arquiteturas computacionais e algoritmos; Análise de dados é

o problema central de *Big Data*; Reuso de dados é a parte fácil; *Data Science* é o mesmo que *Big Data* e *Big Data* é pura excitação.

Para Fang et al. [37], *Big Data* vem trazendo valor para negócios e para pesquisa, mas acompanhado de muitos desafios nas áreas de rede, armazenamento, gerenciamento, análise e ética. Assim, uma discussão sobre os desafios e soluções na indústria e na academia é apresentada a partir da perspectiva de engenheiros, cientistas da computação e estatísticos.

O ser humano sempre buscou por informação e por gerar novas informações através do processamento e manipulação de dados. Para Sener et al. [89], quem mais tira proveito disso são as empresas que procuram armazenar e transformar esses dados para obterem informações relevantes de maneira rápida. Dessa forma, elas obtêm um diferencial competitivo no mercado. Nesse trabalho, Sener et al. [89] fazem um estudo interessante mostrando o ontem, o hoje e o amanhã do *Big Data*.

O cuidado com a saúde é algo relevante, por isso mecanismos que auxiliem na predição de situações de emergência com intuito de preveni-las é algo muito interessante, tanto do ponto de vista da medicina, como do ponto de vista do governo. Nesse contexto, Vijayakumar et al. [99] proveem uma visão geral da análise de *Big Data* para cuidados com a saúde, melhorando sua qualidade ao mesmo tempo que procura diminuir os custos. Eles, também, sugerem onde a análise de *Big Data* pode auxiliar o governo.

Jokonya et al. [98] apresentam uma proposta conceitual de *framework* integrado para *Big Data* para auxiliar na prevenção e controle de HIV/AIDS, Tuberculose (TB) e Silicose (HATS) na indústria da mineração. Jokonya et al. [98] relatam que a mineração na África do Sul vem sofrendo desde 1980 intensamente com HATS. A utilização de *Big Data*, para Jokonya et al. [98], irá possibilitar combinação e correlação de pequenos conjuntos de dados separados para obter informações que auxiliem na tomada de decisão, para que medidas de prevenção e controle das HATS sejam tomadas.

Lee et al. [56] afirmam que os dados geoespaciais vêm crescendo a pelo menos 20% por ano e que uma significante porção desses dados são *Big Data*. Nesse âmbito, eles apresentam, nesse trabalho, os desafios e as oportunidades relacionados aos *Big Data* geoespaciais. Para tanto, vários estudos de caso são comentados para

demonstrar a importância e os benefícios da análise de *Big Data* geoespacial. No mesmo contexto, Lee et al. [56], também, apresentam um projeto de pesquisa que será realizado ao longo de cinco anos.

A.3 Discussão

Na Seção A.1, através dos trabalhos apresentados, tentou-se mostrar aos leitores a maturidade, a potencialidade e os benefícios de MDE. Os primeiros trabalhos apresentados proporcionaram uma visão geral dos resultados da utilização de MDE de maneira prática, ao longo dos anos, em basicamente três eixos: aplicado para sistemas de informação de maneira geral [26], à indústria [85, 101] e às pequenas empresas [24]. Todos esses trabalhos apresentaram lições ou conselhos da melhor forma de se implantar e utilizar MDE. Em seguida, trabalhos nas mais diversas áreas, que utilizaram MDE, foram apresentados, áreas como: Teste de *Software* [42], Segurança de *Software* [63], Geração de Código [83], Ciência Computacional [52] e para Saúde [61]. Esses trabalhos foram apresentados com o intuito de demonstrar que MDE é uma abordagem que pode ser empregada para os mais diversos fins e que está cada vez mais madura. Isso é confirmado por Whittle et al. [101], que constataram em sua pesquisa que MDE é utilizada na indústria, nas mais diversas áreas de atuação e das mais diversas formas.

A observação dos trabalhos apresentados levaram as seguintes conclusões:

- MDE deve ser usada para resolver um problema: O foco na solução de um problema, usando MDE, vai permitir a automatização total ou parcial dele, além de permitir a reutilização dos modelos e transformações para leques de problemas semelhantes. A utilização de MDE apenas por conveniência é amplamente desaconselhável, pois sua implantação requer conhecimento especializado;
- MDE gera mais valor quando aplicada a domínios específicos: Empresas que especializam sua área de atuação, ou seja, focam o desenvolvimento em determinado domínio, conseguem extrair um maior retorno de abordagens como MDE por conseguirem automatizar determinadas tarefas, aumentando a sua produtividade;

- MDE pode ser aplicada para empresas de grande e de pequeno porte: A questão para usar MDE não se concentra no porte da empresa ou na área que ela atua, mas sim nos problemas que ela deseja resolver. Empresas que tenham caráter generalista não vão colher com a mesma eficácia os benefícios que MDE pode proporcionar a empresas com caráter especialista, pois a concepção de soluções baseadas em MDE são mais trabalhosas que abordagens tradicionais;

- MDE requer conhecimento especializado: A utilização de MDE requer capacitação, pois necessita de aplicação de conhecimentos especializados, como de linguagens de transformação e de modelagem de metamodelos. Dependendo da situação, o custo da capacitação pode inviabilizar a implantação da MDE;

- MDE não permite apenas automatização de código: Muitas vezes, se atribui unicamente como objetivo para MDE a geração de código fonte, mas MDE pode ser aplicada para representação, para implantação, para configuração ou para outras situações, dependendo da solução que se deseja alcançar.

Em seguida, na Seção A.2, ofereceu-se uma visão geral de como *Big Data* ainda é um conceito novo que ainda pode ser bastante explorado. Muitas áreas vêm tentando oferecer soluções práticas, utilizando as mais diversas combinações de metodologias e ferramentas disponíveis. *Big Data* tem enorme potencialidade para contribuir nas mais diversas áreas da sociedade seja para saúde, comércio, propaganda, governo, etc. Dessa forma, começou-se apresentando trabalhos que dão uma visão geral [37, 89], destacando o trabalho de Jagadish [45], que procura desmistificar alguns conceitos de *Big Data*. Em seguida, na mesma linha de MDE, apresentou-se trabalhos para áreas mais específicas como, por exemplo: saúde [98, 99] e dados geoespaciais [56].

B Notação Gráfica para VisualAlf

Neste apêndice, a notação gráfica, para algumas instruções de VisualAlf, será apresentada, bem como um exemplo ilustrativo utilizando o VisualAlf Editor.

B.1 Notação Gráfica

Uma notação gráfica para VisualAlf é proposta para que ela seja gráfica e intuitiva como UML. Essa notação gráfica tem uma estrutura modular, pois representa seus elementos e a relação entre eles através de módulos aninhados. A Figura B.1 apresenta um exemplo do aninhamento dos elementos utilizando a notação gráfica proposta. É possível ver que o elemento *UNamespace* contém *Classe*, que por sua vez, *Classe* contém *Método*, *Método* contém um *Bloco*, um *Bloco* contém *Instruções* e que as *Instruções* são logicamente distribuídas através de setas direcionais.

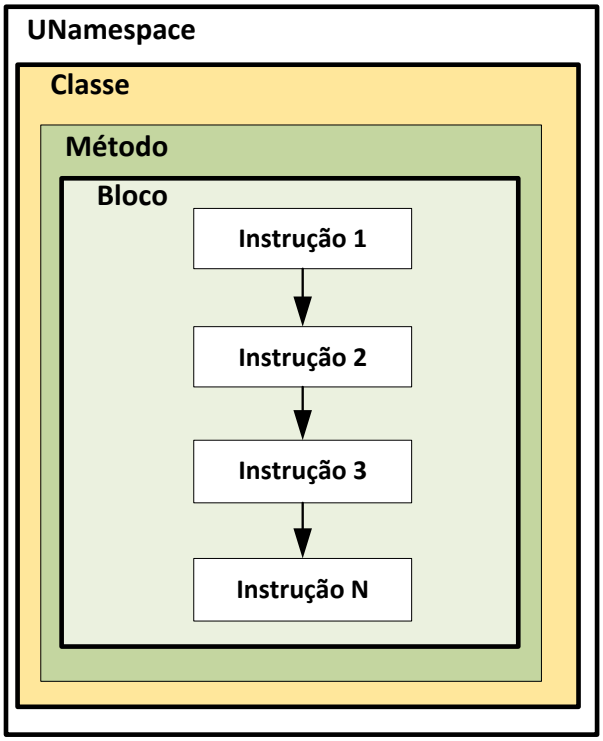


Figura B.1: Notação gráfica proposta para VisualAlf.

A Figura B.2 apresenta a notação gráfica para três instruções. A Figura B.2a apresenta a notação gráfica utilizada para instrução *for*. A instrução *for* possui três

instruções padrões: inicializar a variável (i = 0), condição de parada (i < 10) e controle do laço (i++), além de um bloco que poderá conter um novo conjunto de instruções. A Figura B.2b apresenta a notação gráfica utilizada para instrução *if*. A instrução *if* possui apenas uma instrução padrão (condição de entrada) e um bloco de instruções que devem ser executadas, se a condição de entrada for verdadeira.

Finalmente, a Figura B.2c apresenta a notação gráfica para instrução *Return*. Essa instrução não possui instruções padrões e tem o papel de retornar um resultado que pode ser representado por uma variável, uma expressão, um valor literal, ou seja, qualquer classe filha da classe *Result*.

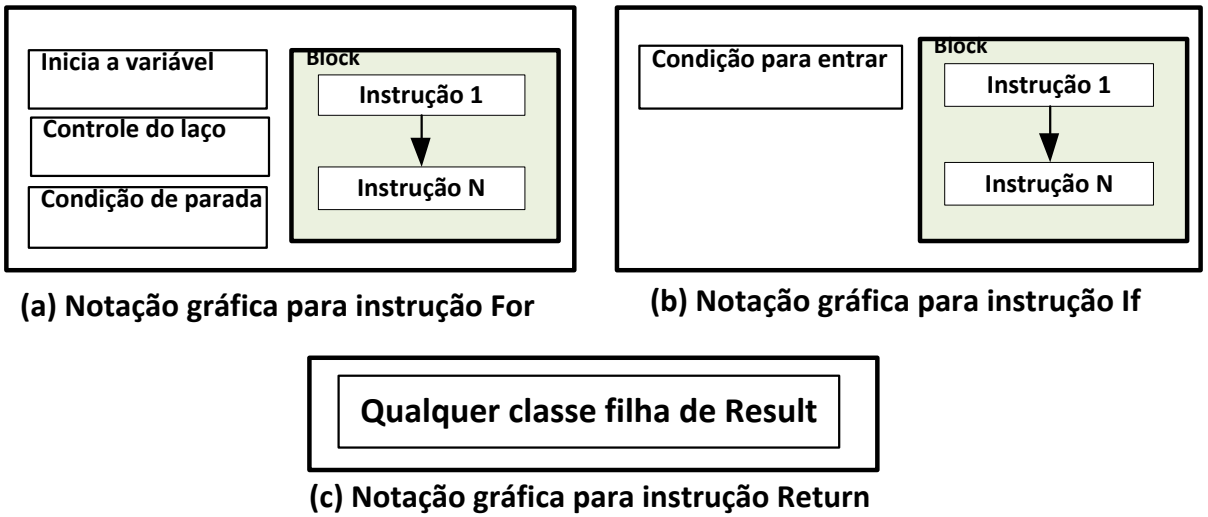


Figura B.2: Notação gráfica proposta para as instruções *For*, *If* e *Return*.

B.2 Exemplo Ilustrativo

O exemplo ilustrativo equação do 2° grau será apresentado nessa seção para demonstrar o uso da notação gráfica. Este exemplo ilustrativo foi escolhido por ser simples, de fácil compreensão e por ter baixa complexidade, para que o foco se concentre na criação do PIM, utilizando a notação gráfica.

Uma equação do 2° grau é qualquer equação com a forma $ax^2 + bx + c$ onde "x" representa um valor desconhecido e "a", "b" e "c" são os coeficientes da equação. Uma equação do 2° grau com coeficientes reais tem duas soluções, chamadas raízes. A fórmula da equação para obtenção das raízes é $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$.

O PIM para equação do 2° grau foi criado usando o VisualAlf Editor e é composto de três classes: *Coefficient*, *Result* e *QuadraticEquation*. A classe *Coefficient* contém os coeficientes da equação ("a", "b" e "c"), a classe *Result* armazenará os valores das raízes ("x1" e "x2") e a classe *QuadraticEquation* resolve a equação. A Figura B.3 apresenta um fragmento da classe *Result* modelada, usando a ferramenta VisualAlf Editor. Nessa figura, são destacados os métodos *getX1* e *setX1* e seus respectivos comportamentos, ou seja, a visão completa da lógica de negócio no modelo.

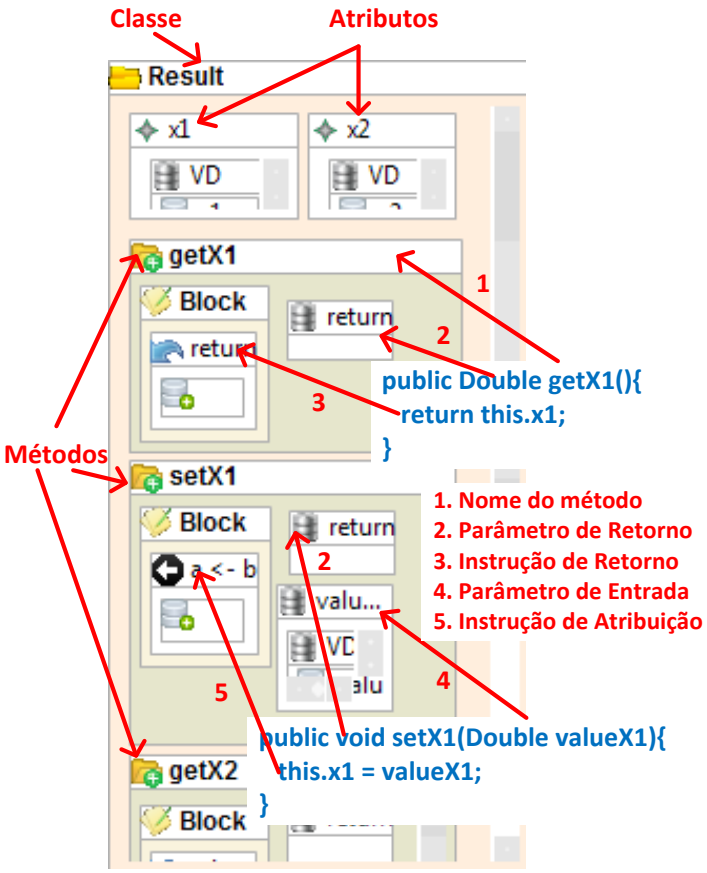


Figura B.3: Classe Result criada usando o plug-in VisualAlf Editor.

Enquanto na Figura B.4, pode-se ver o detalhamento do método *calculate* da classe *QuadraticEquation* também modelado, usando a ferramenta VisualAlf Editor. Esse método calcula o valor das raízes da equação, dessa forma tem como entrada uma instância da classe *Coefficient* e retorna uma instância da classe *Result*. Nessa figura, também, é possível observar a lógica de negócio de forma completa, para esse método, e que ela é composta de sete instruções que estão interligadas através de setas e que estas setas estão enumeradas. As setas determinam a ordem lógica com que as instruções devem ser executadas, e a enumeração foi colocada na figura somente para associar a representação visual com a representação textual em destaque.

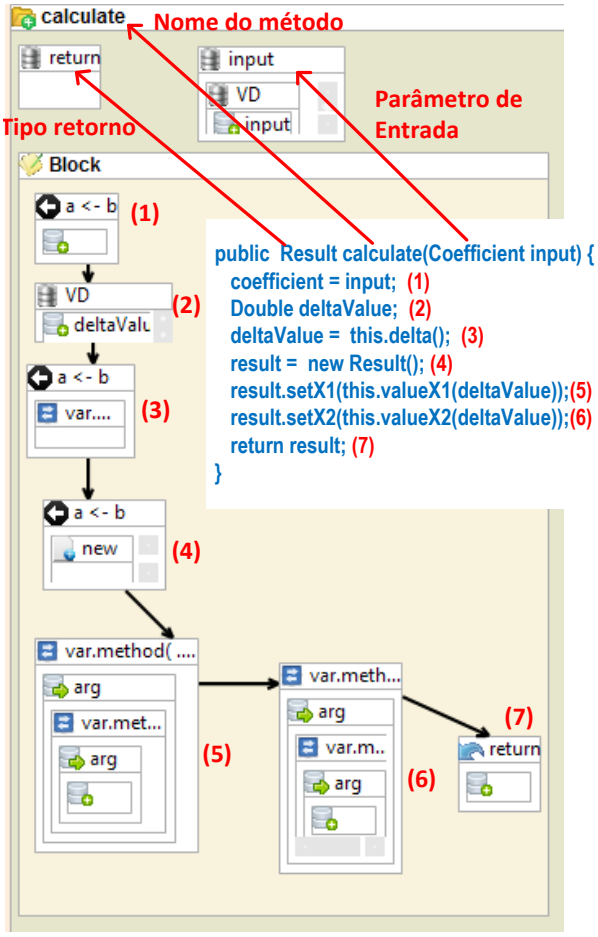


Figura B.4: Detalhamento do Método *calculate* da classe *QuadraticEquation* criado usando o plug-in VisualAlf Editor.

C Editores da Ferramenta (TF2BD)

O *framework* proposto tem modelos como artefato de entrada e saída para praticamente todas as tarefas. Assim, é pertinente que a ferramenta que implemente o *framework* ofereça formas de manipulação desses modelos. Sendo assim, *TF2BD* implementa um conjunto de editores que vão permitir que os usuários de domínio possam manusear os modelos exigidos por tal *framework*. Todos os editores foram gerados usando EMF e a ferramenta *GenModel*. A primeira foi escolhida, pois permite a geração de código baseado na definição de modelos, e a segunda foi utilizada, pois permite que o código gerado, a partir de EMF, seja transformado em um *plug-in* para *Eclipse*. Uma visão geral sobre como estender o *Eclipse*, assim como, de cada editor criado, será apresentada nas subseções subsequentes.

C.1 Estendendo Eclipse com EMF

O processo de extensão do *Eclipse* através de EMF é baseado na utilização de modelos de dados estruturados. Seu propósito é, a partir da especificação de modelos, prover ferramentas e suporte em tempo de execução para produzir um conjunto de classes Java para o modelo [95].

A primeira tarefa a ser realizada é criar a especificação do modelo, ou seja, metamodelo para o domínio desejado. Essa especificação deve usar a metalinguagem *Ecore* e, preferencialmente deve ser criada dentro de um projeto de modelagem (*modeling project*). Uma das formas de criar um arquivo *Ecore* é clicar com o botão direito do mouse na pasta "*model*" existente no projeto e seguir os passos: *"New"* -> *"Other..."* -> *"Ecore Model"* em seguida, siga as instruções e pressione *next* em cada etapa até finalizar;

O próximo passo é usar a ferramenta *GenModel* para gerar as classes definidas na especificação do modelo. Para realizar essa tarefa, a primeira coisa a fazer é criar um arquivo "*Generator Model*", da seguinte maneira: clique com o botão direito

do mouse na pasta *"model"* e siga os seguintes passos: *"New"* -> *"Other..."* -> *"EMF Generator Model"*.

O arquivo *"Generator Model"* possui quatro opções de geração de *plug-ins*:

- **Model:** Armazena todas as entidades, pacotes e fábricas para criar instâncias do modelo;
- **Edit:** Disponibiliza provedores para exibir um modelo em uma interface gráfica com usuário;
- **Editor:** Cria um editor que permite a criação e edição de instâncias de um modelo;
- **Teste:** Oferece *templates* para escrever testes para um modelo.

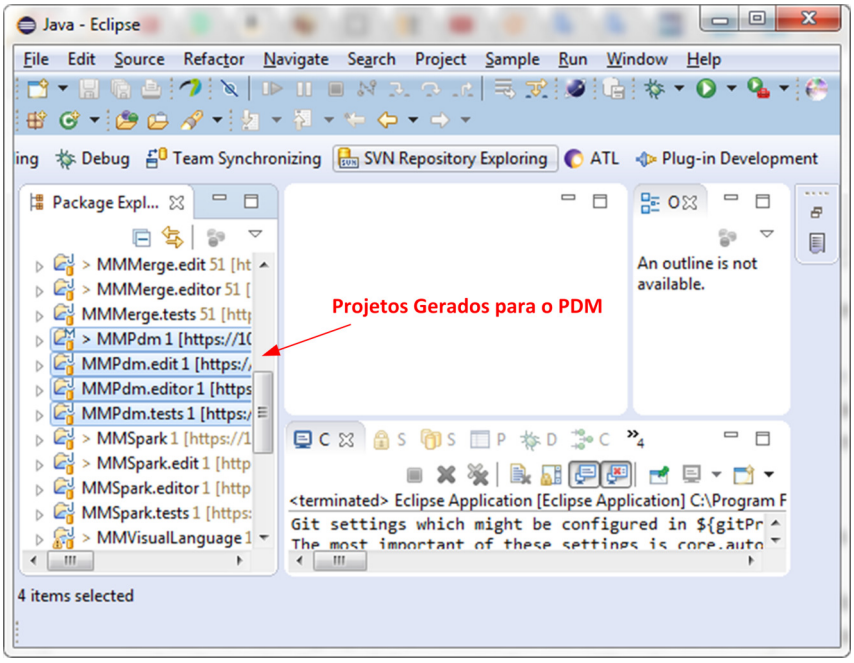


Figura C.1: Projetos gerados pelo *Generator Model* com base no metamodelo do PDM.

Cada uma dessas opções de geração dão origem a um projeto, por exemplo, pode ser visualizado em destaque na Figura C.1 os quatro projetos para o editor do PDM.

C.2 PIMEditor

O *PIMEditor* é um *plug-in* que permite que o modelo do PIM seja criado e editado com base na extensão proposta para o metamodelo de UML. Nesse editor, as entidades do modelo serão criadas, através do formato de árvore, que é a representação gráfica padrão utilizada pela EMF. Adequações nas classes, que geram o editor, foram realizadas para melhorar a representação visual das entidades no modelo, por exemplo, os ícones e *labels* foram estilizados.

As entidades são criadas clicando com o botão direito em cima de uma entidade qualquer do modelo e acessando a opção *"New Child"* do *PopupMenu*. As opções apareceram conforme as associações definidas no metamodelo. Por exemplo, se o clique for em cima da entidade *UMethod* as opções que aparecerão no *PopupMenu* serão as entidades do tipo *UParameter* e do tipo *Block*.

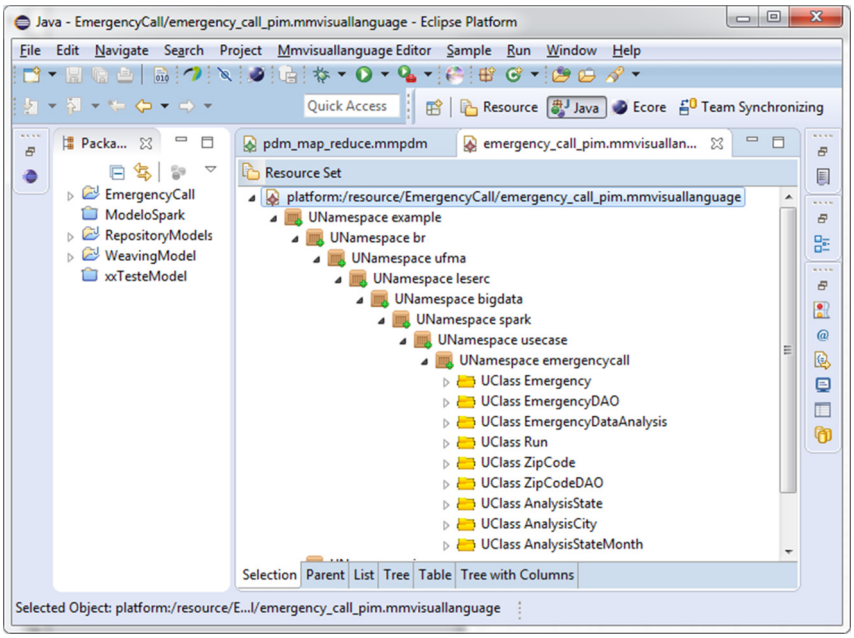


Figura C.2: Exemplo de um PIM criado usando o *plug-in PIMEditor*.

A Figura C.2 apresenta um PIM modelado usando o *plug-in* criado. Nessa figura, pode-se observar uma sequência de *UNamespace* e um conjunto de *UClass*.

C.3 VisualAlfEditor

O *VisualAlfEditor* é um *plug-in* alternativo para criação e edição do PIM. Assim como o *PIMEditor*, esse editor foi construído com base na extensão proposta

para o metamodelo de UML. O *VisualAlfEditor* implementa a notação gráfica proposta na Seção 4.4 e foi criado para que a forma gráfica e intuitiva característica de UML permaneça, mesmo após a extensão realizada. A ferramenta utilizada para criação desse *plug-in* foi o GMF.

A criação das entidades do modelo ocorre com um simples arrastar e soltar, e segue as mesmas regras estipuladas no metamodelo. Por exemplo, a representação gráfica de *UMethod* só poderá conter representações gráficas das entidades *UParameter* e *Block*.

A Figura C.3 apresenta um PIM criado, usando *VisualAlfEditor*. Nessa figura, pode-se visualizar o *namespace* "example" que contém a classe "Teste". A classe "Teste" contém um método "somar", que possui três parâmetros e um bloco de instruções que retorna a soma dos mesmos.

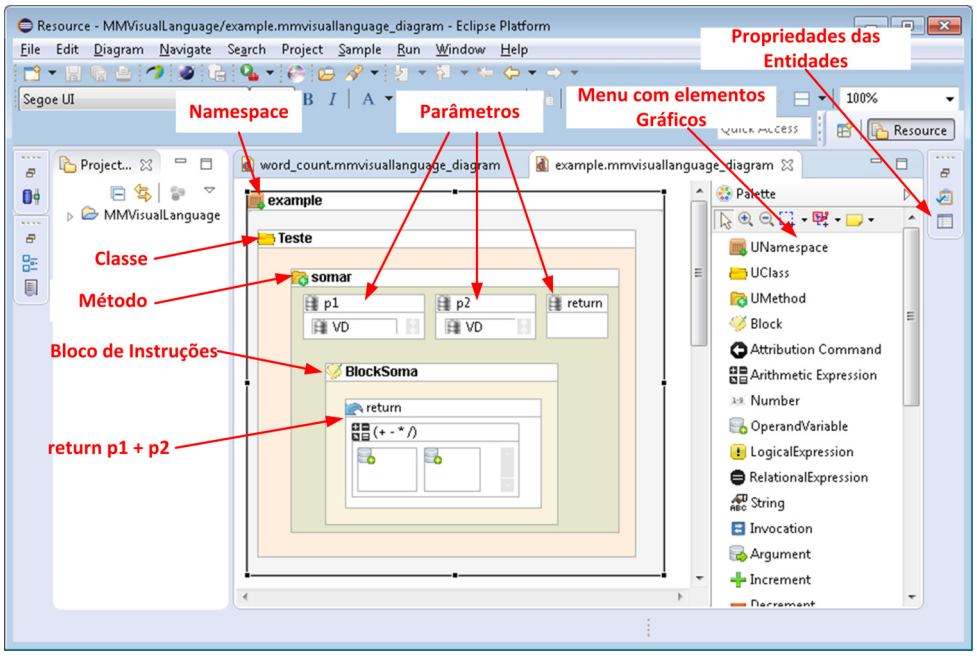


Figura C.3: Exemplo de um PIM criado usando o *plug-in VisualAlfEditor*.

C.4 PDMEditor

O *PDMEditor* é um *plug-in* que permite que o PDM seja criado e editado com base no metamodelo proposto para descrever *MapReduce* de *Big Data*. Nesse editor, assim como no *PIMEditor*, as entidades do modelo serão criadas, através do formato de árvore, que é a representação gráfica padrão utilizada pela EMF. Nesse editor, também

foram feitas adequações nas classes, que geram o editor para melhorar a representação visual das entidades no modelo, por exemplo, os ícones e *labels* foram estilizados.

A forma como as entidades são criadas ocorrem da mesma forma descrita na Seção C.2. A Figura C.4 apresenta o PDM modelado usando o *plug-in* criado.

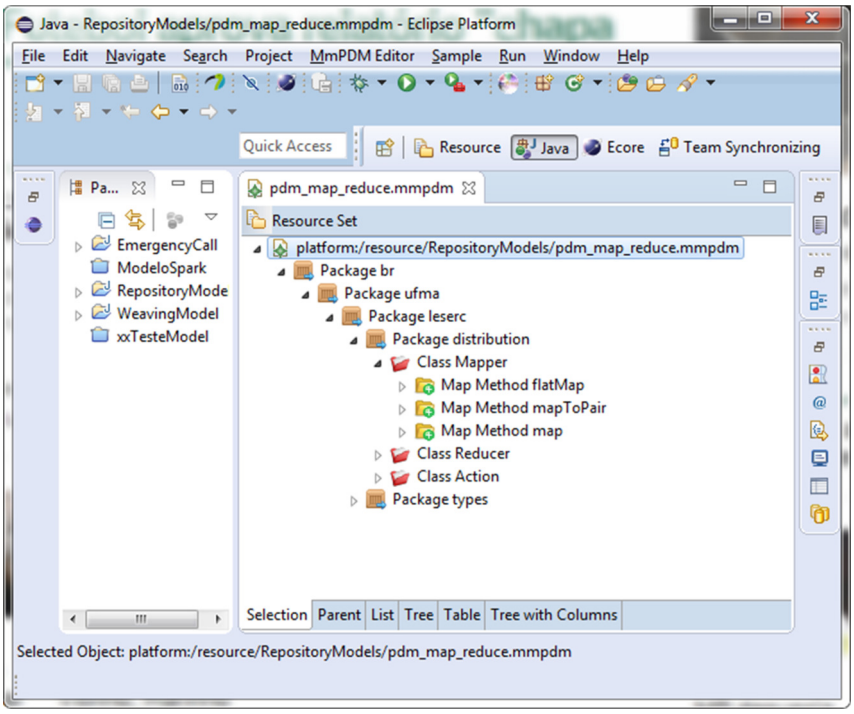


Figura C.4: Exemplo do PDM criado usando o *plug-in PDMEditor*.

Esse editor será usado eventualmente para criação do PDM, pois quando o PDM é criado ele pode ser utilizado diversas vezes. O editor foi criado apenas para quando houvesse necessidade de manutenção ou evolução do PDM, por exemplo, quando ocorrerem a criação de novos comportamentos de *Map* e *Reduce*.

C.5 WeavingEditor

Esse editor é dividido em três seções para facilitar a criação do modelo de *weaving*. A primeira seção, da esquerda para direita, contém o PIM; a segunda, no centro, conterá o modelo de *weaving*, que deve ser construído pelo usuário; e a terceira conterá o PDM. Os modelos colocados à esquerda e à direita do modelo do *weaving* são apenas para facilitar a visualização dos elementos que deverão ser entrelaçados pelo usuário.

A Figura C.5 apresenta um *screenshot* do *WeavingEditor*, desenvolvido como um *plug-in* para *Eclipse* IDE, desenvolvido com base no trabalho de Fabro et al. [34]. Nessa figura, pode-se observar o modelo de *weaving* criado para o problema *WordCount*, onde o PIM se encontra à esquerda; o modelo de *weaving*, se encontra no centro; e o PDM para *MapReduce de Big Data*, à direita. O modelo de *weaving* usa *handlers* para gerenciar os elementos dos modelos da esquerda e da direita, o que permite que o entrelaçamento possa acontecer.

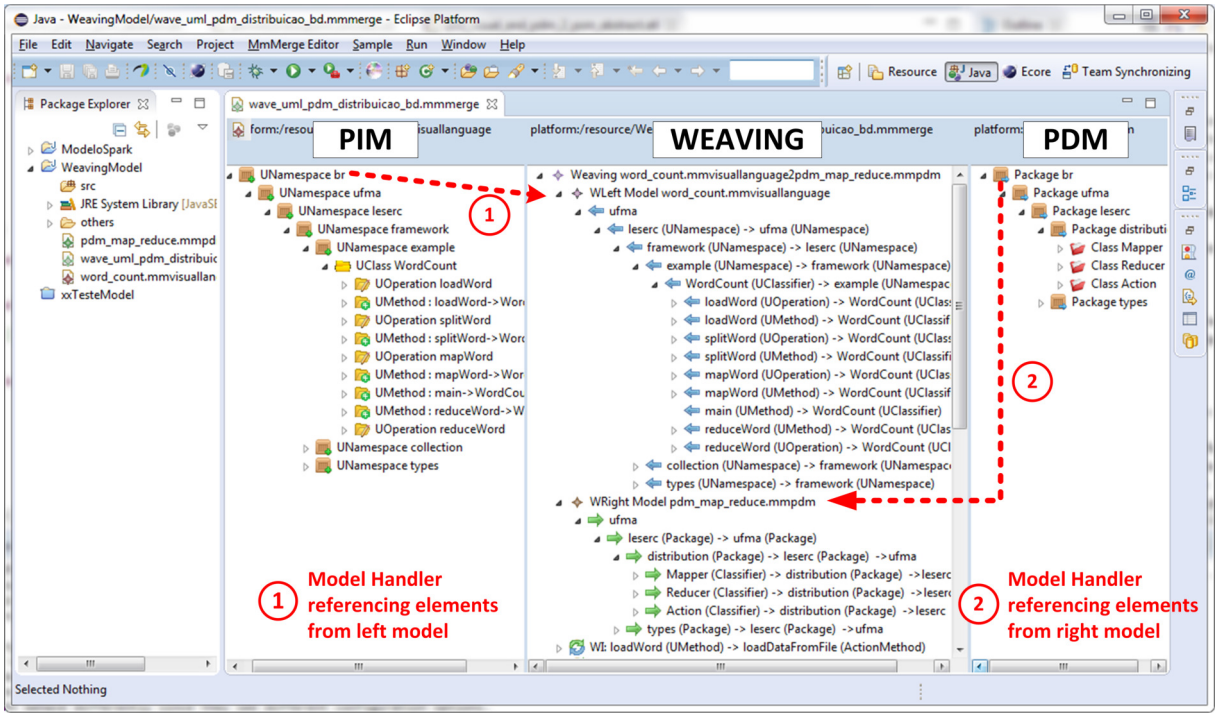


Figura C.5: Exemplo de Weaving criado usando o *plug-in WeavingEditor*.

C.6 PSMAbstratoEditor, PSMConcretoEditor e APIMapReduce Editor

Os editores *PSMAbstratoEditor* e *PSMConcretoEditor* manipulam o PSM abstrato e o PSM concreto, respectivamente. Esses editores são criados para eventualidades, por exemplo a visualização de resultados, visto que os modelos que eles manipulam são gerados automaticamente, através da execução das definições de transformação por um motor de transformação. O *PSMAbstratoEditor* é um *plug-in* criado, baseado no metamodelo proposto para o PSM abstrato, que será apresentado

na Seção 5.3.1, enquanto o *PSMConcretoEditor* é um *plug-in* criado, baseado no metamodelo da linguagem Java, que será apresentada na Seção 5.3.2.

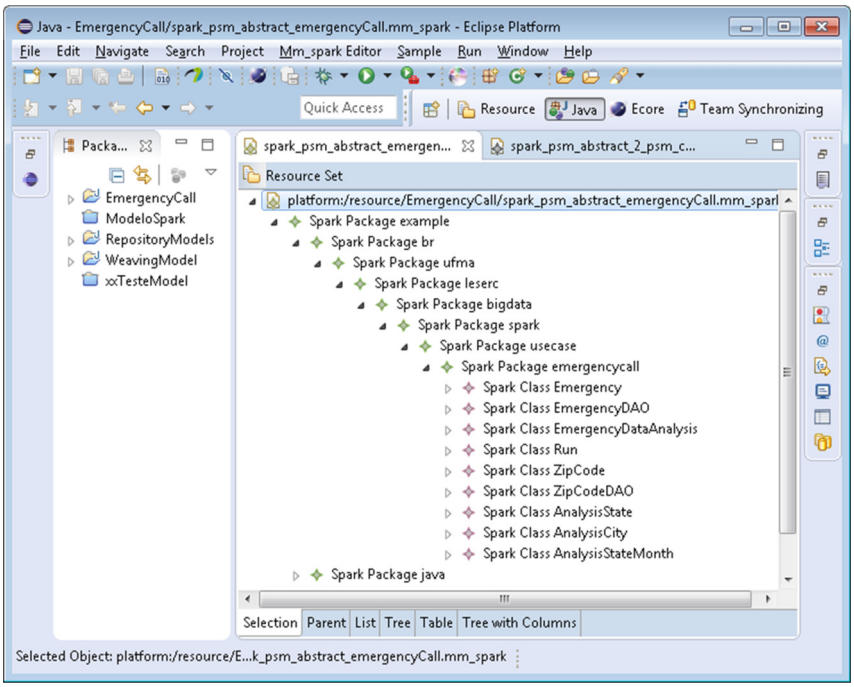


Figura C.6: Exemplo de PSM Abstrato sendo exibido usando o *plug-in PSMAbstratoEditor*.

O *APIMapReduceEditor* é utilizado para manipular o modelo da API, que implementa os conceitos de *MapReduce* concretamente. Como esse modelo deve ser gerado, utilizando o mesmo metamodelo do PSM concreto, logo, esse editor apenas faz referência ao mesmo *plug-in*, que é utilizado para manipular o PSM concreto.

A Figura C.6 e a Figura C.7 apresentam um PSM abstrato e um PSM concreto, respectivamente em seus editores.

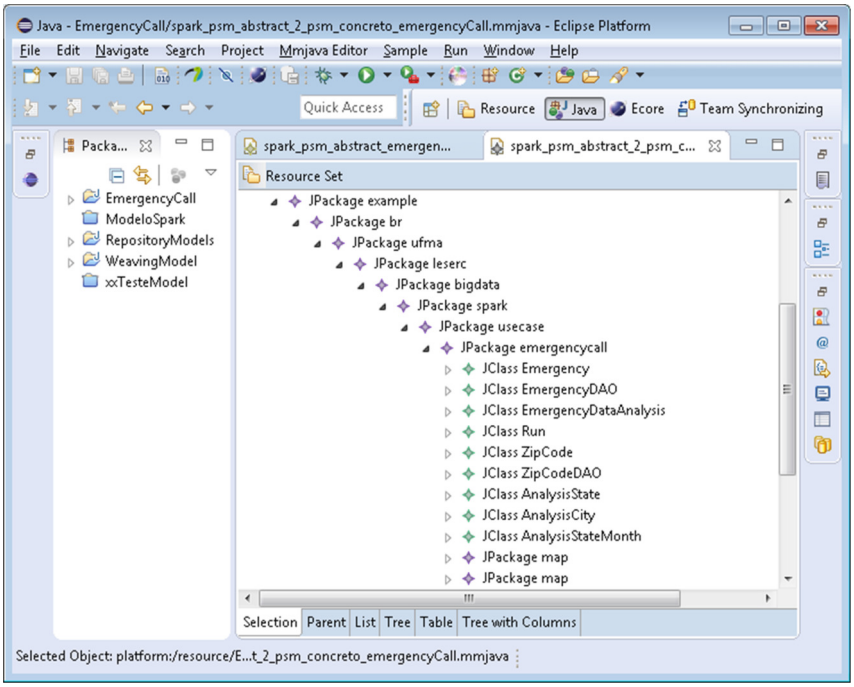


Figura C.7: Exemplo de PSM Concreto sendo exibido usando o *plug-in PSMConcretoEditor*.

D Questionário de Identificação do Participante

Prezado participante,

Esse estudo experimental está sendo realizado com o objetivo de analisar o suporte ao desenvolvimento de *software Big Data* com e sem o uso da TF2BD com o propósito de avaliar o efeito desses dois suportes a respeito da percepção da qualidade, do esforço e da eficácia no desenvolvimento, manutenção, evolução e documentação, do ponto de vista do pesquisador, em um contexto de analistas e técnicos em tecnologia da informação, alunos de graduação e pós-graduação em Ciência da Computação.

Todas as questões éticas relacionadas aos experimentos com seres humanos foram consideradas durante o planejamento desse estudo. Portanto, todos os participantes serão instruídos quanto ao objetivo do estudo, às implicações de sua participação, ao valor científico desse estudo empírico e à confiabilidade dos dados.

O questionário a seguir visa caracterizar os participantes para facilitar a compreensão dos resultados. Por favor, responda-o fielmente.

1. Dados Gerais

1.1 Nome Completo:

1.2 E-mail:

1.3 Qual seu nível de formação? (marque apenas uma opção)

( ) Graduando ( ) Graduado

( ) Mestrando ( ) Mestre

( ) Doutorando ( ) Doutor

1.4 Qual o nome da empresa/universidade que você atua?

1.5 Quanto tempo de experiência você possui na área que você atua?

2. Conhecimento de Programação

2.1 Como você definiria seus conhecimentos com relação à programação orientada a objetos?

( ) 0 - Nunca programei em nenhuma linguagem orientada a objetos;

( ) 1 - Minha experiência é básica. Possuo conhecimento teórico aplicado apenas no contexto acadêmico;

( ) 2 - Minha experiência é moderada. Possuo conhecimento teórico somado de experiências práticas individuais;

( ) 3 - Minha experiência é avançada. Possuo conhecimento teórico somado de experiências práticas em projetos na indústria.

2.2 Como você definiria a importância da documentação para o desenvolvimento, manutenção e evolução de um *software*?

( ) 0 - A documentação é perda de tempo, prefiro ir diretamente ao código;

( ) 1 - A documentação é importante para o desenvolvimento, depois deve ser descartada;

( ) 2 - A documentação só é importante para manter ou evoluir o código;

( ) 3 - A documentação é essencial para o desenvolvimento, manutenção e evolução de um *software*.

2.3 Com que frequência você atualiza a documentação do *software* que você constrói?

( ) 0 - Nunca é atualizada;

( ) 1 - É atualizada somente na fase de desenvolvimento;

( ) 2 - É atualizada somente quando há supervisão;

( ) 3 - Sempre que o código fonte é atualizado, a documentação também é atualizada.

3. Conhecimento em *MapReduce* para *Big Data*

3.1 Informe seu nível de conhecimento sobre o modelo de programação *MapReduce* de acordo com a listagem abaixo:

( ) 0 - Nunca utilizei;

( ) 1 - Minha experiência é básica. Possuo conhecimento teórico aplicado apenas no contexto acadêmico;

( ) 2 - Minha experiência é moderada. Possuo conhecimento teórico somado de experiências práticas individuais;

( ) 3 - Minha experiência é avançada. Possuo conhecimento teórico somado de

experiências práticas em projetos na indústria.

3.2 Informe seu nível de conhecimento sobre a API de *Spark* para linguagem Java de acordo com a listagem abaixo:

( ) 0 - Nunca utilizei;

( ) 1 - Minha experiência é básica. Possuo conhecimento teórico aplicado apenas no contexto acadêmico;

( ) 2 - Minha experiência é moderada. Possuo conhecimento teórico somado de experiências práticas individuais;

( ) 3 - Minha experiência é avançada. Possuo conhecimento teórico somado de experiências práticas em projetos na indústria.

3.3 Informe seu nível de conhecimento sobre o problema *WordCount* desenvolvido, utilizando a API de *Spark* para a linguagem Java de acordo com a listagem abaixo:

( ) 0 - Nunca ouvi falar;

( ) 1 - Minha experiência é básica. Ouvi falar, mas nunca me interessei pelo mesmo;

( ) 2 - Minha experiência é moderada. Ouvi falar e tive contato com a solução de forma teórica;

( ) 3 - Minha experiência é avançada. Ouvi falar e tive contato com a solução de forma prática.

3.4 Informe seu nível de conhecimento sobre o problema Chamada de Emergência (911) desenvolvido, utilizando a API de *Spark* para a linguagem Java de acordo com a listagem abaixo:

( ) 0 - Nunca ouvi falar;

( ) 1 - Minha experiência é básica. Ouvi falar, mas nunca me interessei pelo mesmo;

( ) 2 - Minha experiência é moderada. Ouvi falar e tive contato com a solução de forma teórica;

( ) 3 - Minha experiência é avançada. Ouvi falar e tive contato com a solução de forma prática.

4. Conhecimento em ferramentas aplicadas a *MapReduce* de *Big Data*

4.1 Qual o seu nível de conhecimento em relação a ferramentas/*frameworks* aplicados para o desenvolvimento, manutenção, evolução e documentação de *software* para *Big Data* que utilizem *MapReduce*?

( ) 0 - Desconheço ferramentas/*frameworks* aplicadas a *MapReduce*;

( ) 1 - Minha experiência é básica. Possuo conhecimento teórico aplicado apenas no contexto acadêmico;

( ) 2 - Minha experiência é moderada. Possuo conhecimento teórico somado de experiências práticas individuais;

( ) 3 - Minha experiência é avançada. Possuo conhecimento teórico somado de experiências práticas em projetos na indústria.

4.2 Qual o seu nível de conhecimento em relação ao *plug-in* para IDE Eclipse para desenvolvimento de software, utilizando a API *Spark* para Java?

( ) 0 - Desconheço completamente;

( ) 1 - Minha experiência é básica. Possuo conhecimento teórico aplicado apenas no contexto acadêmico;

( ) 2 - Minha experiência é moderada. Possuo conhecimento teórico somado de experiências práticas individuais;

( ) 3 - Minha experiência é avançada. Possuo conhecimento teórico somado de experiências práticas em projetos na indústria.

E Questionário Pós-Experimento

Prezado participante,

Para finalizar nosso estudo experimental precisamos coletar seu *feedback* sobre a condução do experimento e sobre a ferramenta avaliada. Todas as sugestões, críticas e comentários serão bem-vindos, portanto fique à vontade.

1. Que nota você atribuiria à execução do experimento?

() 0 - Mal executado. Os objetivos não estavam claros e o experimento foi mal conduzido;

() 1 - Razoável. Os objetivos estavam claros, entretanto ocorreram muitos problemas durante a execução do estudo;

() 2 - Bem executado. Os objetivos estavam claros, fui bem orientado e não ocorreram problemas durante a execução do estudo.

Comentários:

2. Que nota você atribui à forma com que a ferramenta TF2BD oferece suporte aos seguintes aspectos usando a legenda abaixo?

Suporte	Nota
Desenvolvimento	
Manutenção	
Evolução	
Documentação	

0 - Inadequado. A forma que a ferramenta oferece suporte é praticamente nula;

1 - Razoável. A ferramenta apresenta suporte, mas eu não usaria;

2 - Boa. A ferramenta apresenta suporte, mas não indicaria para um ambiente comercial;

3 - Excelente. A ferramenta apresenta suporte e eu a usaria para um ambiente comercial.

Comentários:

3. Na sua opinião, quais são os pontos fortes e fracos da ferramenta TF2BD?

4. Diante da necessidade da preservação do investimento, através do suporte ao desenvolvimento, manutenção, evolução e documentação de soluções computacionais para Big Data usando MapReduce, como você utilizaria a ferramenta TF2BD?

() 0 - Não utilizaria;

() 1 - Utilizaria apenas para projetos acadêmicos;

() 2 - Utilizaria em projetos acadêmicos e comerciais.

Comentários:

5. Qual seria sua percepção se a ferramenta TF2BD oferecesse outros mecanismos de interação para inserção da lógica de negócio, como por exemplo, inserção textual do código-fonte e posterior conversão para o modelo?

() 0 - Não utilizaria;

() 1 - Utilizaria apenas para projetos acadêmicos;

() 2 - Utilizaria em projetos acadêmicos e comerciais.

Comentários:

6. Fique à vontade para comentar quaisquer outros aspectos que não foram considerados.

Em nome de toda a equipe de pesquisadores envolvidos nesse projeto, gostaríamos de lhe agradecer pela contribuição concedida a esse estudo experimental. Acreditamos no poder que a ciência tem para transformar e facilitar a vida de todos.

Muito obrigado.