UNIFACS UNIVERSIDADE SALVADOR
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO EM SISTEMAS E COMPUTAÇÃO

FÁBIO COSTA SILVA

SOURCEMINER TRENDS: UMA INFRAESTRUTURA PARA ANÁLISE DE EVOLUÇÃO DE SOFTWARE BASEADA EM MÚLTIPLAS VISÕES

Salvador
2016

FÁBIO COSTA SILVA

SOURCEMINER TRENDS: UMA INFRAESTRUTURA PARA ANÁLISE DE EVOLUÇÃO DE SOFTWARE BASEADA EM MÚLTIPLAS VISÕES

Dissertação elaborada junto ao programa de Mestrado em Sistemas e Computação. Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação.

Orientador: Prof. Dr. Glauco de Figueiredo Carneiro.

Salvador
2016

FICHA CATALOGRÁFICA
Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate Internacional Universities

Silva, Fábio Costa

Sourceminer trends: uma infraestrutura para análise de evolução de software baseada em múltiplas visões./ Fábio Costa Silva.- Salvador: UNIFACS, 2016.

67 f.: il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação, Universidade Salvador – UNIFACS, Laureate International Universities como requisito parcial para obtenção do grau de Mestre.

Orientador Prof. Dr. Glauco de Figueiredo Carneiro.

1. Engenharia de software. 2. Evolução de Software. I. Carneiro, Glauco de Figueiredo, orient. II. Universidade Salvador – UNIFACS. III. Título

CDD: 005.1

TERMO DE APROVAÇÃO

FÁBIO COSTA SILVA

SOURCEMINER TRENDS: UMA INFRA-ESTRUTURA PARA ANÁLISE DE EVOLUÇÃO DE SOFTWARE BASEADA EM MÚLTIPLAS VISÕES

Dissertação aprovada como requisito final para obtenção do grau de Mestre em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate International Universities pela seguinte banca examinadora:

Glauco de Figueiredo Carneiro – Orientador _____
Doutor em Ciência da Computação pela Universidade Federal da Bahia - Ufba
UNIFACS Universidade Salvador, Laureate International Universities

Paulo Caetano da Silva _____
Ph.D. in Computer Science, Universidade Federal de Pernambuco - UFPE
UNIFACS Universidade Salvador, Laureate International Universities

Michel dos Santos Soares _____
Doutor em Computer Science, pela Delft University of Technology, TUDelft, Holand
Universidade Federal de Sergipe - UFS

Salvador, 21 de outubro de 2016.

Dedico este trabalho a minha família, amigos e a todos que me apoiaram ao longo dessa jornada.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus que iluminou o meu caminho durante esta difícil caminhada. Agradeço também a minha noiva, Talita, que de forma especial e carinhosa me deu força e coragem, me apoiando nos momentos de dificuldades. Agradeço também a grandiosa contribuição do meu irmão e amigo Fabrício, que esteve presente mesmo distante. Também sou eternamente grato aos meus pais, Jairo e Ananira, que sempre me guiaram para o caminho correto com palavras de sabedoria. Em particular sou grato ao meu orientador Dr. Glauco Carneiro, pelo constante apoio e incentivo.

“Enquanto houver vontade de lutar haverá
esperança de vencer”

(Santo Agostinho)

RESUMO

Analisar à dinâmica que concerne a evolução de um software permite adotar medidas para manter um programa útil. Ferramentas de visualização de dados de evolução são uma das linhas de pesquisa nessa direção. Elas têm o potencial de agregar grandes massas de dados, bem como permitir comparações, destacar elementos destoantes, dentre outros aspectos, numa interface de fácil leitura. No entanto, a heterogeneidade das fontes de dados de evolução, bem como a correlação entre elas representa um desafio. Ao mesmo tempo, as ferramentas de visualização disponíveis tendem a focar na representação de dados ao invés de focarem em atender determinados objetivos de compreensão que um usuário pode porventura almejar. Neste trabalho, é apresentado o Sourceminer Trends, uma ferramenta de análise de evolução que procura sanar estes problemas, numa abordagem orientada a objetivos. São apresentados modelos para caracterizar dados de evolução, de código-fonte e de estruturas visuais, bem como o mapeamento entre eles. Estes modelos, assim como o processo de transformação é apresentado por meio de exemplos de uso. Observou-se que a adoção de modelos permite flexibilizar o ambiente de análise e facilita a serialização dos dados obtidos. Como limitações, vale ressaltar que a utilização de modelos acarretou em problemas adicionais de desempenho.

Palavras-chave: Visualização de Informação. Evolução de Software. Engenharia de Software Experimental.

ABSTRACT

Analyzing the dynamics related to software evolution allow users and stakeholders to adopt measures to keep a program useful. Software evolution visualization tools are one of the research lines towards this direction. They have the potential to aggregate large amounts of data, as well as allowing comparisons, highlighting outliers, among other aspects, in an easy-to-read interface. However, the heterogeneity of evolutionary data sources as well as the correlation among them is challenging. At the same time, the available visualization tools tend to focus on data representation rather than focusing on meeting certain comprehension goals that a user might have. In this work, Sourceminer Trends is presented, an evolution analysis tool that seeks to solve these problems in an objective-oriented approach. Models are presented to characterize data evolution, source code and visual structures, as well as the mapping between them. These models, as well as the transformation process is presented through use examples. It was observed that the adoption of models make the analysis environment more flexible and facilitate the serialization of the retrieved data. As limitations, it is worth mentioning that the use of models has led to additional performance problems. In addition, the mapping between objectives and visual representations presented a greater level of coupling than expected.

Keywords: Information Visualization. Software Evolution. Experimental Software Engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIMV	Ambiente Interativo Baseado em Múltiplas Visões
API	Application Programming Interface
EMF	Eclipse Modeling Framework
GQM	Goal Question Metric
IDE	Integrated Development Environment
KDM	Knowledge Discovery Metamodel

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo aborda os conceitos iniciais desta dissertação, problemas a serem abordados, objetivos, metodologia de pesquisa adotada, contribuições, além da estrutura dos próximos capítulos.

1.1 MOTIVAÇÃO DESTE TRABALHO

Para manter sua utilidade, softwares devem ser constantemente modificados e aperfeiçoados, caso contrário correm o risco de tornarem-se obsoletos (EICK; GRAVES, *et al.*, 2001). O ciclo de gerações originária destas constantes adaptações é chamado de Evolução de Software (LEHMAN, 1980). Durante este ciclo é produzida uma série de informações oriundas de diferentes fontes, sendo os sistemas de controle de versão, em geral, a mais utilizada para este fim. Nestes sistemas é possível obter não apenas a exata alteração que foi efetuada num arquivo, mas também seu autor e o registro de quando a modificação foi confirmada. Artefatos como a documentação, os registros de defeitos e de execução, e-mails, o controle de atividades dos desenvolvedores dentre outros também pode ser utilizado para se obter informações sobre como um software evolui. *Pull requests* (GOUSIOS; PINZGER; DEURSEN, 2014), prática que permite submeter alterações em código para revisão e aprovação, ilustra como até mesmo o próprio processo de desenvolvimento está sujeito a melhorias. Assim, a adoção de ferramentas capazes de lidar com a heterogeneidade, complexidade e volume de dados históricos é fundamental para analisar a evolução de um software. Compreender como se dá toda essa dinâmica representa um passo em direção a identificação de problemas de processo e produto, redução de custos e melhoria de qualidade em projetos de software.

1.2 PROBLEMA A SER ABORDADO

Analisar e compreender como um software evolui não é trivial. Primeiramente, evolução de software deve ser caracterizada através do uso de dados de diferentes origens, tais como o código-fonte, registros de alterações e registros de auditoria. Agregar informações destas diferentes fontes de dados representa por si só um desafio, uma vez que frequentemente não há um relacionamento explícito entre elas. Mesmo com este desafio superado, ainda é

necessário identificar como tornar essa informação produzida legível e útil. Uma das linhas de pesquisa nesta direção é a visualização de evolução de software, um ramo do conhecimento que é uma especialização da visualização de informação. Ela se beneficia da nossa capacidade em processar informações mais facilmente por meio de imagens, elementos geométricos, animações e outros recursos visuais. Essa abordagem também é capaz de racionalizar informações obtidas de uma grande massa de dados num espaço reduzido.

Visualização de evolução de software vem sendo abordada na literatura de maneira predominantemente *ad-hoc*, no que tange a construção da visualização e a correlação entre as metáforas visuais adotadas e o que o usuário deseja obter de informação, seus objetivos de compreensão (NOVAES; TORRES, *et al.*, 2013). Promover um modelo ou estrutura de dados para representação das visões adotadas em ferramentas de visualização, bem como um processo que torne explícito o mapeamento de objetivos de compreensão para as propriedades dessas visões representa um passo nessa direção. O Sourceminer Trends, desenvolvido no presente trabalho, é um projeto que tem esta finalidade. Trata-se de uma ferramenta para extração, transformação e análise de dados de evolução de software, através de um ambiente interativo baseado em múltiplas visões (SILVA; CARNEIRO, 2012). As visões utilizadas, bem como suas propriedades, são explicitamente mapeadas a objetivos de compreensão, num processo baseado na metodologia Goal Question Metric (BASILI, 1992).

1.3 INFRAESTRUTURA PARA ANÁLISE DE EVOLUÇÃO DE SOFTWARE

Para realização deste trabalho, foi desenvolvida uma ferramenta que serve como infraestrutura para o estudo de evolução de software. A mesma fundamenta-se na utilização de estruturas para representação de dados de evolução, código e representações visuais, abordagem que a torna flexível e padronizada. Também faz parte do Sourceminer Trends módulos de cálculo de métricas e mapeamento de seus valores com representações visuais.

1.4 ESTRUTURA DESTA DISSERTAÇÃO

Os próximos capítulos seguem a seguinte estrutura:

Capítulo 2: Neste capítulo é apresentada a fundamentação teórica, no qual são abordados sistemas de controle de versão, programação baseada em modelos e tecnologias associadas.

São introduzidos os conceitos relacionados ao Eclipse EMF, tecnologia que serve como alicerce para infraestrutura;

Capítulo 3: Este capítulo descreve o processo metodológico e o ciclo da pesquisa para o desenvolvimento do estudo, cuja metodologia utilizada foi uma revisão sistemática e posterior caracterização de usos relatados na literatura;

Capítulo 4: Apresenta e discute a solução proposta para a ferramenta desenvolvida neste trabalho;

Capítulo 5: Este capítulo discorre acerca do processo de mapeamento entre atributos reais e visuais;

Capítulo 6: Este capítulo descreve um cenário no qual o Sourceminer Trends é aplicado;

Capítulo 7: O último capítulo discute os resultados obtidos e contribuições futuras.

1.5 ESTUDOS REALIZADOS COM A INFRAESTRUTURA

Os resultados obtidos nesta dissertação são descritos a seguir: (a) Proposta de modelos para representar evolução de software a estruturas visuais (b) Mapeamento explícito representar informações em metáforas visuais de acordo com sua natureza.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda as referências utilizadas neste projeto.

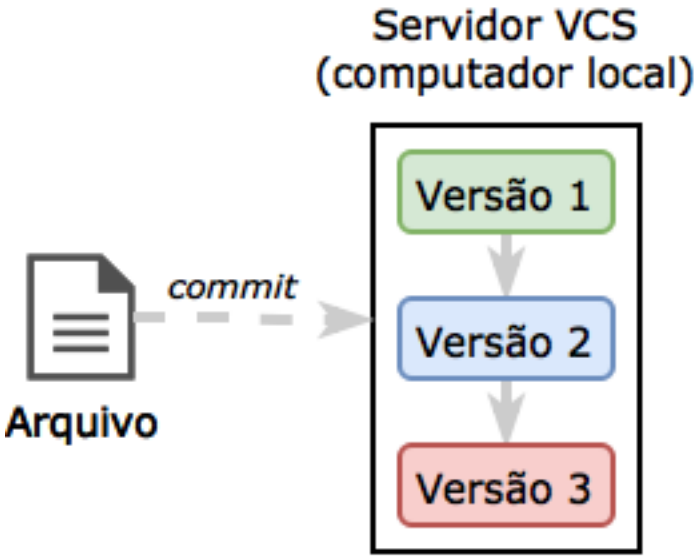
2.1 EVOLUÇÃO DE SOFTWARE

O conceito de evolução de software está atrelado a noção de tempo, refletindo a possibilidade de verificar como o software está organizado num determinado momento A e o que fazer para que ele atinja a configuração desejada num momento B. A complexidade desses ciclos evolutivos tende a ser proporcional ao tempo de resposta ao aparecimento de novos requisitos (LEHMAN, 1980). Medidas podem ser adotadas para mitigar esta diferença, como a preparação de uma extensiva documentação, treinamentos e outras formas de transferência de conhecimento, projeto para mudança e a utilização de ferramentas de gerenciamento e análise de evolução (DIEHL, 2007). Para o gerenciamento, sistemas de controle de versão são largamente utilizados. Eles são descritos a seguir.

2.1.1 Sistemas de Controle de Versão

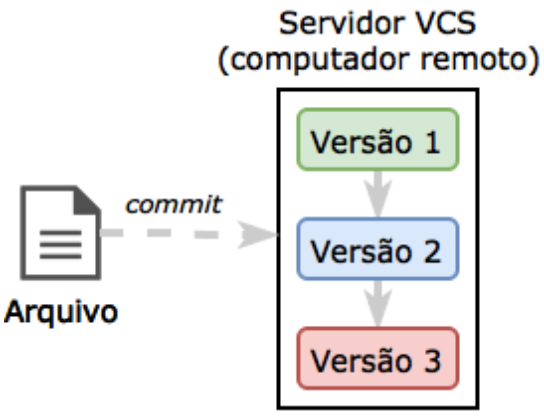
A maneira mais rudimentar de gerenciar alterações em arquivos é simplesmente criar copias dos mesmos. No entanto, manter o registro de alterações organizado se torna um problema nesta abordagem. Este papel pode ser realizado manualmente ou através de ferramentas especializadas chamadas de Sistemas de Controle de Versão (VCS). Além de manter um registro das modificações, estes sistemas armazenam metadados como o autor e a data em que ela ocorreu. Tradicionalmente, os sistemas de controle de versão são classificados em três categorias: local, centralizado e distribuído. No modelo local, ilustrado na figura 1, todas as alterações são registradas num banco de dados armazenado no computador do usuário. Este é o modelo mais simples e normalmente adotado para o uso pessoal. Softwares como o RCS (TICHY, 1985) e o Source Code Control System (ROCHKIND, 1975) são exemplos de VCSs locais.

Figura 1 - Sistema de Controle de Versão Local



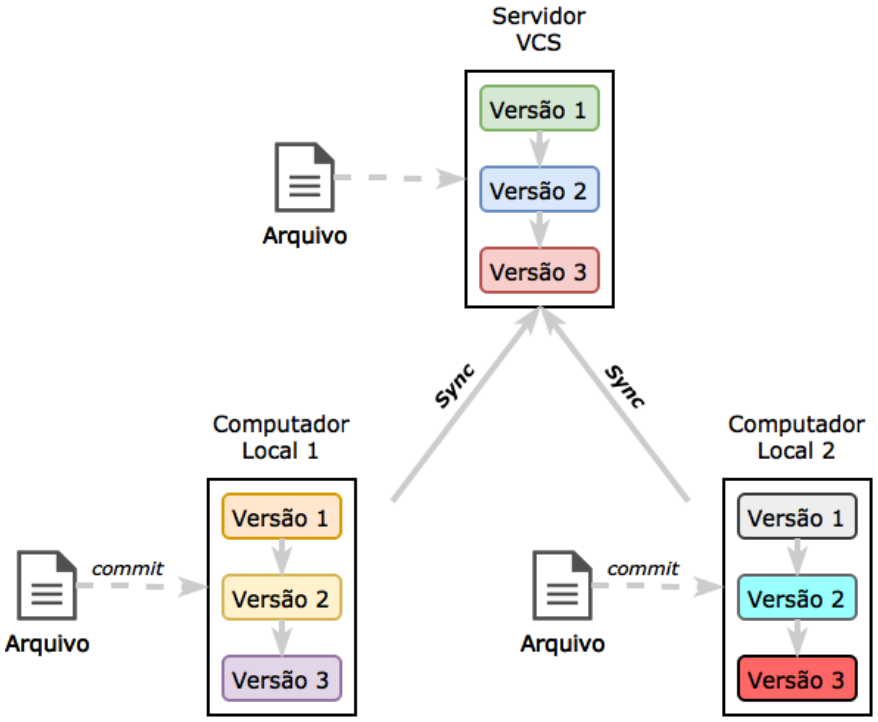
Há também os sistemas de controle de versão centralizados - figura 2 -, onde todos os registros são armazenados num servidor central. Cabe ao cliente enviar as alterações e obter as atualizações a partir deste computador. Este comportamento é vantajoso pois permite que diferentes pessoas editem o mesmo arquivo. Além disso, é uma abordagem mais segura que a anterior, já que os dados estão armazenados em um local externo que, em tese, está mais preparado para lidar com falhas. Por outro lado, depende de conexão remota, o que pode se tornar um problema quando as opções de conectividade são restritas ou inviáveis. O sistema de controle de versão centralizado mais popular no momento da escrita deste trabalho é o Subversion (APACHE SUBVERSION, 2016).

Figura 2 - Sistema de Controle de Versão Centralizado



Por fim, o modelo distribuído, que funciona como uma mistura dos dois modelos citados anteriormente. Neste modo de funcionamento, cada computador envolvido mantém um repositório local, que pode ser original ou proveniente do espelhamento de um outro repositório. Quando espelhado, operações de sincronização adicionais são necessárias para manter ambos atualizados. A figura 3 exemplifica este cenário. Nela, o servidor VCS é mantém o repositório que serve como base para os interessados em gerenciar as versões de um determinado arquivo. **Computador Local 1** e **Computador Local 2** mantém seus próprios repositórios, completamente independentes entre si. Quando necessário, podem obter novos dados do repositório original ou enviar alterações para o mesmo. A grande vantagem deste modelo é permitir a continuidade do trabalho mesmo quando não há conectividade disponível.

Figura 3 – Sistema de Controle de Versão Distribuído

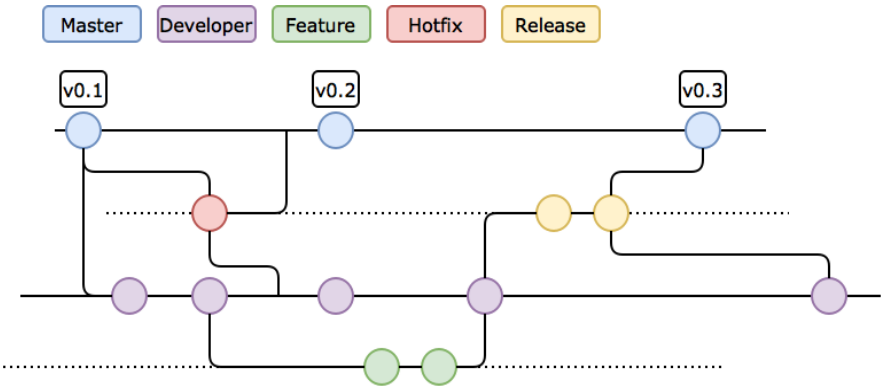


Projetos de software estão gradativamente movendo do modelo centralizado para o distribuído (ALWIS; SILLITO, 2009), principalmente devido a capacidade de operação *offline*. Como consequência, a sincronização com o servidor tende a ser realizada com menos frequência e mais complexa, quando comparada ao modelo centralizado tradicional. O principal representante desta categoria de VCS é o Git, detalhado em seguida.

2.1.2 Git

O Git (2016) surgiu para atender necessidades específicas do desenvolvimento do Kernel do Linux (LOELIGER; MCCULLOUGH, 2012). No momento de sua criação, os requisitos a serem atendidos se direcionavam a facilitar o desenvolvimento distribuído, com bom desempenho, disponibilizando repositórios completos para os desenvolvedores e um processo de desenvolvimento que favorece o uso de ramificações (*branches*). O Git Flow (DRIESSEN, 2010) é um dos modelos de *branching* propostos para uso com o Git. Seu fluxo é ilustrado na figura 4. No exemplo, o desenvolvedor inicial o ciclo copiando a ramificação inicial. A partir daí, podem ser criadas ramificações para funcionalidades específicas, correções urgentes e para a verificação de *pull requests*, técnica na qual o desenvolvedor submete suas alterações para serem integradas, caso aprovadas, ao código principal (*master*) (GOUSIOS; PINZGER ; DEURSEN, 2014).

Figura 4 – Git Flow



Fonte: Driessen (2010).

2.1.3 Visualização de Evolução de Software

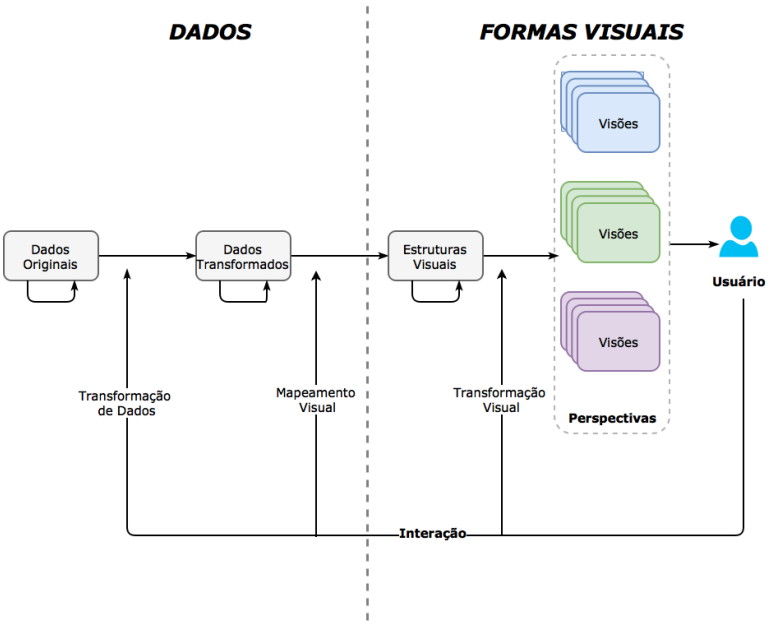
A grande quantidade informações produzidas durante o ciclo de vida de um software implica na adoção de técnicas de análise que simplifiquem a navegação, intercorrelação, abstração e, consequentemente, a compreensão destes dados. As ferramentas de visualização possuem a vantagem de explorar as capacidades do sistema cognitivo humano, aproveitando sua habilidade em reconhecer padrões visuais. Visualizações, quando comparadas a outras abordagens de análise de dados, possuem como vantagens a potencialização do uso da

memória, redução da necessidade de busca de informações, simplificação do reconhecimento de padrões e capacidade fornecida ao usuário de interagir com os dados utilizando mecanismos interação afins a sua percepção (CARD; MACKINLAY; SHNEIDERMAN, 1999). Ambientes mais sofisticados disponibilizam uma gama de representações visuais para que o usuário escolha a mais adequada para as suas necessidades. Estes são os Ambientes Interativos Baseados em Múltiplas Visões (AIMVs).

2.1.3.1 Ambientes Interativos Baseados em Múltiplas Visões

AIMVs permitem que diferentes visões sejam utilizadas de maneira coordenada, de modo a permitir a análise de um conjunto de dados em diferentes perspectivas e meios de interação (SILVA; CARNEIRO, 2012). A figura 5 apresenta o fluxo utilizado como modelo de referência para o desenvolvimento de um AIMV. Os dados originais são obtidos a partir de uma fonte primária, sendo transformados para adaptá-los em estruturas visuais. Finalmente, estas estruturas são utilizadas como insumos para as visões disponíveis na ferramenta de visualização.

Figura 5 – Modelo de Referência para AIMVs

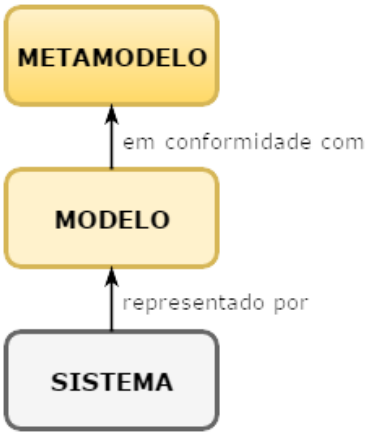


Fonte: Carneiro e Mendonça (2008).

2.2 ENGENHARIA BASEADA EM MODELOS (MDE)

Engenharia baseada em modelos (MDE) é um ramo do conhecimento que visa prover ferramentas que permitam expressar conceitos de domínio de maneira mais abstrata e efetiva. Um de seus fundamentos é o conceito de transformação de modelos. Modelos são representações simplificadas de uma certa realidade, de acordo com regras definidas por uma linguagem de modelagem (GENOVA, 2005), e que devem estar em conformidade com um metamodelo. Isso garante que diferentes modelos sejam consistentes entre si. Esse cenário é representado na figura 6. Ela ilustra a conformidade entre modelos.

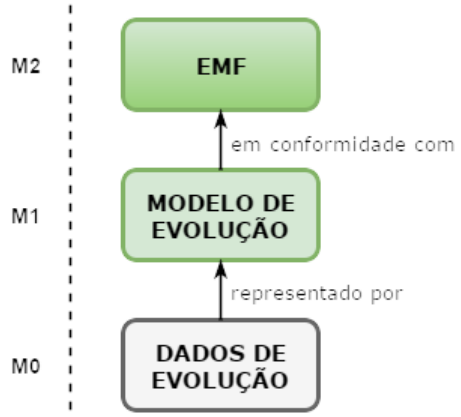
Figura 6 - Conformidade de modelos



Fonte: Adaptado de Genova (2005).

Já a figura 7 representa um exemplo de transformação. O Eclipse Modeling Framework (STEINBERG; BUDINSKY, *et al.*, 2008) é o metamodelo (M2) e serve como base para construção do modelo de evolução (M1), cuja instância depende de dados extraídos do Sourceminer Trends (M0).

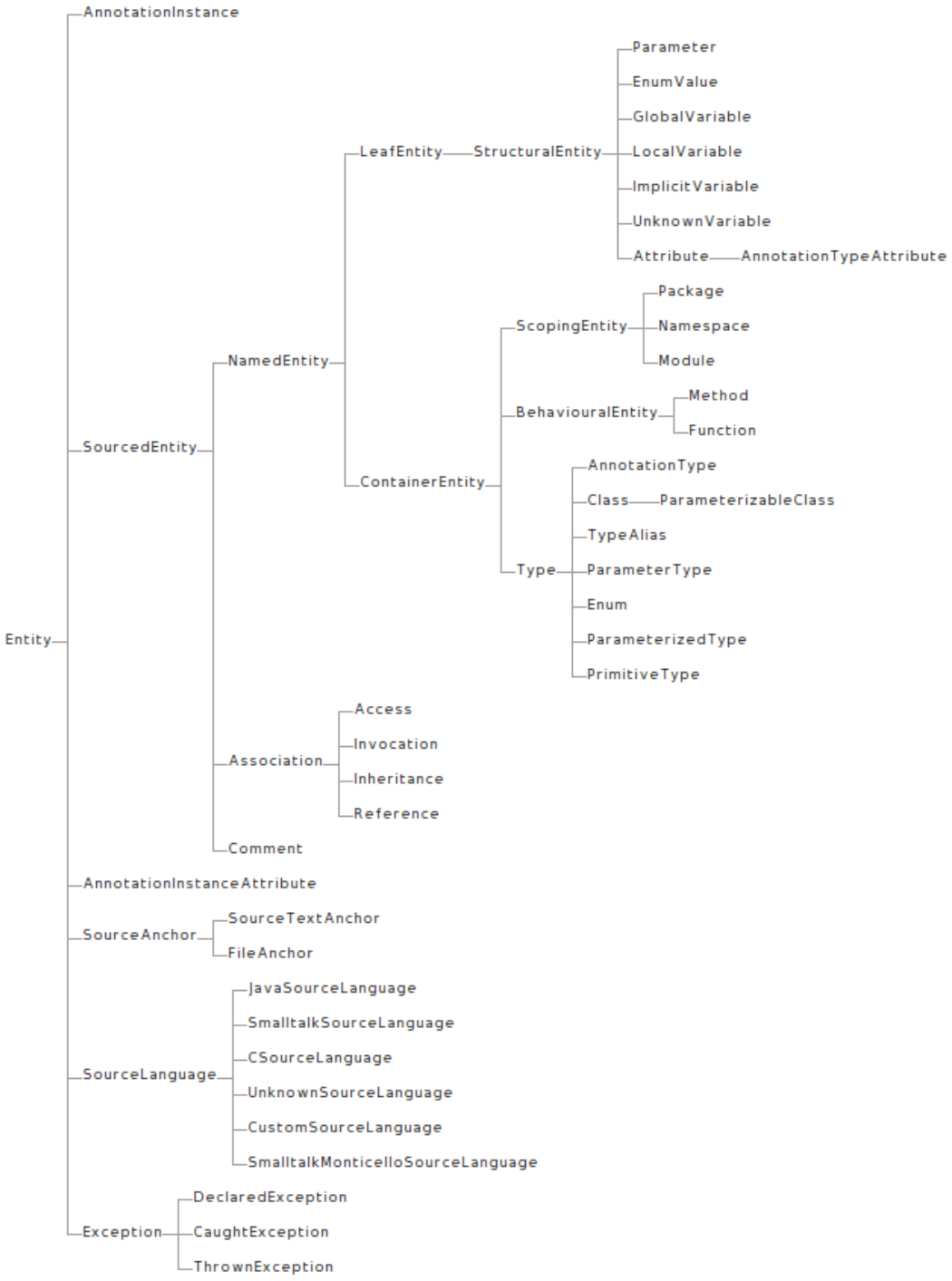
Figura 7 - Exemplo de Transformação de Modelos



2.2.1 FAMIX

O FAMIX é uma família de metamodelos para representação de diversos aspectos de sistemas de software (DUCASSE; ANQUETIL, *et al.*, 2011). Seus metamodelos são implementados na linguagem de programação Pharo (BLACK, 2010) e seu metamodelo principal pode representar variadas linguagens de programação orientadas a objetos de maneira uniforme. Uma versão simplificada do metamodelo principal pode é ilustrada na figura 8.

Figura 8 - Visão Geral do FAMIX

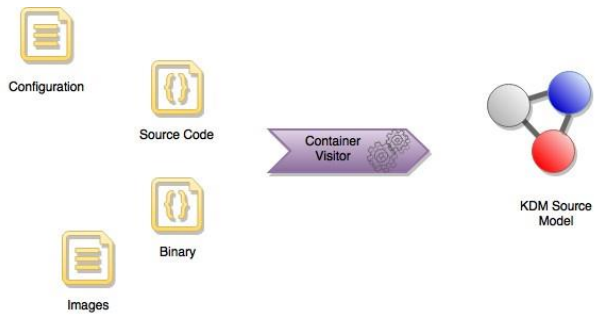


Fonte: Ducasse e Anquetil et al. (2011).

2.2.2 Knowledge Discovery Metamodel

Assim como o FAMIX, o Knowledge Discovery Metamodel (KDM) (PÉREZ-CASTILLO; DE GUZMAN; PIATTINI, 2011) é um metamodelo para representação de diferentes aspectos acerca de sistemas software. É uma especificação pública proposta pela organização Object Management Group (OMG) para a modernização de software, gerenciamento de portfólio de TI e de garantia de software. É um metamodelo para descoberta de conhecimento em software, que define um vocabulário comum de conhecimentos relacionados aos artefatos de engenharia de software, independentemente da linguagem de programação e implementação de tempo de execução da plataforma - uma lista de itens que uma ferramenta de mineração de software deve descobrir e uma ferramenta de análise de software pode utilizar. Esse padrão é projetado para permitir a integração do conhecimento entre as ferramentas, conforme expressa a figura 9.

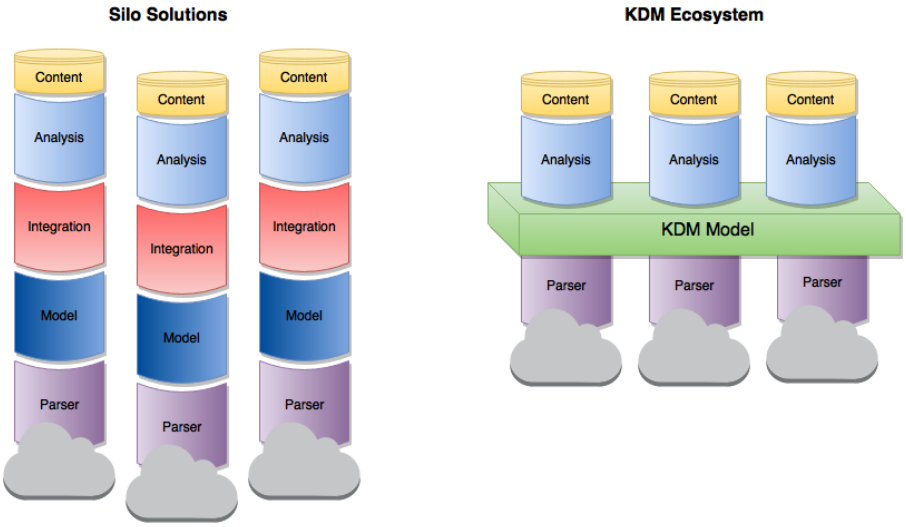
Figura 9 - Integração do conhecimento entre as ferramentas para o KDM



Fonte: KDM Source Discoverer (2016).

Um aspecto fundamental do KDM é que o mesmo lida com artefatos de naturezas distintas de maneira uniforme, conforme descrito na figura 10.

Figura 10 - Arqueologia Software: soluções silo e ecossistema KDM



Fonte: Pérez-Castillo, De Guzman e Piattini (2011).

2.2.3 Eclipse EMF

O Eclipse Modeling Framework é um projeto utilizado na IDE Eclipse como infraestrutura de modelagem e geração de código (STEINBERG; BUDINSKY, *et al.*, 2008) na qual modelo e código podem ser gerados um, a partir do outro, por representarem o mesmo dado. Um uso comum é a transformação do modelo de domínio de uma aplicação para um código-fonte Java correspondente. Ele possui raízes na especificação MOF (Meta Object Facility), criada pela OMG para definição de modelos serializáveis e portáveis entre diferentes aplicações. O EMF é baseado no metamodelo ecore, descrito na seção seguinte.

2.2.3.1 Ecore

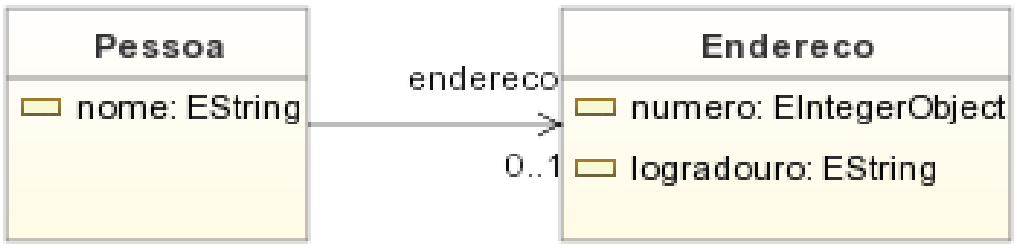
Modelos EMF são instâncias do metamodelo ecore. Como o mesmo é também descrito em ecore, pode ser chamado de meta-metamodelo. Ele se apropria de conceitos da UML, do XML, e do Java, possuindo elementos como os descritos na tabela 1:

Tabela 1 - Elementos Básicos do Metamodelo Ecore

ELEMENTO	DESCRIÇÃO
`EPackage`	Um pacote
`EClass`	Uma classe com zero ou mais atributos e zero ou mais referências
`EAttribute`	Um atributo com nome e tipo
`EReference`	O relacionamento entre duas classes
`EDataType`	O tipo de um atributo

XML Metadata Interchange (XMI) fornece uma representação textual para modelos ecore, a qual pode ser convertida para EMF e vice-versa. Um exemplo de modelo ecore é ilustrado na figura 11. O diagrama contém dois elementos `EClass`: `Pessoa` e `Endereco`, contidas no `EPackage` pessoas.

Figura 11 - Exemplo de modelo ecore representado em UML



Nota-se que os tipos dos atributos definidos nas classes são genéricos, o que abre a possibilidade de gerar código para diferentes linguagens de programação. No mesmo diagrama ainda há um relacionamento do tipo um para um, representado por um elemento `EReference`. Na listagem 1, o mesmo modelo é apresentado no formato XMI.

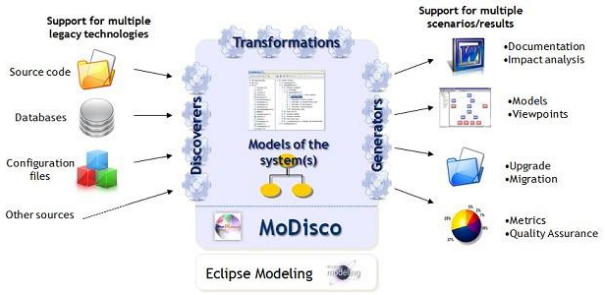
Listagem 1 - Exemplo de modelo ecore representado em XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="pessoas"
nsURI="http://www.example.org/asd" nsPrefix=" pessoas ">
  <eSubpackages name="pessoas">
    <eClassifiers xsi:type="ecore:EClass" name="Pessoa"
instanceClassName="Pessoa">
      <eStructuralFeatures xsi:type="ecore:EReference" name="endereco"
eType="#//pessoas/Endereco"/>
      <eStructuralFeatures xsi:type="ecore:EAttribute" name="nome"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    </eClassifiers>
    <eClassifiers xsi:type="ecore:EClass" name="Endereco">
      <eStructuralFeatures xsi:type="ecore:EAttribute" name="numero"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EIntegerObject"/>
      <eStructuralFeatures xsi:type="ecore:EAttribute" name="logradouro"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    </eClassifiers>
  </eSubpackages>
</ecore:EPackage>
```

2.2.3.2 Modisco

O Modisco é um framework criado para desenvolver ferramentas de engenharia reversa baseadas em modelos (BRUNELIERE; CABOT, *et al.*, 2010). Para o Modisco, modelos são considerados como artefatos a partir do qual partes de um software podem ser geradas automaticamente. Na Figura 12, são representados código-fonte, banco de dados e arquivos de configuração como artefatos, tendo como resultado da engenharia reversa documentação, modelos, pontos de vista, código migrado e métricas.

Figura 12 - Modelo de transformação do Modisco



Fonte: Bruneliere e Cabot *et al.* (2010).

A transformação de modelos no Modisco ocorre por meio de *discoverers*. Cada discoverer do Modisco responde a uma interface e pode ser chamado programaticamente. A listagem 2 ilustra como transformar um projeto Java num modelo UML.

Listagem 2 - Descoberta de modelo UML a partir de projeto Java

```
	DiscoverUmlModelWithBidirectionalAssociationsFromJavaProject discoverer = new
DiscoverUmlModelWithBidirectionalAssociationsFromJavaProject();
	discoverer.discoverElement(javaProject, monitor);
	Resource umlModel = discoverer.getTargetModel();
```

Existem diversos fluxos possíveis, sendo a transformação de código-fonte Java a mais trivial. A tabela 2 descreve alguns deles:

Tabela 2 - Opções do Modisco de Descoberta de Modelos

Classe de Descoberta	Descrição
`DiscoverJavaModelFromJavaProject`	Obtém o modelo Java a partir de um projeto Eclipse Java
`DiscoverKDMModelFromJavaModel`	Obtém o modelo KDM a partir de um modelo Java
`DiscoverKDMModelFromJavaProject`	Obtém o modelo KDM a partir de um projeto Eclipse Java
`DiscoverMethodCallsModelFromJavaModel`	Obtém as chamadas de métodos contidas num modelo Java
`DiscoverUmlModelFromJavaProject`	Obtém o modelo UML a partir de um projeto Eclipse Java
`DiscoverUmlModelFromKdmModel`	Obtém o modelo UML a partir de um modelo KDM

2.3 PERSISTÊNCIA DE METAMODELOS

Armazenar instâncias de modelos em banco de dados é um requisito comum em sistemas baseados em EMF. Quando essas instâncias se tornam excessivamente grandes, o consumo de memória pode ser impeditivo para serem processadas de uma vez. A persistência também visa aumentar a disponibilidade desses objetos ao copiá-los para um repositório e também visa disponibilizar diversas instâncias de um mesmo modelo para uso. Diversas são as abordagens adotadas na academia e na indústria. Exemplos incluem o Hawk (BARMPIS; KOLOVOS, 2013), que versiona modelos e permite a execução de consultas em OCL; o Neo4EMF (BENELALLAM; GOMEZ, *et al.*, 2014), framework para persistência de modelos EMF no banco de dados Neo4J; O EMFStore (KOEGEL ; HELMING, 2010), cuja finalidade é controlar a versão de instâncias de modelos, permitindo executar operações como *checkouts* e *commits*, comparação de versões; e o Connected Data Objects(CDO) (STEPPER, 2012), que assim como o EMFStore, é uma tecnologia desenvolvida pela fundação Eclipse para armazenamento de modelos. Ele facilita a utilização de bancos de dados tradicionais como Oracle e PostgreSQL para esse propósito, tendo pouco suporte, entretanto, para bancos de dados NoSQL.

2.3.1.1 Bancos de dados NoSQL

Bancos de dados NoSQL não tem uma definição clara, mas uma característica comum é que não são relacionais. Além disso costumam ser utilizados em ambientes que lidam com grandes volumes de dados (Big Data). Sua principal vantagem é que, ao contrário dos bancos de dados relacionais, eles são projetados para lidar com dados não estruturados tais como arquivos JSON, arquivos XML, e-mail e documentos de texto. Alguns desses bancos de dados, como o CouchDB (COUCHDB, 2015), podem funcionar em uma configuração distribuída, permitindo espalhar os dados em diversas máquinas, ao invés de manter todo o armazenamento em uma única máquina poderosa e cara. De modo geral, bancos de dados NoSQL podem ser classificados de acordo com a lista abaixo:

- Natureza do dado: modelos de objetos do mundo real, entidades altamente conectadas, cadeias de eventos;
- Natureza das operações: write once/read many, write intensive, insert/append only, OLTP, OLAP;
- Necessidade de negócio: escalabilidade, disponibilidade, desempenho, consistência,

flexibilidade, time-to-market;

- Custo total de propriedade: custo, complexidade, requisitos de hardware, suporte.

Geralmente bancos NoSQL processam dados mais rapidamente do que os bancos de dados relacionais principalmente por não implementar ou implementar apenas parcialmente as propriedades ACID (LEAVITT, 2010).

2.3.1.2 CouchDB

O CouchDB é um banco de dados simples de usar, que não possui conceitos como tabelas e cujas consultas são realizadas como funções `Map/Reduce`. Cada banco de dados é um conjunto independente de documentos. Metadados de documentos contém informações de revisão, o que torna possível mesclar quaisquer diferenças que possam ter ocorrido quando um servidor estava fora do ar. O CouchDB disponibiliza uma interface REST por padrão, o que auxilia na realização de interações sem a necessidade de qualquer tipo de conector. Entretanto, consultas são executadas através de `views`, que nada mais são de que uma forma programática de realizar consultas em documentos. A tabela 3 exibe um resumo das funcionalidades do CouchDB.

Tabela 3 - Conceitos do CouchDB

Conceito	**Exemplo**
document	/trends/project
attachment	/trends/attachment.txt
design doc	/trends/ design/releases
view	/trends/ design/releases/byDeveloper
changes	/trends/ changes

2.4 CONCLUSÃO

Este capítulo apresentou as bases teóricas do Sourceminer Trends. Foram abordados os conceitos de engenharia baseada em modelos, e EMF em particular, que é a tecnologia base para construção dos modelos propostos no trabalho. Também foi abordado o Modisco, plugin do Eclipse utilizado para transformação de código Java em modelos EMF. Por fim, foram apresentados conceitos de bancos de dados não relacionais. O CouchDB é o mecanismo de armazenamento utilizado neste trabalho. No próximo capítulo é discutida a metodologia de

pesquisa que norteou os estudos, desenvolvimento da ferramenta e exemplos de uso.

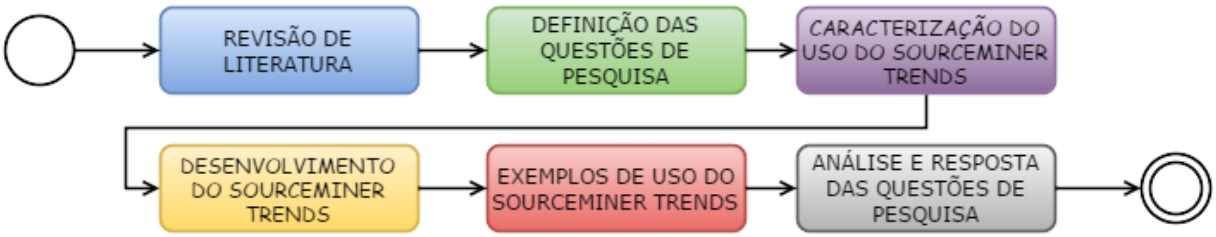
3 METODOLOGIA DE PESQUISA

Este capítulo descreve a metodologia de pesquisa utilizada nesta dissertação. As seções que se seguem apresentam como a pesquisa foi conduzida para atingir os objetivos propostos.

3.1 ETAPAS DA METODOLOGIA DE PESQUISA

As etapas que constituem a metodologia adotada nesta dissertação são ilustradas na figura 13. A revisão da literatura realizada na primeira etapa desta dissertação permitiu levantar estudos relevantes ao tema abordado. Essa etapa auxiliou na definição das questões de pesquisa que nortearam a condução deste trabalho. Também permitiu tomar decisões acerca de ferramentas e técnicas a serem adotadas. Em seguida, o Sourceminer Trends planejado e desenvolvido. É realizada a caracterização dessa ferramenta através de um exemplo de uso, no qual são extraídas estratégias de utilização a partir desses estudos. Por fim, as questões de pesquisa são analisadas e respondidas.

Figura 13 - Metodologia de Pesquisa



3.1.1 Revisão de Literatura

Os fundamentos teóricos para elaboração do projeto foram baseados numa revisão de literatura. A estratégia adotada foi a realização de sucessivas buscas em repositórios de trabalhos científicos e sistemas de busca, bem como a adição de artigos previamente conhecidos como chave, como por exemplo (NOVAES; TORRES, *et al.*, 2013). Este processo identificou trabalhos relevantes e o estado da arte nos campos de evolução de software, visualização de evolução de software e engenharia baseada em modelos.

3.1.2 Definição das Questões de Pesquisa

A definição das questões de pesquisa que norteiam esta dissertação foi realizada em abril de 2015. As questões de pesquisa foram consolidadas através da revisão da literatura, do objetivo e questões de pesquisa identificados anteriormente. O objetivo é caracterizar a compreensão de atributos de evolução de software através do uso de um ambiente interativo baseado em múltiplas visões. Os itens a seguir apresentam as Questões de Pesquisa (QPD) desta dissertação:

- Questão de Pesquisa da Dissertação 1 (QPD1): Qual proposta de metamodelo pode ser utilizada para instanciar dados de atributos de evolução para alimentar metáforas visuais?
- Questão de Pesquisa da Dissertação 2 (QPD2): Como identificar quais metáforas visuais em ambientes interativos baseados em múltiplas visões podem ser utilizadas para a visualização de atributos de evolução de software?
- Questão de Pesquisa da Dissertação 3 (QPD3): Como os objetivos de compreensão de atributos de evolução de software podem ser mapeados para atributos visuais?
- Questão de Pesquisa da Dissertação 4 (QPD4): Quais evidencias podem ser utilizadas para ilustrar a efetividade da compreensão de atributos de evolução de software no contexto de projetos de software reais?

3.1.3 Caracterização de uso do Sourceminer Trends

Nesta etapa foi realizada a definição de objetivos de compreensão e métricas a serem extraídas, com base na metodologia Goal Question Metric (GQM). A caracterização também envolveu a definição das representações visuais a serem utilizadas e a estratégia para criá-las de acordo com os dados obtidos através da GQM.

3.1.4 Desenvolvimento do Sourceminer Trends

O desenvolvimento do Sourceminer Trends compreendeu a definição arquitetural da ferramenta e a codificação da infraestrutura de extração de dados de evolução, da geração de modelos serializáveis para as diferentes versões do projeto analisado e a representação visual dos dados obtidos.

3.1.5 Exemplos de Uso

Os exemplos de uso foram elaborados de forma a verificar os conceitos propostos na etapa de caracterização de uso, bem como auxiliar nas respostas às questões de pesquisa.

3.1.6 Análise e Resposta das Questões de Pesquisa

Por fim, os resultados dos exemplos de uso foram analisados e as questões de pesquisa foram respondidas.

3.2 CONCLUSÃO

Este capítulo apresentou a metodologia utilizada no desenvolvimento desta dissertação, que compreendeu uma revisão de literatura para identificar os fundamentos teóricos do trabalho, a definição das questões de pesquisa, o planejamento e desenvolvimento do Sourceminer Trends, a caracterização de seu uso e análise dos resultados. O próximo capítulo entrará em detalhes no que tange ao projeto do Sourceminer Trends.

4 SOLUÇÃO PROPOSTA

4.1 INTRODUÇÃO

O Sourceminer Trends foi desenvolvido como uma aplicação de extração, armazenamento e análise de evolução de projetos de software, num ambiente interativo baseado em múltiplas visões (CARNEIRO; MENDONÇA, 2013). Os dados são estrategicamente extraídos em baixa granularidade, em nível de linhas alteradas, para maximizar a capacidade de análise. Ademais, a linguagem de programação subjacente do projeto examinado é abstraída em metamodelos disponibilizados pelo Modisco. Essa técnica objetiva permitir que projetos escritos em diferentes linguagens sejam estudados de maneira similar.

Tanto os dados históricos quanto a estrutura do código-fonte obtido de repositórios de código são representadas através de modelos EMF. O primeiro, de evolução, é uma proposta do Sourceminer Trends. Para o modelo de código, o Sourceminer Trends implementa representações em Java e KDM disponibilizadas pelo Modisco, porém o Java é utilizado como padrão. Na ferramenta, ao invés de representar a história de cada entidade separadamente (DUCASSE; GÎRBA; FAVRE, 2005), um projeto é tratado como um conjunto de versões em sequência, cada qual com seus commits. Essa representação é apresentada na figura 14. Este é um modelo inspirado na API JGit (JGit, 2016), a qual delimita commits como séries de alterações, definidas por regiões do código-fonte onde ocorreram as mudanças. Estão representados no diagrama as classes `Project`, `Release`, `Developer`, `Commit`, `Change` e `ChangeRegion`, sendo que as duas últimas possuem tipos específicos. Essas classes são caracterizadas por representar:

Project: o projeto importado, contendo apenas seu nome e repositório. Não está envolvida diretamente no mapeamento da evolução, mas é o ponto de partida para qualquer pesquisa;

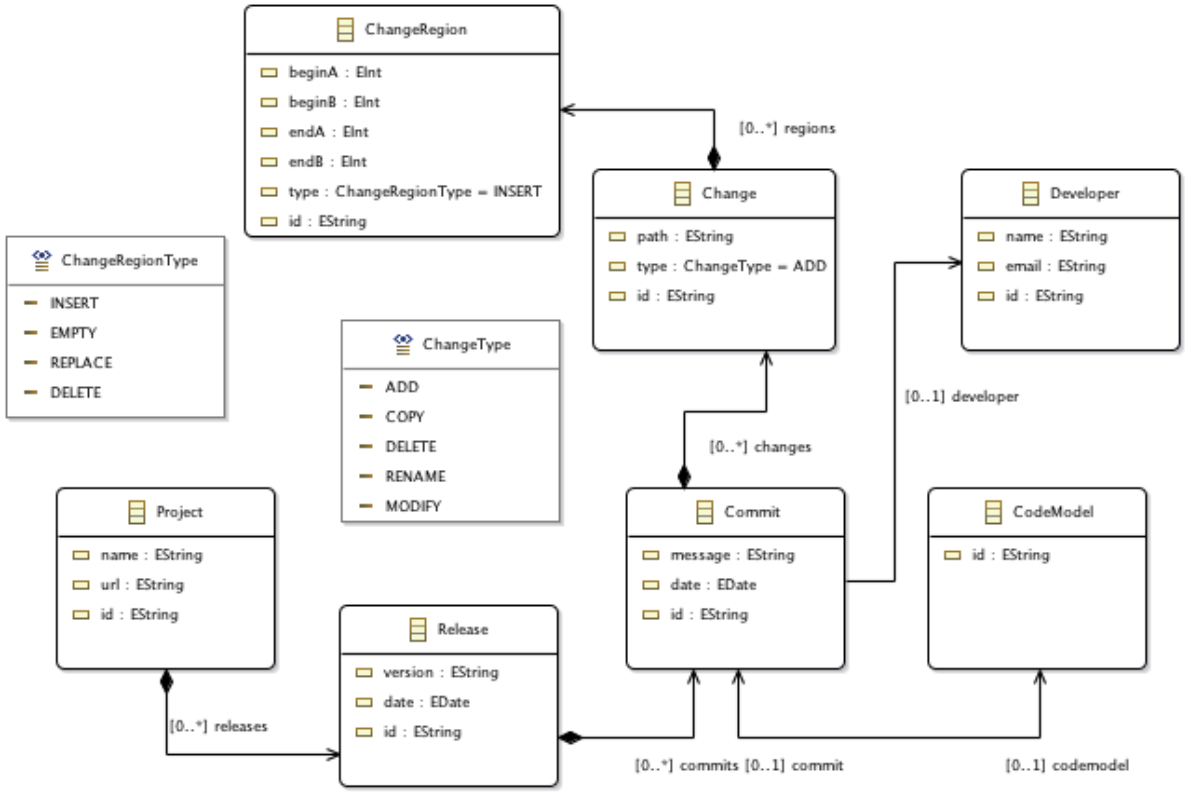
Release: primeiro nível de controle relacionado a evolução do projeto e contém a data de criação e a versão que representa;

Commit: os dados de alto nível dos commits. Contém a data que o commit foi enviado, a mensagem. Conforme citado anteriormente, cada commit contém um snapshot específico do projeto;

Change: as alterações enviadas num commit. Cada arquivo modificado num commit é representado unicamente por uma instância da classe Change. As modificações representadas em nível de arquivo são adição de arquivo (ADD), Cópia, caso detectada (COPY), remoção (DELETE), renomeação, caso detectável (RENAME), e modificação (MODIFY);

ChangeRegion: as alterações no conteúdo de um arquivo. Cada modificação é representada unicamente por uma instância da classe ChangeRegion. As modificações representadas em nível de linha modificada são remoção da sequência B (DELETE), alteração não detectada (EMPTY), inserção da sequência B (INSERT), substituição pela sequência B (REPLACE).

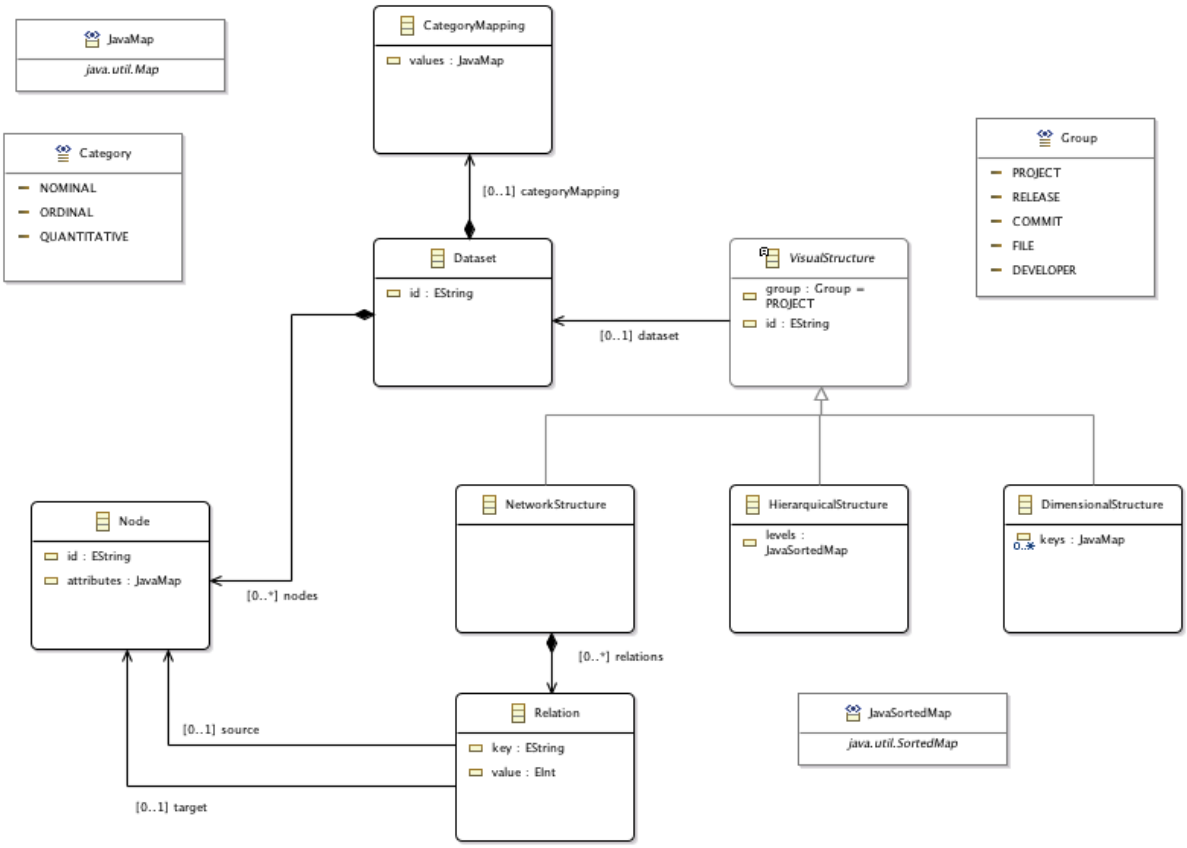
Figura 14 - Modelo de Evolução do Sourceminer Trends



Do ponto de vista das representações visuais, também foi criado um modelo. Ele serve de elo entre as visões disponíveis e o conjunto de dados extraídos. O objetivo é permitir a utilização de múltiplas visões, desde que sigam a estrutura definida neste modelo. O modelo visual foi projetado para ser flexível a ponto de suportar um conjunto abrangente de visões e, ao mesmo tempo, suportar serialização em JSON num formato compatível com os gráficos

presentes na API de visualização d3 (Data Driven Documents, 2016). No modelo, uma estrutura visual (`VisualStructure`) está relacionada com um conjunto de dados (`Dataset`), o que é formado por nós (`Node`). Datasets são independentes para permitir uma configuração baseada em múltiplas visões, na qual cada visão realiza uma manipulação específica no conjunto de dados. O modelo é ilustrado na figura 15.

Figura 15 - Modelo Visual

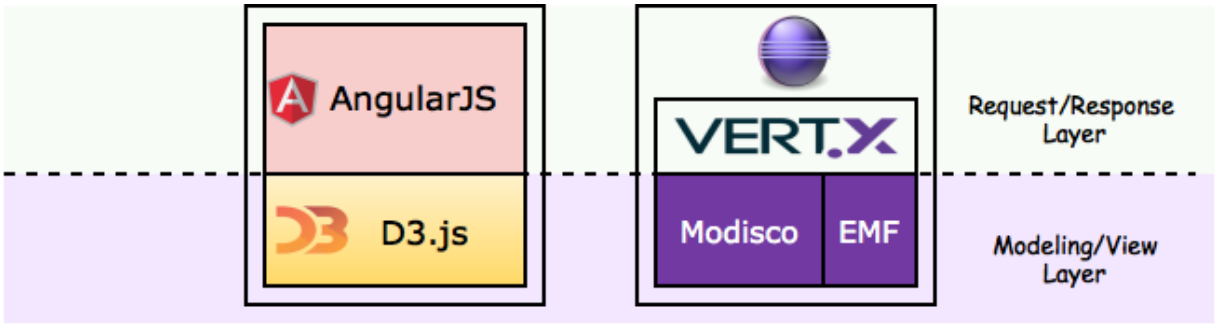


O conceito de versão ou release adotado no Sourceminer Trends refere-se a uma marcação (`tag`) significativa na ramificação estável de um projeto. Determinar se uma tag é significativa ou não depende inteiramente do propósito da análise. Nesse sentido, optou-se por criar um filtro para a seleção de tags por meio de expressão regular, definida pelo usuário. O escopo de cada release compreende os commits não computados na release anterior seu último commit. Cada instância dos objetos de commit contém uma representação completa do projeto. Isso permite fazer comparações em diferentes níveis de detalhe (SCHEIDGEN; FISCHER, 2014).

4.2 SOLUÇÃO PROPOSTA

O Sourceminer Trends oferece os serviços de importação de dados históricos de softwares hospedados em repositórios de código e o serviço de análise dos dados obtidos. A ferramenta é composta por uma interface visual baseada em tecnologias web e um lado servidor no qual a plataforma Eclipse é utilizada como base. Do lado cliente, os principais componentes são o Angular JS (ANGULAR JS, 2016), utilizado para articular a interação com elementos da interface com o usuário e o D3, biblioteca que implementa as visões propostas na aplicação. Já no lado servidor, são utilizados predominantemente plugins do Eclipse, o qual encontra-se em execução sem interface gráfica, modo de uso também chamado de *headless*. Este artifício é necessário para que os plugins funcionem corretamente, visto que em geral eles não são projetados para serem utilizados de forma independente. Esse esquema é representado na figura 16, onde o EMF também aparece como uma das tecnologias fundamentais do Sourceminer Trends. Nela as tecnologias envolvidas são separadas em duas camadas para facilitar a compreensão. A camada de requisição/resposta representa a interação cliente/servidor, ao passo que a camada de modelagem/visão demonstra onde reside a lógica de negócio e os produtos gerados para visualização das metáforas visuais.

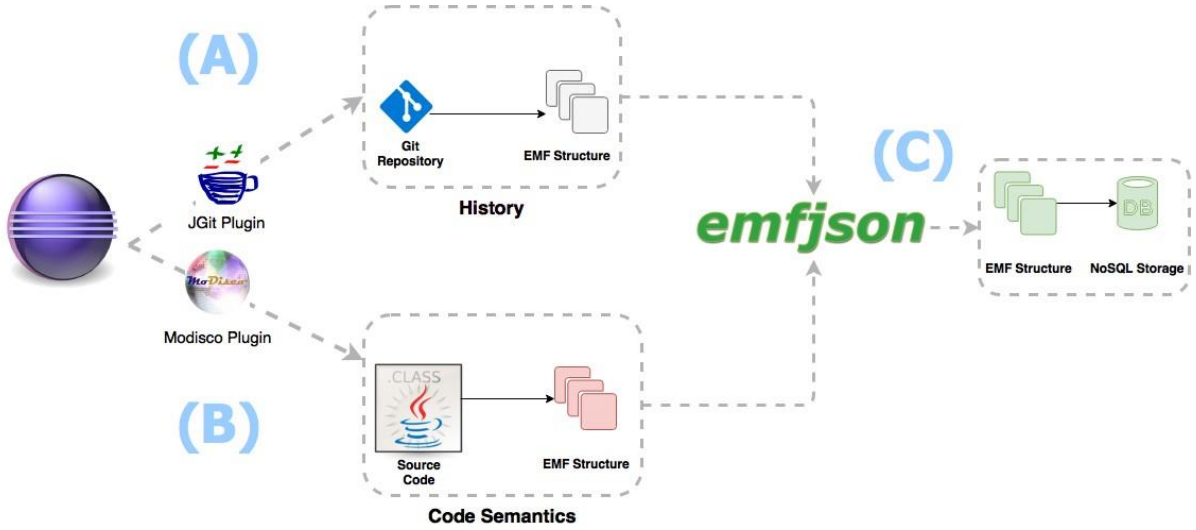
Figura 16 - Camadas do Sourceminer Trends



A solução proposta foi projetada para lidar com processos de longa duração, tipicamente em atividades de importação e transformação de dados. Essa dinâmica é ilustrada na figura 17 e consiste em realizar a cópia de um projeto hospedado num repositório de código baseado em Git como o GitHub ou Bitbucket. Esse processo é iniciado com a solicitação de importação pelo usuário, quando são informados os dados de conexão com o repositório e critérios para limitar o escopo do que deve ser extraído. Após a importação, o projeto é

armazenado numa pasta temporária, de onde são extraídos os dados históricos e a estrutura do código-fonte para cada versão disponível. A transformação consiste em gerar instâncias do modelo de evolução para cada release (A) e dos modelos de código-fonte para cada commit (B). Por fim, as instâncias são transformadas em JSON para armazenamento no banco de dados subjacente (C), através do plugin emf-json.

Figura 17 - Mecanismo de Importação de Projetos



O mecanismo de importação é detalhado a seguir.

4.2.1.1 Obtenção do Projeto

Nesta etapa tanto o código-fonte quanto seu histórico de alterações são copiados para um diretório temporário, local cuja rotina de conversão em modelos utiliza como referência. Informações de ambos os artefatos são então extraídas e unificadas numa instância do modelo de evolução. Com o Git, esta fase refere-se a clonagem do projeto.

4.2.1.2 Transformação em Instância de Modelo

Uma vez que o projeto resida localmente, segue-se para geração das instâncias dos metamodelos de código e evolução. Como cada commit contém sua própria instância de modelo de código, é necessário que o conteúdo do projeto seja sincronizado com o commit. Nesse ponto o fato do Git permitir fazer checkouts locais emerge como uma vantagem, pois com ele permite reajustar a pasta do projeto com o conteúdo específico do commit, sem

realizar qualquer operação remota. Tecnicamente, é feito um *checkout* para cada commit antes da geração da instância do modelo de código. O processo de transformação é direto, conforme evidenciado na listagem 3. Ao percorrer cada commit, é executado o comando `discoverElement` do discoverer indicado. Vale ressaltar que o `DiscoverJavaModelFromJavaProject` assume que o projeto selecionado para importação contém os metadados do Eclipse que identificam as pastas de código-fonte, bibliotecas, etc. Portanto, apenas projetos compatíveis com os metadados do Eclipse são suportados.

Listagem 3 - Conversão de código Java em modelo Java através do Modisco

```
	public static Resource discoverKDMModel(String projectPath) throws
DiscoveryException {
		IJavaProject project = loadProject(new File(projectPath));

		if (project == null) {
			return null;
		}

		AbstractModelDiscoverer<IJavaProject> discoverer = new
DiscoverJavaModelFromJavaProject();
		discoverer.discoverElement(project, new NullProgressMonitor());

		return discoverer.getTargetModel();
	}
```

4.2.1.3 Persistência (b3)

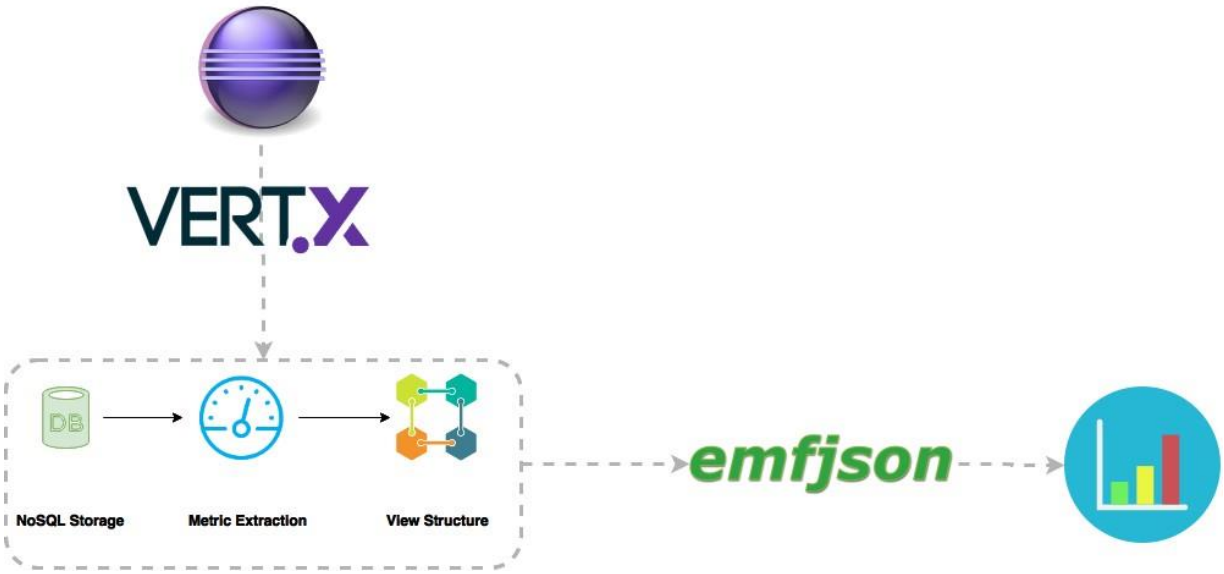
A persistência de dados no Sourceminer Trends é em CouchDB. Instâncias do modelo de evolução são armazenadas como JSON e consultadas programaticamente. Um exemplo de extração pode ser visto na listagem 4.

Listagem 4 - Exemplo de instância do modelo de Evolução no formato JSON

```
{
"_id": "project__HEJHQD2BEeaVe4bgV1tq2Q",
"_rev": "1-4b4f48bf87dc54bb705bb137dbad3dd8",
"contents": {
"eClass": "http://trends.sourceminer.org.br/model/evolution#//Project",
"name": "trends_server",
"url": "https://bitbucket.org/fcostasilva/trends_server",
"id": "_HEJHQD2BEeaVe4bgV1tq2Q",
"releases": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//Release",
"version": "0.1",
"date": 1466091985000,
"id": "5d8cf0eb9a0bb20a2f8aad524f73f8555b16f20d",
"model": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//CodeModel",
"$ref": "model__Hq_E0D2BEeaVe4bgV1tq2Q?rev=1-
0db9d11909e929b833d078299021ae44#_Hq_E0D2BEeaVe4bgV1tq2Q"
},
"commits": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//Commit",
"message": "Optimizations in the model",
"date": 1466001313000,
"id": "511cc4a28dc2ccd728cfcd2c05bfe39e9f5180a3",
"parentCommit": "9ab5667f5dd78fd6ad5ef23bac360aaabb89db75",
"developer": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//Developer",
"$ref": "developer__Gy8CgT2BEeaVe4bgV1tq2Q?rev=1-
b5212116fadb2de1edd382e0c2c1c1fa#_Gy8CgT2BEeaVe4bgV1tq2Q"
},
"changes": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//Change",
"path":
"br.org.sourceminer.trends.server/src/br/org/sourceminer/trends/server/persistence/Mongo
DBProvider.java",
"type": "MODIFY",
"regions": {
"eClass":
"http://trends.sourceminer.org.br/model/evolution#//ChangeRegion",
"beginA": 42,
"beginB": 43,
"endA": 43,
"endB": 44,
"type": "REPLACE"
}
}
}
}
}
}
```

O serviço de análise é representado na figura 18. Ela baseia-se extração de métricas a partir de instâncias do modelo de evolução de um projeto. As métricas são então mapeadas em estruturas de dados adequadas para representação visual.

Figura 18 - Estrutura de Análise Projetos

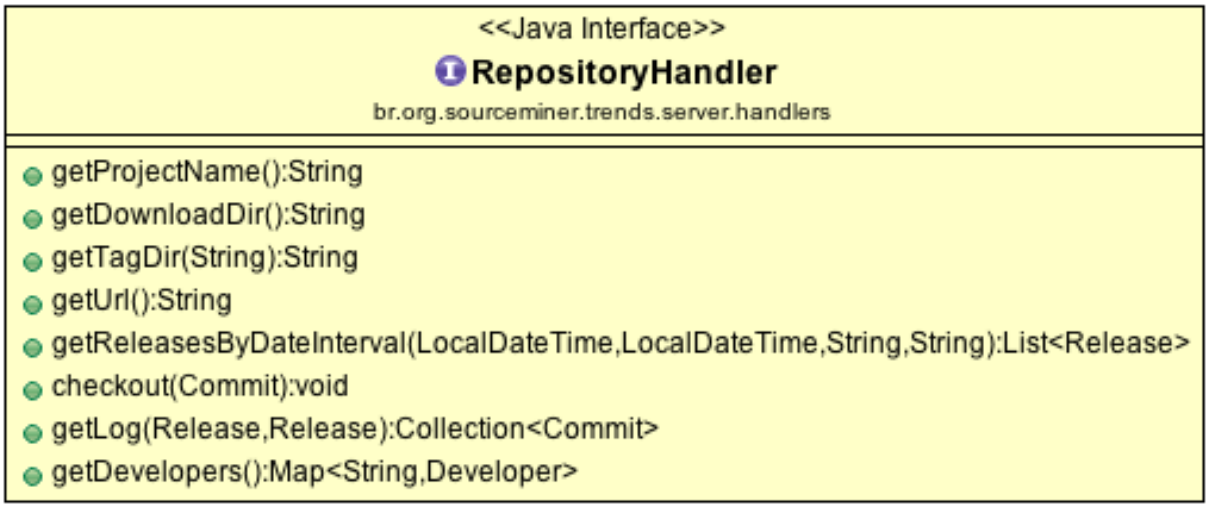


4.3 RESTRIÇÕES E LIMITAÇÕES

A solução proposta suporta apenas projetos Java, compatíveis com o formato de metadados utilizado no Eclipse, e hospedados em repositórios Git. Essas limitações são impostas pelo ambiente de execução baseado em Eclipse. No caso do Git, optou-se por utilizá-lo em detrimento de outros sistemas como o SVN para simplificar a solução, restringindo o suporte ao sistema de controle de versão que desperta mais interesse na comunidade científica e comercial [1]. No entanto a ferramenta é extensível e permite a adição de suporte a outros sistemas de controle de versão por meio de conectores, conforme ilustrado na figura 19. O conector provê a interface mínima de interação com repositórios, na qual são definidas operações de cópia e filtros. Cada sistema de controle de versão a ser suportado deve implementar esta interface. O conector padrão implementado no Sourceminer Trends é baseado em repositórios Git. Essas operações são executadas no Sourceminer Trends por meio do JGit, um framework de código-aberto. Do ponto de vista de análise, o Sourceminer Trends depende de implementações de terceiros para as métricas. Métricas não são pré-definidas.

[1] http://www.netinstructions.com/the-case-for-git

Figura 19 - Interface para Conectores de Repositórios



4.4 CONCLUSÃO

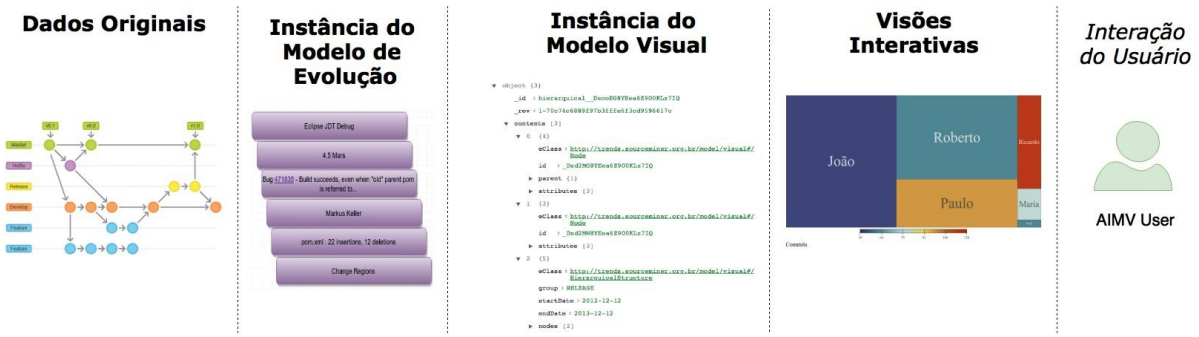
Este capítulo apresentou como o Sourceminer Trends foi projetado para realizar a extração de dados de evolução, o mapeamento de todas as versões do projeto analisado em modelos EMF e a representação dos dados obtidos em estruturas visuais. Identificou-se que uma instância *headless* da IDE Eclipse é utilizada como espinha dorsal, necessária para utilização de plugins como o Modisco. Também foram abordadas as restrições da ferramenta, a qual está limitada a projetos obtidos de repositórios Git, cujo código-fonte é escrito em Java e cujos metadados são baseados na IDE Eclipse. O próximo capítulo abordará a estratégia de mapeamento de dados de evolução em representações visuais.

5 MAPEAMENTO ATRIBUTO REAL X ATRIBUTO VISUAL

5.1 INTRODUÇÃO

De modo geral, estudos a respeito de evolução de software através de ferramentas de visualização não costumam abordar explicitamente o mapeamento entre o conjunto de dados a ser analisado e as propriedades das representações visuais. Isto é considerado um detalhe de implementação (NOVAES; TORRES, *et al.*, 2013). Essa formalização, entretanto, é um aspecto relevante da criação de uma visualização. As representações visuais devem ser consequência da transformação dos dados originais em estruturas de dados visuais, as quais por sua vez são insumos para visões (CARD; MACKINLAY; SHNEIDERMAN, 1999). Essa cadeia de transformação é ilustrada na figura 20. Ela procura ilustrar o fluxo desde a transformação dos dados originais até o ponto em que as visões estão disponíveis para interação com o usuário. Importante notar a presença dos modelos de evolução e visual, utilizados para formalizar respectivamente os dados originais e como eles serão dispostos para representação visual.

Figura 20 - Cadeia de Interação/Transformação de Dados

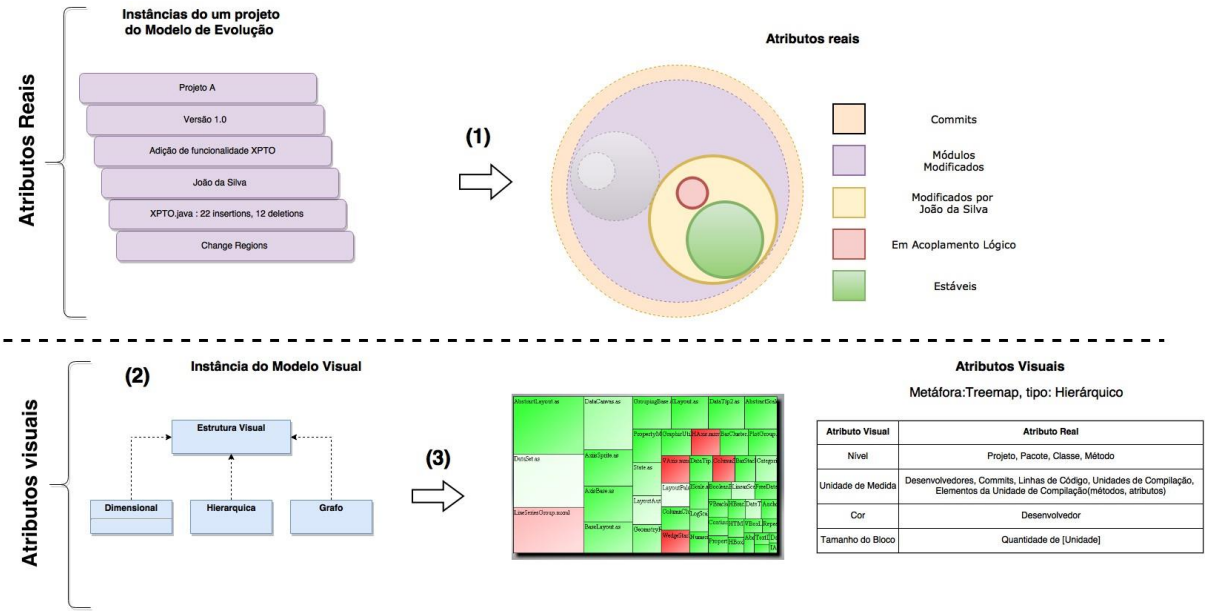


Definir esta cadeia de eventos, baseada em modelos, permite que um mesmo conjunto de dados seja analisado em diferentes perspectivas, desde que mapeado numa estrutura de dados compatível. Ao mesmo tempo, formalizar a configuração da representação visual favorece a utilização de metáforas que utilizem configurações semelhantes. O mapeamento, portanto, implica em classificar as metáforas visuais em categorias. Essa classificação tem sido definida como dimensional (1 a n dimensões), hierárquica, temporal e em rede

(SHNEIDERMAN, 1996); e relacionada com técnicas de visualização (CHI, 2000); (KIRK, 2012).

Com base nesses estudos, foi elaborada uma estratégia de mapeamento exemplificada na figura 21. Ela está dividida em duas seções: uma para os atributos dos dados e outra para os atributos da visão. Na primeira seção, são extraídas da instância de um modelo de evolução informações específicas para um objetivo de compreensão, como análise de contribuição de desenvolvedores, compreendendo os chamados atributos reais. Em seguida, os atributos reais são mapeados para estruturas de dados visuais, as quais são associadas a categorias específicas de metáforas visuais. Por fim, a metáfora visual escolhida pelo usuário pode ser configurada para representar os atributos reais de acordo com seus atributos visuais. A figura ilustra um treemap, que utiliza uma estrutura de dados hierárquica e possui, dentre outros atributos visuais, o tamanho do bloco.

Figura 21 - Estratégia de Mapeamento



As estruturas visuais sugeridas são a dimensional, hierárquica e de rede. Os elementos do conjunto de dados são sempre categorizados como nominais, ordinários ou quantitativos. Elementos nominais são adequados para simbolizar categorias, ao passo que os ordinários pressupõem uma ordem entre os elementos. Já os quantitativos correspondem a qualquer valor numérico. Essa classificação tem por objetivo identificar quais atributos de nós podem ser

mapeados para propriedades de uma determinada visão. Por exemplo, a propriedade `posicao` é mais adequada para simbolizar atributos numéricos ou ordinários, ao passo que cores são mais indicadas para visualização de elementos discretos. As estruturas dimensional e hierárquica são similares. Ambas compartilham o conceito de utilizar um tipo `Map` definido arbitrariamente, cuja única função é proporcionar a serialização dos atributos em linha, ao invés de coluna. A diferença entre eles é que o Map da estrutura hierárquica é inerentemente ordenado. Já a estrutura de rede contém uma associação com a classe `Relation`, que mapeia o relacionamento entre nós. Um detalhe importante da modelagem é a definição do tipo de associação. Relações todo-parte como `NetworkStructure` e `Relation` formam um único documento, ao passo que associações simples como `VisualStructure` e `Dataset` são serializadas separadamente. As listagens 5,6 3 7 ilustram o código serializado no formato JSON. Para cada uma delas, nota-se que o elemento `Dataset` é apenas referenciado pelas estruturas visuais, ao invés de ser englobado ao documento.

Listagem 5 - Instância do Modelo Dimensional

```
	[
	{
		"eClass": "http://trends.sourceminer.org.br/model/visual#//Dataset",
		"id": "_N0hV0IwYEeaperrLdOk8Pw",
		"nodes": {
			"eClass": "http://trends.sourceminer.org.br/model/visual#//Node",
			"id": "_N0fgoYwYEeaperrLdOk8Pw",
			"attributes": {
				"addedLines": "43",
				"file":
"br.org.sourceminer.sample/src/br/org/sourceminer/sample/Animal.java",
				"releaseId": "af2553fe97e4fb78f499d3e86ae821bbd91c4d36",
				"metric": "churn",
				"releaseVersion": "0.1",
				"deletedLines": "0"
			}
		}],
		"categoryMapping": {
			"eClass":
"http://trends.sourceminer.org.br/model/visual#//CategoryMapping",
			"values": {
				"addedLines": "QUANTITATIVE",
				"file": "NOMINAL",
				"releaseId": "NOMINAL",
				"metric": "NOMINAL",
				"releaseVersion": "ORDINAL",
				"deletedLines": "QUANTITATIVE"
			}
		}
	},
	{
		"eClass":
"http://trends.sourceminer.org.br/model/visual#//DimensionalStructure",
		"group": "RELEASE",
		"keys": {
			"x": "releaseId",
			"y": "file"
		},
		"dataset": {
```

```
                    "$ref": "_N0hV0IwYEeaperrLdOk8Pw"
                }
        }
        ]
```

Listagem 6 - Instância do Modelo de Rede/Grafo

```
        [ {
          "eClass" : "http://trends.sourceminer.org.br/model/visual#//Dataset",
          "id" : "_bQImYIwYEeaAZZwHJPswTg",
          "nodes" : [ {
            "eClass" : "http://trends.sourceminer.org.br/model/visual#//Node",
            "id" : "_bQEU8YwYEeaAZZwHJPswTg",
            "attributes" : {
              "addedLines" : "43",
              "file" :
"br.org.sourceminer.sample/src/br/org/sourceminer/sample/Animal.java",
              "releaseId" : "af2553fe97e4fb78f499d3e86ae821bbd91c4d36",
              "metric" : "churn",
              "releaseVersion" : "0.1",
              "deletedLines" : "0"
            }
          } ],
          "categoryMapping" : {
            "eClass" : "http://trends.sourceminer.org.br/model/visual#//CategoryMapping",
            "values" : {
              "addedLines" : "QUANTITATIVE",
              "file" : "NOMINAL",
              "releaseId" : "NOMINAL",
              "metric" : "NOMINAL",
              "releaseVersion" : "ORDINAL",
              "deletedLines" : "QUANTITATIVE"
            }
          }
        }, {
          "eClass" : "http://trends.sourceminer.org.br/model/visual#//NetworkStructure",
          "group" : "RELEASE",
          "dataset" : {
            "$ref" : "_bQImYIwYEeaAZZwHJPswTg"
          },
          "relations" : [ {
            "eClass" : "http://trends.sourceminer.org.br/model/visual#//Relation",
            "key" : "releaseId",
            "source" : {
              "$ref" : "_bQEU8YwYEeaAZZwHJPswTg"
            },
            "target" : {
              "$ref" : "_bQEU8owYEeaAZZwHJPswTg"
            }
          } ]
        } ]
```

Listagem 7 - Instância do Modelo Hierárquico

```
        [ {
          "eClass" : "http://trends.sourceminer.org.br/model/visual#//Dataset",
          "id" : "_VtJe8IwYEea5_tS_tqCnIQ",
          "nodes" : [ {
            "eClass" : "http://trends.sourceminer.org.br/model/visual#//Node",
            "id" : "_VtHCsYwYEea5_tS_tqCnIQ",
            "attributes" : {
              "addedLines" : "43",
              "file" :
"br.org.sourceminer.sample/src/br/org/sourceminer/sample/Animal.java",
              "releaseId" : "af2553fe97e4fb78f499d3e86ae821bbd91c4d36",
              "metric" : "churn",
              "releaseVersion" : "0.1",
              "deletedLines" : "0"
            }
          } ],
          "categoryMapping" : {
            "eClass" : "http://trends.sourceminer.org.br/model/visual#//CategoryMapping",
            "values" : {
              "addedLines" : "QUANTITATIVE",
              "file" : "NOMINAL",
              "releaseId" : "NOMINAL",
              "metric" : "NOMINAL",
              "releaseVersion" : "ORDINAL",
              "deletedLines" : "QUANTITATIVE"
            }
          }
        }, {
          "eClass" :
"http://trends.sourceminer.org.br/model/visual#//HierarquicalStructure",
          "group" : "RELEASE",
          "levels" : {
            "0" : "releaseId",
            "1" : "file"
          },
          "dataset" : {
            "$ref" : "_VtJe8IwYEea5_tS_tqCnIQ"
          }
        } ]
```

Associar atributos reais com visuais traz uma série de benefícios para construção de um ambiente interativo baseado em múltiplas visões. Primeiramente, essa associação permite que diferentes metáforas trabalhem de forma sincronizada. Isso significa que um mesmo atributo, seja ele real ou visual, pode ser representado em diferentes visões. Nesse sentido dois cenários são possíveis: representar o mesmo conjunto de dados em duas ou mais metáforas ou representar o mesmo mapeamento atributo real X atributo visual para um ou mais elementos do conjunto de dados. Assim, ilustrativamente a quantidade de desenvolvedores de um projeto pode ser representada tanto num *treemap* quanto num grafo, e o tamanho dos elementos pode indicar a quantidade de commits em ambas as metáforas.

5.2 ESTRATÉGIA DE MAPEAMENTO

A estratégia é definida em duas etapas: a definição dos atributos reais por meio da aplicação do GQM, e o mapeamento para os atributos visuais com base nos dados obtidos na etapa anterior. O GQM direciona a extração de informações para contextos específicos, definidos pelo usuário, antes do processo de visualização iniciar. Já o mapeamento para atributos visuais é definido de acordo com o processo no qual devem ser considerados (MAZZA, 2009):

- Natureza dos dados: Podem ser quantitativos, ordinais (não numéricos, porém com ordem intrínseca), ou categóricos (não numéricos sem ordem definida). Outras classificações são possíveis;
- Número de dimensões: se diferentes atributos reais são dependentes entre si;
- Estrutura de dados: dimensional, hierárquico ou em rede;
- Tipo de Interação: estático, transformável (quando o usuário pode configurar a visão), ou manipulável (quando o usuário pode navegar nos dados, como em técnicas de zoom in/out).

5.3 CONCLUSÃO

Este capítulo detalhou como os dados de evolução obtidos pelo Sourceminer Trends devem ser mapeados para representações visuais. Identificou-se que os dados devem ser obtidos através da aplicação do GQM, classificados em categorias (nominal, quantitativo e ordinal), e associados com estruturas de dados visuais compatíveis. Na seção seguinte é realizado um experimento ilustrando exemplos de uso da ferramenta.

6 EXEMPLO DE USO

Este capítulo visa apresentar um exemplo de uso do Sourceminer Trends. Ele compreende a definição do objeto de estudo, dos objetivos de compreensão e o mapeamento entre atributos reais e visuais. Ainda, este estudo busca verificar se as funcionalidades disponibilizadas pela ferramenta se encontram alinhadas com os objetivos de pesquisa apontados nesta dissertação.

6.1 OBJETIVOS

O exemplo de uso foi projetado para ilustrar como o Sourceminer Trends pode auxiliar na análise de evolução de software, oferecendo funcionalidades de extração de dados de repositórios de código-fonte e análise num ambiente interativo baseado em múltiplas visões. Com base nas restrições levantadas durante a apresentação da solução proposta, foi selecionado o projeto Eclim (2016) como referência, um software de código aberto que disponibiliza funcionalidades da IDE Eclipse como *code completion* através da linha de comando ou uma conexão de rede local, permitindo que essas características sejam incorporadas a outros editores. Esta não foi uma escolha aleatória.

6.2 PLANEJAMENTO

A estratégia utilizada se baseia em determinar objetivos de compreensão utilizando a metodologia GQM, para assim determinar que tipo de métrica deve ser extraída e qual visão deve ser utilizada. Para tanto, utilizou-se o modelo GQM inspirado no estudo (LEHMAN; PERRY; RAMIL, 1998). O propósito geral foi examinar como um software evolui, sob o ponto de vista de um pesquisador. Os objetivos adotados são descritos a seguir:

Objetivo 1: Identificar se as alterações presentes em releases, num determinado período, seguem um ritmo previsível.

Questão 1: Como o ritmo de crescimento do software utilizado entre releases?

Questão 2: Como quantificar as alterações aplicadas em uma release?

Questão 3: Como comparar duas ou mais releases em relação a quantidade de alterações

aplicadas?

Métricas: Code Churn e Code delta.

Objetivo 2: Identificar como os desenvolvedores participam do software objeto de estudo

Questão 1: Como aferir que desenvolvedores estão ativos num determinado período?

Questão 2: Como comparar desenvolvedores entre si, em termos de quantidade de alterações efetuadas?

Métrica: atividade dos desenvolvedores

Code churn e code delta (HALL; MUNSON, 2000) são duas medidas similares e que auxiliam na avaliação do crescimento de um software ao longo do tempo. Ambas quantificam os elementos (linhas de código, número de arquivos de código ou número de pacotes) que foram adicionados ou removidos entre duas ou mais releases. No caso do code delta, remoções são calculadas como subtrações, ao passo que o churn calcula remoções (e alterações) como adições. Um arquivo onde foram efetuadas 1 adição, 1 alteração e 1 remoção, por exemplo, terão as medidas 3 para churn e 0 para delta. Nota-se que o churn é uma medida mais eficiente para medir a atividade que ocorre entre releases.

Em se tratando de código-fonte, representa quantidade de linhas de código adicionadas menos as removidas. A soma desses deltas é o resultado final. A quantidade de deltas também é medida para aferir a atividade num elemento. Por fim, a atividade dos desenvolvedores (SHIN, MENEELY, *et al.*, 2011), permite definir quem realizou alterações em arquivos, quando elas ocorreram há acoplamento lógico entre eles.

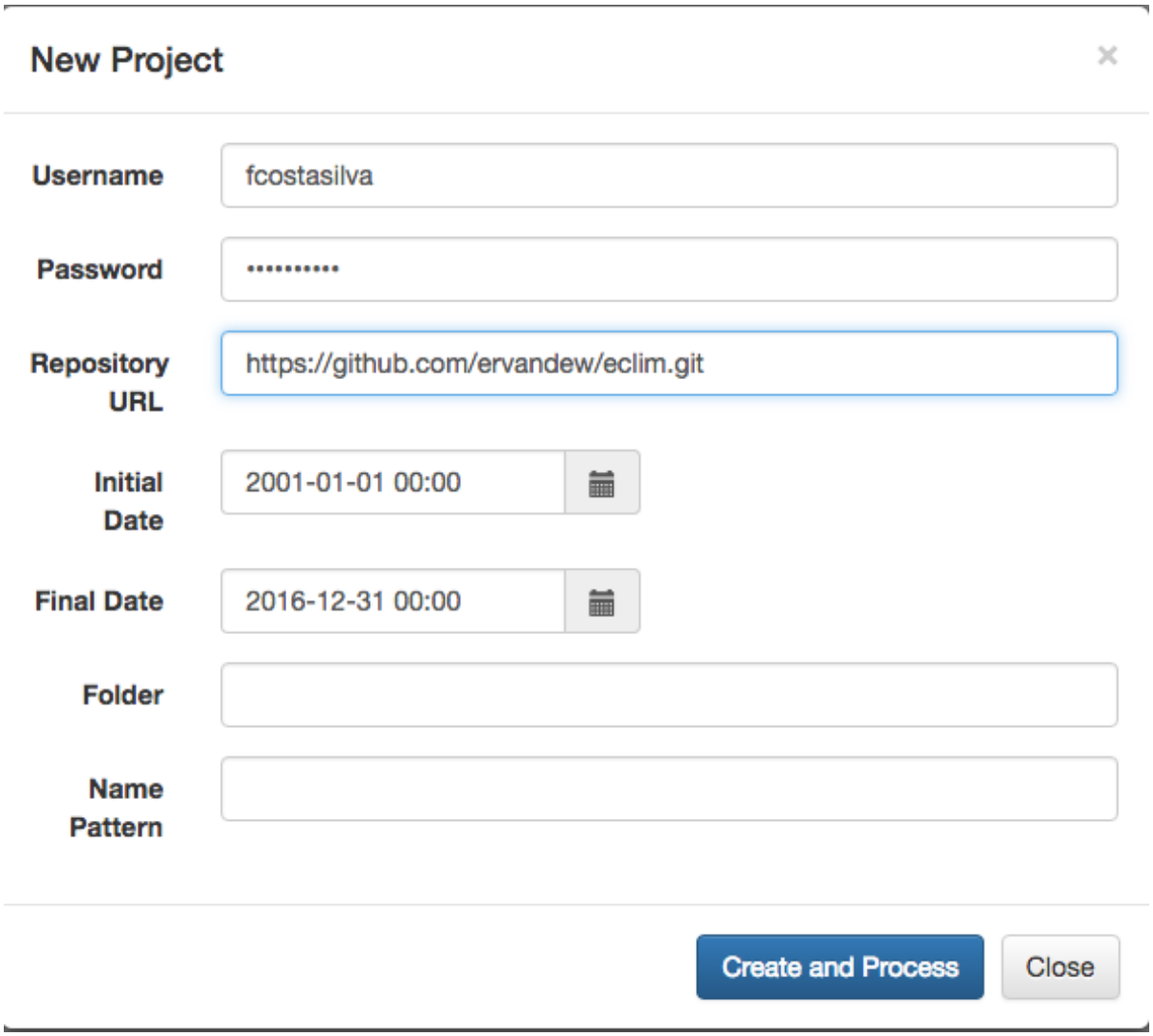
As seções a seguir descrevem as etapas do experimento.

6.3 PROCEDIMENTO E EXECUÇÃO

O primeiro passo consiste na extração dos metadados de evolução do projeto Eclim, a partir de seu repositório no GitHub (GITHUB, 2016). A extração foi iniciada pela interface visual. Entretanto, toda interação com a infraestrutura é realizada por uma interface REST disponível para qualquer cliente que suporte esse formato. A figura 22 exibe a tela de extração. As datas foram escolhidas de forma a obter todo o histórico, e estão no formato yyyy-MM-dd

HH:mm para uma maior precisão na extração. Não foi necessário definir uma pasta para realização da extração, uma vez que o Eclim não contém subprojetos no diretório raiz do repositório. Além disso, o padrão de nomes das releases sugere que elas são versões finais, ao contrário de projetos como o Eclipse, no qual elas podem significar *milestones*, candidatos ou versões finais (ECLIPSE DEVELOPMENT PROCESS, 2016).

Figura 22 - Tela de Importação de Projetos



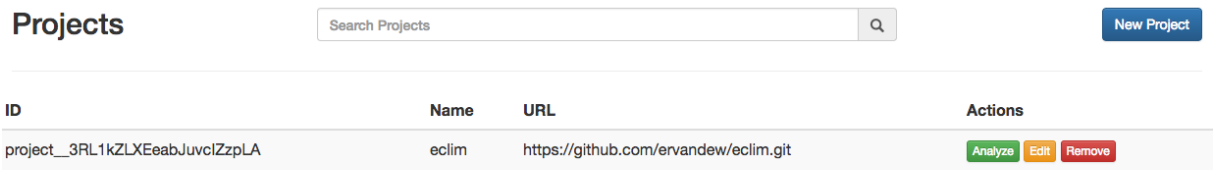
Uma instância do modelo de evolução foi criada com base nos dados obtidos e armazenada no CouchDB por meio da interface provida por um adaptador CouchDB para EMF. Na listagem 8 nota-se que o documento que representa o projeto é filtrado diretamente através do atributo `eClass`.

Listagem 8 - *View* para listagem de projetos no CouchDB

```
        function (doc) {
          //seleciona todos os documentos do tipo Project
           if(doc.contents.eClass ==
"http://trends.sourceminer.org.br/model/evolution#//Project"){
           emit(doc._id, {id:doc._id, name:doc.contents.name, url:doc.contents.url});
          }
        }
```

O conceito é particularmente útil para otimizar o desempenho de buscas que não dependem de conformidade com modelos. Pelo fato de serem indexadas, são rápidas e evitam que o processo de montagem dos objetos EMF. Neste caso particular, são retornados o identificador, o nome do projeto e o endereço do seu repositório. Vale ressaltar também que o CouchDB não utiliza o conceito de tabelas, assim cada documento deve definir um atributo para indicar seu tipo. Como o atributo `eClass` é criado para cada recurso EMF serializado, não houve a necessidade de adaptação do modelo. A importação é realizada de acordo com a tela exibida na figura 23. Uma vez importado, o projeto pode ser analisado. Analisar o projeto implica em definir uma métrica e uma visualização por parte do usuário. Entretanto nem todas as visões estão disponíveis para as métricas indicadas. Estruturas como o grafo são mais adequadas para representar relacionamentos entre elementos, o que exige um tipo de extração específico para esse caso.

Figura 23 - Projeto Importado



Uma vez informadas a visão e a métrica, a requisição é recebida por uma classe que implementa a interface descrita na listagem 9. Para qualquer tipo de extração, é enviado o projeto - já carregado em memória - e um filtro, onde podem ser especificadas as releases afetadas. O resultado é um dataset, que serve como insumo para as visões. As visões são montadas em seguida, dependendo da escolha do usuário. Este fluxo torna evidente a

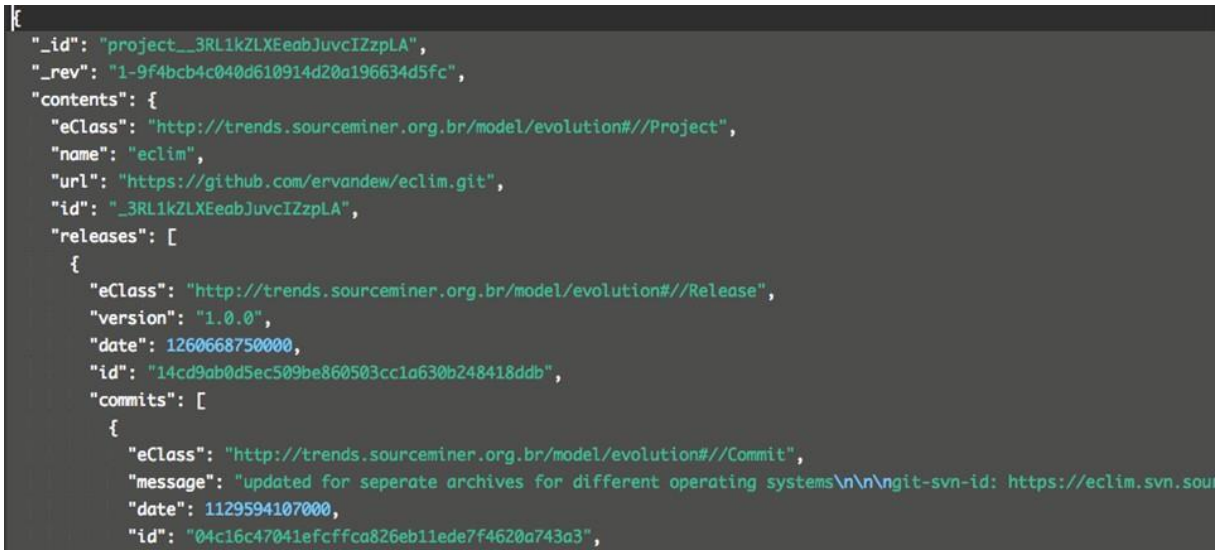
independência do dataset em relação às estruturas visuais.

Listagem 9 - Interface para Processamento de Métricas

```
public interface MetricExtractor {
        Dataset extract(Project project, Filter filter);
}
```

Na figura 24 é exibido um trecho do código JSON gerado pelo Sourceminer Trends para o Eclim. Como ilustrado, cada objeto armazenado tem um tipo explicitamente identificado no atributo `eClass`. Esse atributo permite que objetos EMF sejam criados a partir do JSON e transformados para as classes que os originaram.

Figura 24 - Estrutura de Projeto Eclim Importado



Cada atributo do modelo de evolução pode ser mapeado, em momento de execução, para um atributo da representação visual ativa, desde que estejam igualmente categorizados. Dados numéricos, por exemplo, devem ser relacionados com atributos visuais categorizados de tal maneira. Para o churn, a categorização foi realizada conforme a tabela 4.

Tabela 4 - Mapeamento de Atributos Reais para a Métrica Churn

Atributo	Categoria	Descrição
File	NOMINAL	Nome do arquivo alterado
addedLines	QUANTITATIVO	Quantidade de linhas adicionadas
deletedLines	QUANTITATIVO	Quantidade de linhas removidas
releaseId	NOMINAL	Identificador da Release
releaseVersion	ORDINAL	Versão da Release
metric	NOMINAL	Métrica extraída
Churn	QUANTITATIVO	Medida do Churn
developers	NOMINAL	Desenvolvedores que atuaram no arquivo
changeTimes	TEMPORAL	Datas em que ocorreram alterações
releaseDate	TEMPORAL	Data de lançamento da release

Os atributos `developers` e `changeTimes` descrevem listas e foram mantidos desde modo para possibilitar a navegação numa visão secundária/auxiliar. O atributo `changeTimes`, por exemplo, pode ser utilizado numa visão que sugere uma linha do tempo. Cada extração é realizada no menor nível de granularidade, neste caso em nível de arquivo. Os dados são extraídos de forma a serem compatíveis com o d3plus, cabendo a visão ou ao usuário agrupá-los para uma visualização particular. Na tabela 5 são representados os atributos da métrica Atividade dos Desenvolvedores. Este dataset foi otimizado para montagem e processamento de visões baseadas em grafos, de modo a representar relacionamentos entre desenvolvedores. Neste formato é possível associar releases e arquivos em particular aos desenvolvedores. Em todos os mapeamentos, a própria métrica é citada como atributo. Esta modelagem visa permitir, no futuro, que um mesmo dataset contenha dados de diferentes fontes, cabendo ao usuário manipular os dados na camada visual.

Tabela 5 - Mapeamento de atributos reais para a métrica Atividade dos Desenvolvedores

Atributo	Categoria	Descrição
developerName	NOMINAL	Nome do Desenvolvedor
developerEmail	NOMINAL	E-mail do Desenvolvedor
fileChanges	QUANTITATIVE	Arquivos modificados pelo desenvolvedor
releaseIds	NOMINAL	Releases onde o desenvolvedor participou
numberChanges	QUANTITATIVE	Total de modificações realizadas pelo
metric	NOMINAL	Métrica extraída

Entre a proposta original e o uso real, os modelos de evolução e visuais sofreram ajustes, de forma incremental, até atingirem um ponto em que a serialização se tornou ideal para armazenamento e exportação. Em relação ao modelo visual, as mudanças visaram aumentar a coesão, de forma que elementos relacionados fossem exportados numa mesma estrutura. Além disso, o modelo de código gerado pelo Modisco foi associado inversamente com o commit que o originou para permitir a navegação em ambas as direções. Em relação ao modelo visual, o dataset foi englobado à estrutura visual para que fosse retornado numa única consulta. Caso contrário, a estrutura e o dataset dependeriam de chamadas REST separadas. Além deste caso, tanto a estrutura dimensional quanto a hierárquica foram simplificadas para refletir a forma como o d3plus define dimensões e hierarquias. Por fim, os atributos de nós, os níveis das hierarquias e as dimensões não são conhecidos a priori, por tanto dependem de uma estrutura de dados que no qual eles possam ser adicionados dinamicamente, quando conhecidos. Como a serialização dos metamodelos EMF é feita por meio de atributos e por padrão o EMF não suporta adição de atributos em objetos em tempo de execução, foram criados dois tipos de dados: `JavaMap` e `JavaSortedMap`. Ambos encapsulam estruturas do tipo `java.util.Map`, o qual é compatível com a forma de serialização do emf-json (emf-json, 2016). Apesar do EMF possuir o tipo `EMap`, o mesmo não é serializável, ao passo que o Map é compatível com a serialização suportada pelo emf-json, framework utilizado para fazer as conversões entre EMF e JSON. Uma vez que o projeto esteja importado, o usuário deve informar a visão e o indicador, conforme ilustrado na figura 25. Após a obtenção das visões em sua configuração padrão, é possível refazer o mapeamento de propriedades da visão de acordo com a categoria de dados. O Sourceminer Trends disponibiliza inicialmente três visões: Treemap, Linear e Grafo. Elas representam, respectivamente, as estruturas de dados `HierarquicalStructure`, `DimensionalStructure` e `NetworkStructure`, herdando suas características e casos de uso. Por padrão, o Treemap, Line e Network - visões utilizadas na interface - são mapeados de acordo com as tabelas 6, 7 e 8. Todavia, estas tabelas devem ser expandidas diretamente pelo usuário.

Tabela 6 - Mapeamento Visual Padrão para Treemap

Atributo	Categoria	Descrição
`Id`	NOMINAL	Níveis da Hierarquia
`Type`	NOMINAL	Treemap

Tabela 7 - Mapeamento Visual Padrão para Visão Line

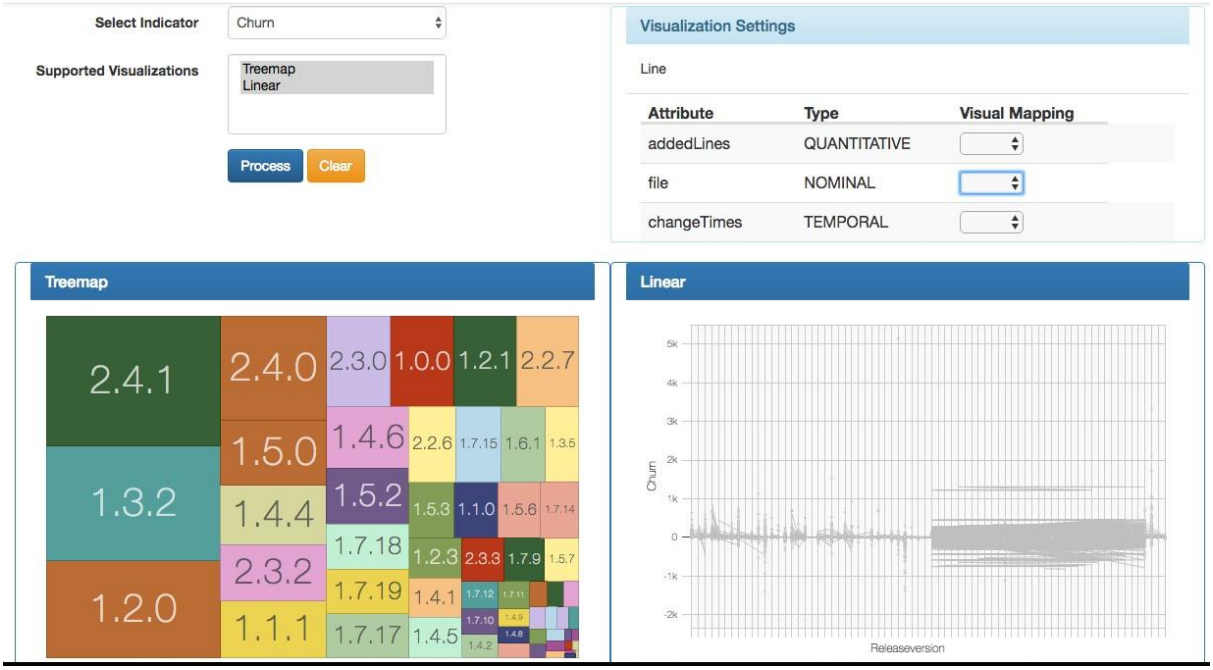
Atributo	Categoria	Descrição
Id	NOMINAL	Ponto de interseção entre as
Color	NOMINAL	Cor da linha
Type	NOMINAL	Line
X	QUANTITATIVO	Valor para coordenada x
Y	QUANTITATIVO	Valor para coordenada y

Tabela 8 - Mapeamento Visual Padrão para Visão Network

Atributo	Categoria	Descrição
Id	NOMINAL	Identificador do nó
Edges	NOMINAL	Conexões
Tipo	NOMINAL	Network

As visões são ilustradas na figura 25.

Figura 25 - Tela de Análise



A visões mapeadas foram:

- Treemap: Objetivo: mapear churn de cada release, com possibilidade de detalhamento;

Descrição: representar dados agrupados, detalhando-os através de navegação.

- Line: Objetivo: mapear atividade dos desenvolvedores em cada release, com possibilidade de detalhamento; Descrição: representar dados em sequência lógica.

6.4 RESULTADOS E OBSERVAÇÕES

Observou-se que a utilização de modelos facilitou a adoção de uma solução incremental, que foi adaptada de acordo com os problemas decorrentes da aplicação dos exemplos de uso. As alterações incrementais se deram na maior parte do tempo nos modelos e não no código gerado para eles. Os modelos também facilitaram a serialização e a interpretação dos dados armazenados no CouchDB pelo fato das manipulações terem sido realizadas em objetos com interface definida. Deste modo não foi necessário utilizar técnicas de *parsing* de texto. Por outro lado, a serialização baseada em EMF necessita de tratamentos adicionais quando o conjunto de dados a ser analisado é demasiado grande. Finalmente, durante o experimento foi notado que filtrar as releases pela data de seu lançamento causava uma distorção na extração. Para evitar que commits fora do intervalo de datas entrassem no escopo da release, o filtro também foi aplicado aos commits.

6.4.1 Resposta às Questões de Pesquisa

A partir do estudo de caso, as questões de pesquisa deste trabalho foram respondidas da seguinte forma:

QPD1: *qual proposta de metamodelo pode ser utilizada para instanciar dados de atributos de evolução para alimentar metáforas visuais?*

Um modelo inspirado na API JGit para representação de releases, commits e seus dados associados

QPD2: *como identificar quais metáforas visuais em ambientes interativos baseados em múltiplas visões podem ser utilizadas para a visualização de atributos de evolução de software?*

As metáforas visuais devem estar associadas com a natureza dos dados analisados.

QPD3: *como os objetivos de compreensão de atributos de evolução de software podem ser mapeados para atributos visuais?*

O mapeamento de atributos reais nas categorias NOMINAL, ORDINAL, QUANTITATIVE, TEMPORAL sugere que atributos de metáforas visuais classificados de forma correspondente são mais adequados para visualização.

QPD4: *quais evidências podem ser utilizadas para ilustrar a efetividade da compreensão de atributos de evolução de software no contexto de projetos de software reais?*

O mapeamento adotado neste trabalho permite que o usuário ajuste às metáforas visuais de acordo com sua conveniência, desde que atributos reais e visuais estejam classificados analogamente.

6.5 AMEAÇAS À VALIDADE

Tanto o desenvolvimento do Sourceminer Trends quanto sua análise foram realizados pelo mesmo pesquisador. Numa segunda fase esperava-se envolver outros pesquisadores no estudo. Também não foi adotado um modelo para as métricas, que foram derivadas empíricas. O presente trabalho carece de estudos relacionados a modelos para métricas. O mapeamento para categorias deve ser expandindo, principalmente no que se refere a listas. Para a visão grafo, por exemplo, o atributo `edges` esperava um conjunto de objetos com diferentes atributos.

6.6 CONCLUSÃO

Este capítulo buscou ilustrar a utilização do Sourceminer Trends como uma infraestrutura baseada em modelos para construção de ferramentas de análise de evolução interativas. Adotou-se os conceitos de Churn e Atividades dos Desenvolvedores como atributos a serem estudados, utilizando-se visões consagradas na literatura como o Treemap e o Grafo. Neste estudo, foram utilizados modelos de evolução e visual para mapear os dados obtidos através do sistema de controle de versão e exportá-los no formato JSON. Também foi realizada uma classificação dos atributos dos elementos retornados para as visões, as quais

podem utilizar essa categorização para fornecer uma interação mais conveniente para o usuário.

7 CONCLUSÃO E PERSPECTIVAS FUTURAS

Este capítulo visa apresentar as considerações finais desta dissertação, juntamente com suas contribuições e limitações. Também são apresentados os trabalhos em andamento e as próximas atividades previstas com a ferramenta e as aplicações desenvolvidas no âmbito deste trabalho.

7.1 CONSIDERAÇÕES FINAIS

Evolução de software é um ramo do conhecimento que vem sendo estudado por meio de diferentes abordagens. Visualização de software é uma delas e vem sendo amplamente pesquisada pelo potencial de racionalizar uma grande quantidade de informação num espaço reduzido e de fácil leitura. Outras técnicas dependem da interpretação de textos ou análise estatística, que nem sempre são adequadas para o usuário comum. Os ambientes de visualização, porém, apresentam deficiências. Um dos principais problemas é significar os atributos visuais de forma adequada. Ademais, as soluções adotadas costumam abordar o mapeamento de atributos reais e visuais como uma questão de implementação. Nesse sentido, as visualizações propostas não são facilmente comparáveis entre si, o que dificulta a validação das soluções. O objetivo do Sourceminer Trends é fornecer uma infraestrutura comum para a análise de evolução de software baseada em visualização. Para tanto, são disponibilizados modelos tanto para representação de dados de evolução quanto para as visões, que podem ser utilizados por pesquisadores para gerar suas próprias ferramentas. O presente trabalho também desenvolveu uma ferramenta de visualização de evolução, baseada em múltiplas visões interativas, com o objetivo de demonstrar as capacidades da ferramenta.

7.2 CONTRIBUIÇÕES

As contribuições deste trabalho são a apresentação de modelos para representação de dados de evolução de software, bem como para representação visual, ambos serializáveis no formato JSON e disponíveis no formato ecore para utilização de pesquisadores. Além disso, a ferramenta introduz o conceito de mapeamento entre atributos reais e visuais. Estas duas propostas tem o potencial de facilitar a pesquisa em visualização de evolução de software.

7.3 LIÇÕES APRENDIDAS

Os tópicos seguintes se referem ao aprendizado obtido com o estudo dirigido:

- As estruturas visuais devem ser projetadas para que a serialização seja compatível com o toolkit de visualização;
- Definir um relacionamento como composição ou associação no EMF afeta drasticamente a serialização;
- Para a estrutura visuais Grafo, o conceito de ter um único dataset para todas as visões se mostrou ineficaz. Foi criado um dataset específico para facilitar o mapeamento das conexões;
- Coleções no EMF não são serializáveis por padrão. Tipos de dados específicos foram criados para geração de dados no formato correto;
- Modelos EMF não suportam o conceito de adição dinâmica de campos;
- Modelos EMF não suportam a sobrescrita de `equals/hashcode`, tornando-os incompatíveis para uso em `Hashs` ou `Sets`.

7.4 LIMITAÇÕES

Uma limitação técnica deste trabalho é a falta de *lazy loading* ao realizar a busca de um documento no formato EMF armazenado no CouchDB. Isso se deve a uma limitação do driver utilizado. Também não foi possível utilizar a geração do modelo KDM pelo Modisco, também devido a uma limitação do mesmo driver. Por fim, a extração das métricas pode ser adaptada para o uso de modelos, porém no momento da escrita deste artigo possui uma estrutura fechada e pré-definida.

7.5 TRABALHOS EM ANDAMENTO

O planejamento para o projeto é utilizar outros drivers para expandir o mecanismo de armazenamento para mais soluções de banco de dados. O NeoEMF chegou a ser considerado para este trabalho, no entanto se mostrou instável e foi descartado. Outro ponto a ser trabalhado é utilizar o potencial de operações `map/reduce` típicas de bancos de dados NoSQL como o Hadoop e o próprio CouchDB. Por fim, planeja-se adaptar a solução para o Eclipse Che, uma versão do Eclipse adaptada para o contexto de *cloud computing.*

REFERÊNCIAS

ALWIS, B. D.; SILLITO, J. **Why are software projects moving from centralized to decentralized version control systems?** Cooperative and Human Aspects on Software Engineering, 2009. CHASE'09. ICSE Workshop on. [S.l.]: IEEE. 2009. p. 36-39.

ANGULAR JS. **Angular JS**, 2016. Disponível em: <https://angular.io>. Acesso em: 14 outubro 2015.

APACHE Subversion. **Apache™ Subversion®**, 2016. Disponível em: <https://subversion.apache.org>. Acesso em: 2 janeiro 2016.

BARMPIS, K.; KOLOVOS, D. **Hawk:** Towards a scalable model indexing architecture. Proceedings of the Workshop on Scalability in Model Driven Engineering. [S.l.]: ACM. 2013. p. 6.

BASILI, V. R. **Software modeling and measurement:** the goal/question/metric paradigm. [S.l.]: [s.n.]. 1992.

BENELALLAM, A. et al. **Neo4EMF, a scalable persistence layer for EMF models**. European Conference on Modelling Foundations and Applications. [S.l.]: Springer. 2014. p. 230-241.

BLACK, A. P. A. N. O. A. D. S. A. P. D. **Pharo by example**. [S.l.]: Lulu.com, 2010.

BRUNELIERE, H. et al. **MoDisco:** a generic and extensible framework for model driven reverse engineering. Proceedings of the IEEE/ACM international conference on Automated software engineering. [S.l.]: ACM. 2010. p. 173-174.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. **Readings in information visualization:** using vision to think. [S.l.]: Morgan Kaufmann, 1999.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. **Readings in information visualization:** using vision to think. [S.l.]: Morgan Kaufmann, 1999.

CARNEIRO, G. D. F. N.; MENDONÇA, M. G. D. **SourceMiner:** towards an extensible multi-perspective software visualization environment. International Conference on Enterprise Information Systems. [S.l.]: Springer. 2013. p. 242-263.

CARNEIRO, G.; MENDONÇA, M. **The Importance of Cognitive and Usability Elements in Designing Software Visualization Tools**. Proceedings of the 20th Annual Psychology of Programming Interest Group Conference. [S.l.]: [s.n.]. 2008. p. 12.

CHI, E. H. **A taxonomy of visualization techniques using the data state reference model**. Information Visualization, 2000. InfoVis 2000. IEEE Symposium on. [S.l.]: IEEE. 2000. p. 69-75.

COUCHDB. **Seamless multi-master sync, that scales from Big Data to Mobile, with an Intuitive HTTP/JSON API and designed for Reliability**, 2015. Disponível em: <http://couchdb.apache.org>. Acesso em: 13 abril 2015.

DATA Driven Documents. **Data Driven Documents**, 2016. Disponível em: <https://d3js.org>. Acesso em: 05 abril 2016.

DIEHL, S. **Software visualization:** visualizing the structure, behaviour, and evolution of software. [S.l.]: Springer Science \& Business Media, 2007.

DRIESSEN, V. A successful Git branching model, 2010. Disponível em: <http://nvie. com/posts/a-successful-git-branching-model>. Acesso em: 02 fevereiro 2016.

DUCASSE, S. et al. MSE and FAMIX 3.0: an interexchange format and source code model family, 2011.

DUCASSE, S.; GÎRBA, T.; FAVRE, J.-M. Modeling software evolution by treating history as a first class entity. **Electronic Notes in Theoretical Computer Science**, v. 127, n. 3, p. 75-86, 2005.

ECLIM. **Eclim**, 2016. Disponível em: <http://eclim.org>. Acesso em: 07 junho 2016.

ECLIPSE Development Process. **Eclipse Development Process**, 2016. Disponível em: <https://eclipse.org/projects/dev_process/development_process.php>. Acesso em: 19 novembro 2015.

EICK, S. G. et al. Does code decay? assessing the evidence from change management data. **IEEE Transactions on Software Engineering**, v. 27, n. 1, p. 1-12, 2001.

EMF-JSON. **emf-json**, 2016. Disponível em: <http://emfjson.org>. Acesso em: 10 abril 2016.

GENOVA, G. What is a metamodel: the OMG's metamodeling infrastructure. **Software and Systems Modeling**, v. 4, n. 2, p. 171-188, 2005.

GIT. **Git**, 2016. Disponível em: <https://git-scm.com>. Acesso em: 02 fevereiro 2016.

GITHUB. **Github**, 2016. Disponível em: <https://github.com>. Acesso em: 26 setembro 2015.

GOUSIOS, G.; PINZGER, M.; DEURSEN, A. V. **An exploratory study of the pull-based software development model**. Proceedings of the 36th International Conference on Software Engineering. [S.l.]: ACM. 2014. p. 345--355.

HALL, G. A.; MUNSON, J. C. Software evolution: code delta and code churn. **Journal of Systems and Software**, v. 54, n. 12, p. 111-118, 2000.

JGIT. **JGit**, 2016. Disponível em: <https://eclipse.org/jgit/>. Acesso em: 10 março 2016.

KDM Source Discoverer. **KDM Source Discoverer**, 2016. Disponível em: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.modisco.infra.omg.doc%2Fmediawiki%2Fkdm_source_discoverer%2Fuser.html>. Acesso em: 10 março 2016.

KIRK, A. **Data Visualization:** a successful design process. [S.l.]: Packt Publishing Ltd, 2012.

KOEGEL, M.; HELMING, J. **EMFStore:** a model repository for EMF models. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. [S.l.]: ACM. 2010. p. 307-308.

LEAVITT, N. Will NoSQL databases live up to their promise? **Computer**, v. 43, n. 2, p. 12-14, 2010.

LEHMAN, M. M. Programs, life cycles, and laws of software evolution. **Proceedings of the IEEE**, v. 68, n. 9, p. 1060-1076, 1980.

LEHMAN, M. M.; PERRY, D. E.; RAMIL, J. F. **Implications of evolution metrics on software maintenance**. Software Maintenance, 1998. Proceedings., International Conference on. [S.l.]: IEEE. 1998. p. 208-217.

LOELIGER, J.; MCCULLOUGH, M. **Version Control with Git:** Powerful tools and techniques for collaborative software development. [S.l.]: O'Reilly Media, Inc., 2012.

MAZZA, R. **Introduction to information visualization**. Springer Science & Business Media. [S.l.]: [s.n.]. 2009.

NOVAES, R. L. et al. Software evolution visualization: A systematic mapping study. **Information and Software Technology**, v. 55, n. 11, p. 1860-1883, 2013.

PÉREZ-CASTILLO, R.; DE GUZMAN, I. G.-R.; PIATTINI, M. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. **Computer Standards & Interfaces**, v. 33, n. 6, p. 519-532, 2011.

ROCHKIND, M. J. The source code control system. **IEEE Transactions on Software Engineering**, n. 4, p. 364-370, 1975.

SCHEIDGEN, M.; FISCHER, J. **Model-Based Mining of Source Code Repositories**. International Conference on System Analysis and Modeling. [S.l.]: Springer. 2014. p. 239-254.

SHIN, Y. et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. **IEEE Transactions on Software Engineering**, v. 37, n. 6, p. 772-787, 2011.

SHNEIDERMAN, B. **The eyes have it:** A task by data type taxonomy for information visualizations. Visual Languages, 1996. Proceedings., IEEE Symposium on. [S.l.]: IEEE. 1996. p. 336-343.

SILVA, A. N.; CARNEIRO, G. **Propondo uma Arquitetura para Ambientes Interativos baseados em Multiplas Visões**. II Workshop Brasileiro de Visualização de Software. [S.l.]: [s.n.]. 2012. p. 1-8.

STEINBERG, D. et al. **EMF:** eclipse modeling framework. [S.l.]: Pearson Education, 2008.

STEPPER, E. Connected data objects (cdo). **ttp://www.eclipse.org/cdo/documentation/index.php**, novembro 2012.

TICHY, W. F. RCS—a system for version control. **Software: Practice and Experience**, v. 15, n. 7, p. 637-654, 1985.