UNIVERSIDADE FEDERAL DE SANTA CATARINA

BR-Dia
Ferramenta CASE online para modelagem UML com integração a ferramentas de gerência de projetos e controle de versão.

Fábio César Ariati

Florianópolis - SC

2014 / 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

BR-Dia
Ferramenta CASE online para modelagem UML com integração a ferramentas de gerência de projetos e controle de versão.

Fábio César Ariati

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de informação.

Florianópolis - SC

2014 / 1

Dedico este trabalho aos meus pais e minha esposa esposa.

Agradecimentos

Agradeço a minha esposa e minha família pelo apoio, incentivo e paciência

Agradeço ao professor Ricardo pela dedicação e por mostrar o caminho para a realização do trabalho

Ao Roberto pelo apoio e ajuda

Aos membros da banca

Resumo

Ferramentas de modelagem gratuitas do mercado têm pouca ou nenhuma integração com o resto do processo de desenvolvimento de software e geralmente seus recursos avançados existem somente em versões pagas. Este trabalho se propõe construir uma ferramenta CASE livre baseada na Web para modelagem UML e outros diagramas relevantes. A ferramenta integrará os processos de desenvolvimento de software dando suporte à modelagem, geração de código, integração com ferramentas de gerência de projetos e integração com ferramentas controle de versão.

O sistema é composto por duas partes, o servidor, codificado na linguagem Ruby e o cliente, desenvolvido seguindo a especificação do HTML5, utilizando a linguagem Javascript para a interação com o usuário e SVG (Scalable Vector Graphics) para a renderização dos diagramas.

Palavras chave: Ferramentas CASE, UML, modelagem, gerência de projetos, Ruby, Javascript, HTML5, artefatos de software.

Abreviaturas e siglas

- CASE - Computer-aided software engineering
- UML – Unified Modeling Language
- W3C - World Wide Web Consortium
- HTML - HyperText Markup Language
- PHP - Hypertext Preprocessor
- SVG - Scalable Vector Graphics
- AJAX - Asynchronous Javascript and XML
- RoR - Ruby on Rails
- OMG - Object Management Group
- OMT - Object Modeling Technique
- OOSE - Object-oriented software engineering
- ISO - International Organization for Standardization
- IDE - Integrated Development Environment)
- ER – Entidade Relacionamento
- JSON –JavaScript Object Notation
- API - Application Programming Interface
- AJAX - Asynchronous Javascript and XML
- REST - Representational state transfer

Sumário

Índice de figuras

Índice de tabelas

1 Introdução

Ferramentas CASE modernas possuem recursos que facilitam o trabalho do desenvolvedor, como geração de código a partir de diagramas, geração de base de dados, colaboração, prototipagem de telas, entre outros (PRESSMAN, 2001). Mas estas ferramentas possuem recursos avançados somente suas versões pagas e possuem pouca integração com o resto do processo de desenvolvimento, prejudicando a rastreabilidade entre os artefatos do projeto, a qual pode ser atingida com recursos que promovam a reusabilidade e a geração de artefatos padronizados (ALMEIDA; RAMOS; F. NETO, 2010).

1.1 Tema

Este trabalho tem contexto prático e visa o desenvolvimento de um software que consiste em uma ferramenta para modelagem UML 2, que seja extensível, integrável com outras ferramentas e que siga os padrões da W3C (2013), focados em navegadores modernos e atualizados. A integração ocorrerá com o Github e o Redmine. O objetivo é que a ferramenta tenha sua interface com o usuário totalmente desacoplada do *backend* que roda no servidor, assim é possível ter servidores que rodem com diferentes linguagens (PHP, Ruby, Java, C#), funcionando com a mesma interface feita em HTML e JavaScript, e mantendo o mesmo padrão JSON para a troca de dados. Assim, as responsabilidades de integração com outras ferramentas ficam com o *backend.*

1.2 O problema

Existem diversas ferramentas CASE para modelagem de software no mercado, tanto proprietárias quanto livres. Existem ferramentas dedicadas e aquelas que são

integradas na IDE. Também existem ferramentas que rodam na Web, em Flash e HTML. Em Case-tools.org (2013), um fórum dedicado à discussão sobre ferramentas CASE de diversos gêneros, há uma lista exibida em formato hierárquico de pastas e com uma lista bastante grande de ferramentas. Analisando no geral as ferramentas de modelagem constata-se que há poucas ferramentas adequadas ao uso acadêmico e poucas que rodem na Web, um exemplo é o Gliffy (2013), o qual roda na Web mas não é dedicado a projetos de software, então é limitado neste quesito, pois possui suporte apenas para diagramas da UML 1. Recursos avançados geralmente são encontrados em versões pagas, as quais, pelo fato de não serem livre, não são adequadas ao uso acadêmico que explora o uso, construção e pesquisa sobre ferramentas CASE. Processos de engenharia de software são distintos e evoluem com o tempo e as ferramentas precisam ser variadas e acompanhar essa evolução para suprir as necessidades, considerando isto, é interessante ter uma ferramenta que seja fácil expandir para atender os mais variados tipos de projetos.

1.3 Motivação e Justificativa

Um software livre que rode no navegador e com recursos de integração, pode ser útil em diversos contextos, como no uso acadêmico, por exemplo. Pelo fato do software ter o código fonte aberto, pode se abrir caminho para implementações de *backend* voltadas para outras linguagens de programação, outros processos de desenvolvimento ou outros tipos de diagramas. O fato é que poucas ferramentas gratuitas oferecem esta característica, e se pensarmos em ferramentas que rodem no navegador e seguindo especificações do HTML5 (pois a maioria ainda roda em flash, o qual não é um padrão de Web), o número diminui ainda mais.

Uma vantagem de se ter uma ferramenta CASE na Web para uso acadêmico seria pela facilidade de proporcionar uma ferramenta acessível para os alunos sem precisarem baixar e instalar nada, bastaria ter um navegador (podendo ser inclusive em

um *tablet*), seria possível utilizar inclusive o cadastro existente na instituição de ensino para acessar o sistema. Outra vantagem é a facilidade de integração com outras ferramentas, visto que uma ferramenta Web pode facilmente se integrar com outra ferramenta através de Web Services ou como é o caso do escopo deste trabalho, através de integração direta com outra ferramenta Web, que neste caso é via *plugin* do Redmine. No campo da pesquisa acadêmica, a ferramenta pode ser útil pelo fato de ser software livre e seguir padrões da Web, o que facilita a portabilidade e integração. Essas características fornecem subsídio para aprimoramentos e estudos da ferramenta. A flexibilidade da ferramenta pode proporcionar, no âmbito acadêmico a exploração de problemáticas como a falta de uma notação de alto nível que atenda as necessidades específicas de projetos baseados em frameworks e componentes, as quais foram abordadas na tese de doutorado de Ricardo Pereira e Silva (SILVA, 2000). Uma ferramenta flexível pode incorporar outros conceitos não previstos na modelagem tradicional.

1.4 Método de pesquisa

Este trabalho consiste no desenvolvimento de um software, portanto do ponto de vista da sua natureza, uma Pesquisa Aplicada. Do ponto de vista do objetivo, a pesquisa é exploratória, pois estuda a relação de uma ferramenta CASE com as ferramentas utilizadas no resto do processo de desenvolvimento, a fim de desenvolver uma ferramenta útil no âmbito acadêmico e na comunidade de software livre.

1.5 Estrutura do documento

Este documento está dividido em três partes. A parte conceitual , que trata dos conceitos, definições e fundamentação teórica que vai do capítulo 2 até o 5. A parte prática que fala da arquitetura, do código, do uso e das características do sistema, a

qual vai do capítulo 6 até o 8. E parte de conclusões e resultados, a qual engloba o capítulo 9 e o capítulo 10. O apêndice A contém os diagramas do projeto, o apêndice B é dedicado a explicar as decisões e o processo de se construir uma aplicação gráfica em HTML5. E o apêndice C explica passo a passo como se adicionar novos diagramas na ferramenta BR-Dia.

2 Modelagem de Software

Modelagem de software é a atividade de desenvolvimento de modelos que descrevam as características ou o comportamento de um software de maneira simplificada e mais natural de ser entendida que apenas código. O mais comum na modelagem, é utilizar algum tipo de notação gráfica que represente os artefatos de software e suas relações.

O sistema deste trabalho será capaz de modelar diagramas em UML 2 e fornecerá subsídio para a fácil adição de novos tipos de diagramas. A seguir é apresentada a UML e é feito a descrição dos diagramas inicialmente propostos para este trabalho.

2.1 UML

UML é uma linguagem visual para especificação, construção e documentação de artefatos de sistemas de software (OMG, 2011). É uma linguagem de modelagem que utiliza os princípios da orientação a objetos e, segundo OMG (2011), tem por objetivo fornecer aos arquitetos, engenheiros e desenvolvedores, ferramentas de análise para projeto e implementação de sistemas baseados em software. Mas pode servir também para modelagem de negócios e processos similares. A primeira versão da UML foi originada dos três principais métodos orientados a objetos da época (Booch, OMT, e OOSE), e incorporou práticas de projeto de linguagem de modelagem, programação orientada a objetos e linguagens de descrição de arquitetura. Comparando a primeira versão de UML com a versão 2.4.1, a qual é referência para este trabalho, a versão 2.4.1 foi aprimorada com uma estrutura de linguagem mais modular e maior capacidade para modelagem de sistemas de grande porte. A ferramenta proposta neste trabalho permitirá, inicialmente como requisito mínimo, a edição dos diagramas de

classe, máquina de estados, casos de uso, sequência e atividades. Os quais são apresentados a seguir.

2.1.1 Diagrama de Classes

Um diagrama de classes exibe elementos de modelos estáticos, como classes, interfaces e seus relacionamentos (LARMAN, 2004). É o diagrama que mais representa e estrutura de um código orientado a objetos, o que o torna adequado para gerar código.

A figura 1 exemplifica um diagrama de classes, o elemento “classe” é representado por um retângulo de três partes, uma para o nome e esterótipo, se houver, uma para os atributos e outra para os métodos. Os relacionamentos são representados por ligações que tem as pontas em formatos distintos conforme o tipo do relacionamento.

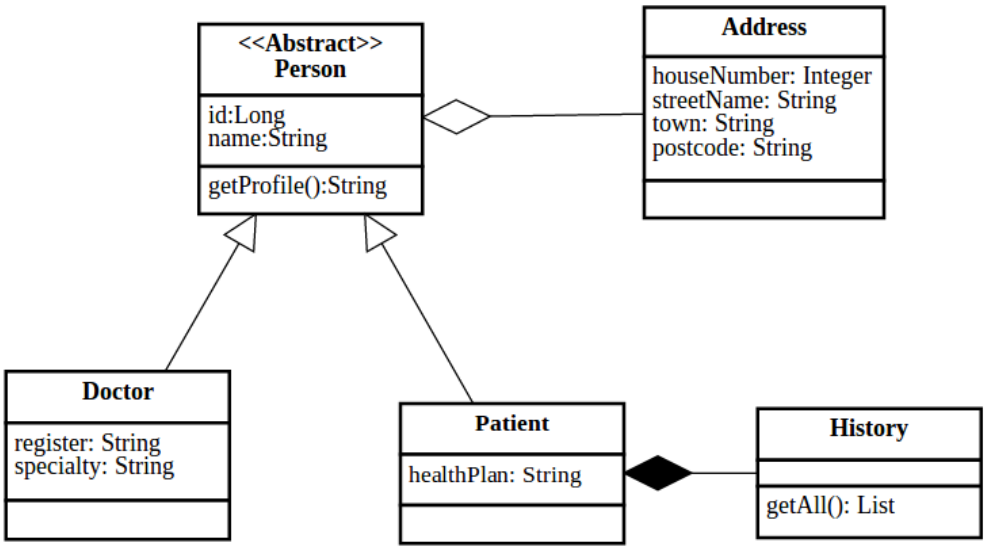

Figura 1: Diagrama de classes

2.1.2 Diagrama de Máquina de Estados

Descreve os estados e sub-estados de um objeto, assim como os eventos e as transições entre os estados. É a representação da situação em que o objeto se encontra a cada transição no decorrer da execução do programa.

O elemento 'estado' é representado por um retângulo arredondado que pode ser dividido em duas partes e as transições são representadas por setas. Este retângulo pode ter internamente outros retângulos representando sub-estados.

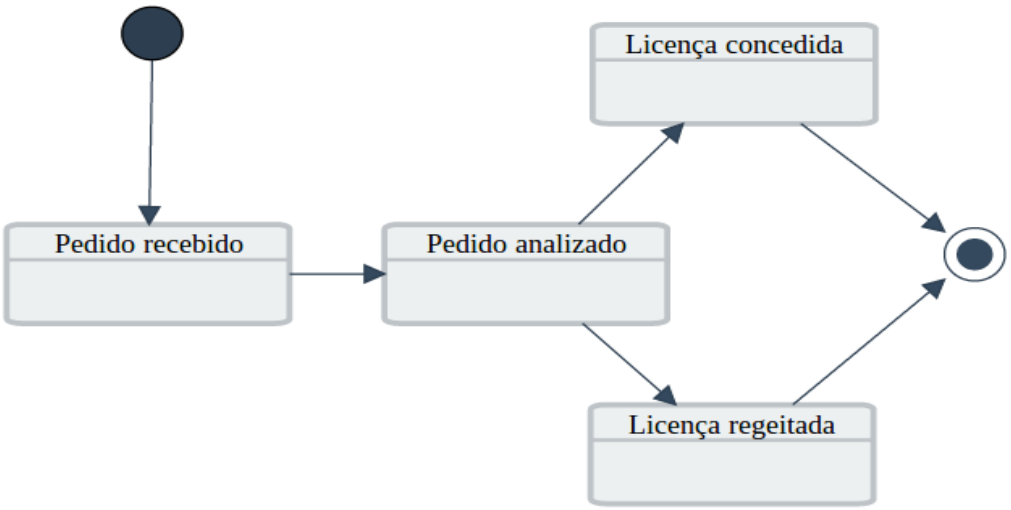

Figura 2: Diagrama de máquina de estados

2.1.3 Diagrama de Casos de Uso

Descreve casos de uso, atores, suas inter-relações, além de relacionamentos de dependência, generalização e associação. Casos de uso representam unidades de interação entre um usuário (humano ou máquina) e o sistema. Um exemplo de caso de uso é a operação de compra em um sistema de comércio eletrônico (Figura 3).

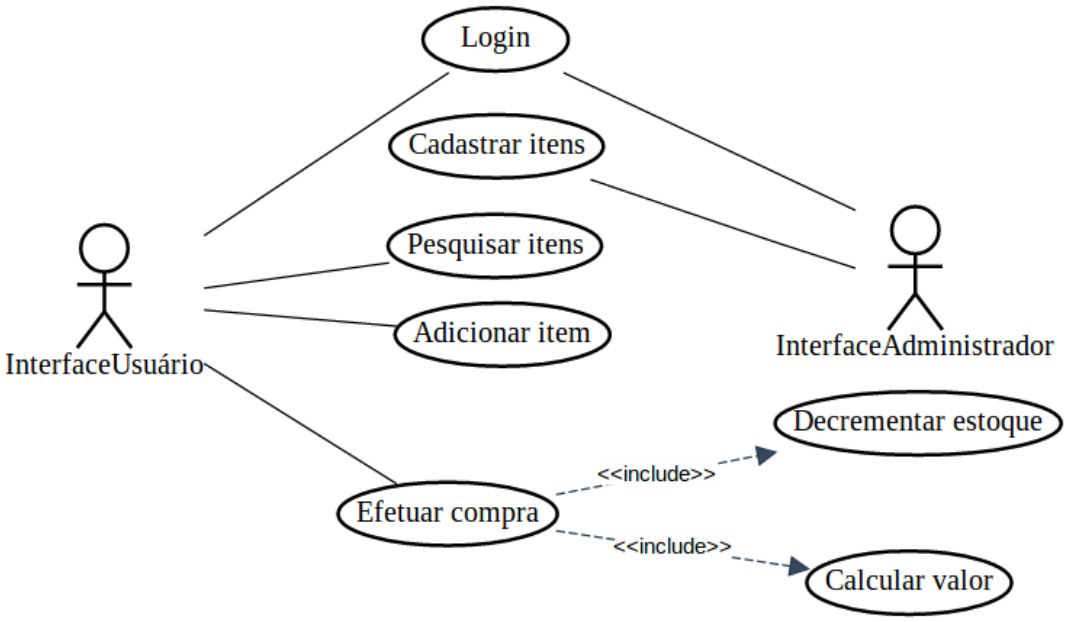

Figura 3: Diagrama de casos de uso

2.1.4 Diagrama de Sequência

Representa a sequência das mensagens trocadas entre objetos no decorrer do tempo para a realização de alguma operação. Projetos grandes podem ter métodos espalhados por diferentes classes, o diagrama de sequência modela de uma forma simples, gráfica e lógica essa informação.

Em diagramas de sequência (figura 4), linhas de existência representam objetos. Mensagens são enviadas de um objeto até o outro através de setas com informação textual.

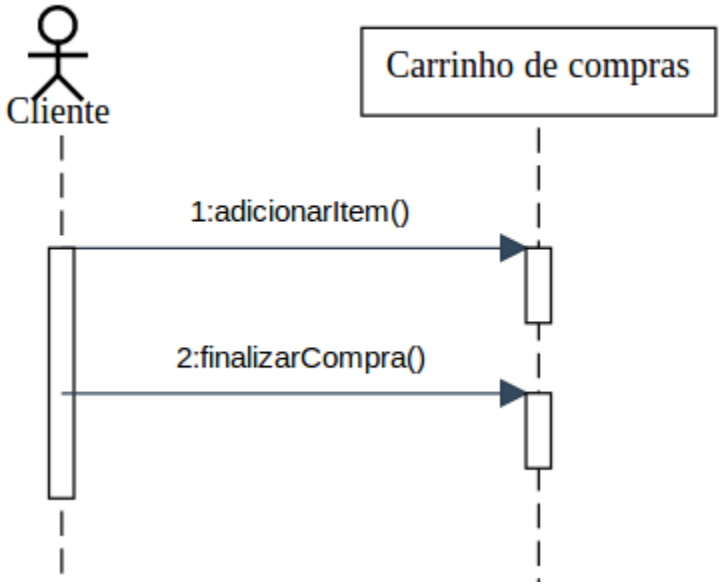

Figura 4: Diagrama de sequência

2.1.5 Diagrama de Atividades

O diagrama de atividades modela aspectos dinâmicos de um sistema. Descreve os passos a serem seguidos, para a execução de uma atividade. Essa descrição é feita mostrando o fluxo das atividades.

O diagrama de atividades (Figura 5) é composto por atividades, transições e objetos.

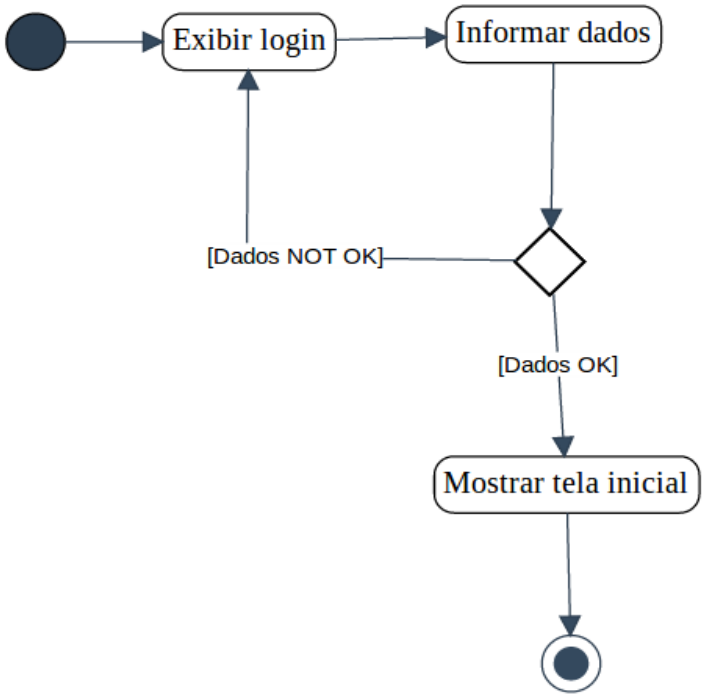

Figura 5: Diagrama de atividades

3 Ferramentas CASE

Ferramentas CASE (Computer Aided Software Engeneering), em português, “Engenharia de Software Auxiliada por Computador” são softwares que prestam auxílio em atividades de engenharia de software como análise de requisitos, modelagem, programação, prototipagem e testes (PRESSMAN, 2001). Esse tipo de ferramenta se destina a gerar produtos de software de qualidade e fácil manutenção. Mas o uso de ferramenta CASE deve ser analisado. Chinubhai (2011) conclui em seu artigo que o uso de ferramentas CASE pode levar à diminuição da eficiência em equipes pequenas. Considerando que equipes pequenas normalmente executam projetos menores e os mesmos têm menor complexidade, o uso dessas ferramentas pode adicionar mais complexidade a um projeto que teria sido simples de executar, visto que a ferramenta tem seu próprio domínio de conhecimento adicionado ao projeto e a consequência desta complexidade é menor eficiência. Por isso, a escolha e a necessidade de se utilizar uma ferramenta para determinada atividade é importante e não é trivial. A seguir, algumas vantagens e desvantagens de ferramentas CASE.

3.1 Vantagens

- **Qualidade no produto final:** Ferramentas CASE apoiam e fornecem funcionalidades como aplicação de padrões, controle de qualidade e apoio a documentação, modelagem e testes.
- **Produtividade:** Ferramentas CASE podem fornecer automação e reduzir o tempo para concluir diversos tipos de tarefas como geração de código-fonte, geração de interface gráfica, geração de base de dados e análise de qualidade.

- **Agilidade para tomada de decisão:** Ao facilitar processos e automatizar tarefas, as ferramentas fazem com que as pessoas fiquem liberadas para outras tarefas que exigem inteligência e tomada de decisão.
- **Melhoria e agilidade na manutenção:** É possível ter mais informações sobre o software na hora de realizar manutenção, resultado de documentação apoiada por ferramentas. E, assim como auxiliam no desenvolvimento, as ferramentas CASE fornecem agilidade e qualidade nas tarefas de manutenção.

3.2 Desvantagens

- **Custo:** Grande parte das ferramentas de qualidade são pagas, algumas oferecem versões gratuitas com limitações qualitativas e quantitativas.
- **Adição de complexidade em alguns casos:** Uma ferramenta CASE tem seu próprio domínio de conhecimento, o qual pode deixar um projeto pequeno consideravelmente mais complexo.
- **Treinamento para utilização:** A produtividade do desenvolvedor pode cair na fase inicial de implementação, pois é necessário tempo para o aprendizado do uso da ferramenta.

3.3 Classificação

Ferramentas CASE podem ser classificadas pela funcionalidade, pelo papel como instrumento para gestores ou técnicos e pelo uso nas etapas do processo de engenharia (PRESSMAN, 2001). Se formos classificar por funcionalidade, a variedade é grande, Pressman (2001) compôs uma lista bastante ampla. A tabela a seguir exibe a classificação por funcionalidade de uma forma menos fragmentada que a lista de Pressman.

	Descrição	Exemplos
Gerência e planejamento	Dão apoio a parte gerencial do projeto	Ferramentas de estimação de custos, de gerência de projetos
Modelagem	Modelagem utilizando notação gráfica	Editores de diagramas
Gerenciamento de configuração	Fornece apoio ao desenvolvimento ajudando no controle de mudanças	Sistemas de controle de versão e sistemas de gerenciamento de mudança
Prototipagem	Apoiam a criação de protótipos.	Editores e geradores de interface gráfica
Suporte a programação	Dão suporte a codificação, depuração, documentação e testes	IDEs, compiladores, editores.
Análise de programas	Inspecionam o código fonte e o fluxo do programa. Apoiam a detecção de vulnerabilidades, código duplicado, etc	Analisadores estáticos e dinâmicos
Reengenharia	Apoiam engenharia reversa, reestruturação de software	Sistemas que convertem código-fonte em modelagem

Tabela 1: Classificação de ferramentas CASE por funcionalidade

As ferramentas CASE também podem ser classificadas quanto à etapa do desenvolvimento, e são divididas em três categorias: *Upper CASE, Lower CASE* e Ferramentas CASE integradas (PI; SU; WANG, 1998).

- ***Upper CASE*:** São ferramentas voltadas para as fases de planejamento, análise e projeto. Fornecem planejamento estratégico, gerenciamento de metas e riscos

e suporte a construção de representações gráficas, como a modelagem através de diagramas.

- ***Lower CASE*:** Utilizadas nas fases posteriores do ciclo de vida do software, como codificação, testes, manutenção e outras atividades relacionadas à construção física (produção do software) dos sistemas.

- ***Ferramentas CASE integradas*:** Atuam em todo o ciclo de vida do software. Podem ser definidas como um conjunto de ferramentas que agem de forma combinada integrando ferramentas *Upper CASE* e *Lower CASE*. Na classificação por funcionalidade, as ferramentas integradas se situam em mais de uma categoria.

3.4 Ferramentas CASE para modelagem de software

Geralmente no processo modelagem de software se utiliza alguma notação gráfica que pode ser apoiada e produzida por ferramentas CASE. O objetivo das ferramentas de modelagem é facilitar a criação de modelos utilizando alguma notação, no caso deste trabalho, UML.

As ferramentas descritas nos exemplos a seguir são voltadas para modelagem orientada a objetos utilizando UML, mas algumas permitem outras funções como geração de esqueleto de código-fonte a partir de diagramas UML, geração esquemas banco de dados a partir do modelo ER, colaboração online entre times, e mapeamento de código para gerar diagramas.

3.3.1 Exemplos de ferramentas para modelagem com UML

Livres

- Argo UML: Ferramenta livre feita em Java para modelagem UML que trabalha com diversos formatos de arquivos e tem suporte a geração de código;

- Violet UML Editor: Ferramenta livre feita em Java para modelagem UML que oferece integração com a IDE Eclipse;
- Star UML: Desenvolvido para ser rápido e flexível, funciona somente em ambiente Windows;
- Umbrello UML Modeller: Ferramenta baseada na tecnologia KDE;
- Open ModelSphere: Ferramenta para Windows que suporta UML e outros tipos de diagramas;
- Modelio: Ferramenta para Windows, Linux e Mac que suporta diversos padrões e recursos avançados;

Proprietárias

- Rational Software Architect: Ferramenta paga feita pela IBM que suporta diversos diagramas;
- Visual Paradigm: Ferramenta paga para diagramas UML e outros diagramas relacionados com o processo de desenvolvimento de software;
- Astah: Ferramenta em Java para diagramas UML com versões paga e gratuita;
- Gliffy: Ferramenta online feita com HTML e javascript para diagramas UML e outros diagramas;
- Draw.io: Ferramenta online feita com HTML e javascript para diagramas UML e outros diagramas como o de Entidade-relacionamento e possui integração com Google Drive e Drop Box;
- UModel: Ferramenta para modelagem UML que oferece opções de geração de código-fonte para Java, C#, ou Visual Basic .NET;
- Enterprise Architect: Ferramenta da empresa Sparx Systems que suporta UML e outros diagramas;

4 O framework OCEAN

O framework OCEAN foi criado por Ricardo Pereira e Silva em sua tese de doutorado para ser flexível e possibilitar o desenvolvimento de ambientes que manipulem diferentes estruturas de especificações de projetos (SILVA, 2000). O framework suporta a construção de ambientes de desenvolvimento que manipulam especificações de projetos de software. A partir da especificação do projeto, utilizando UML, pode realizar funções como geração de código e verificação de consistência.

A figura 6 apresenta o metamodelo do framework OCEAN.

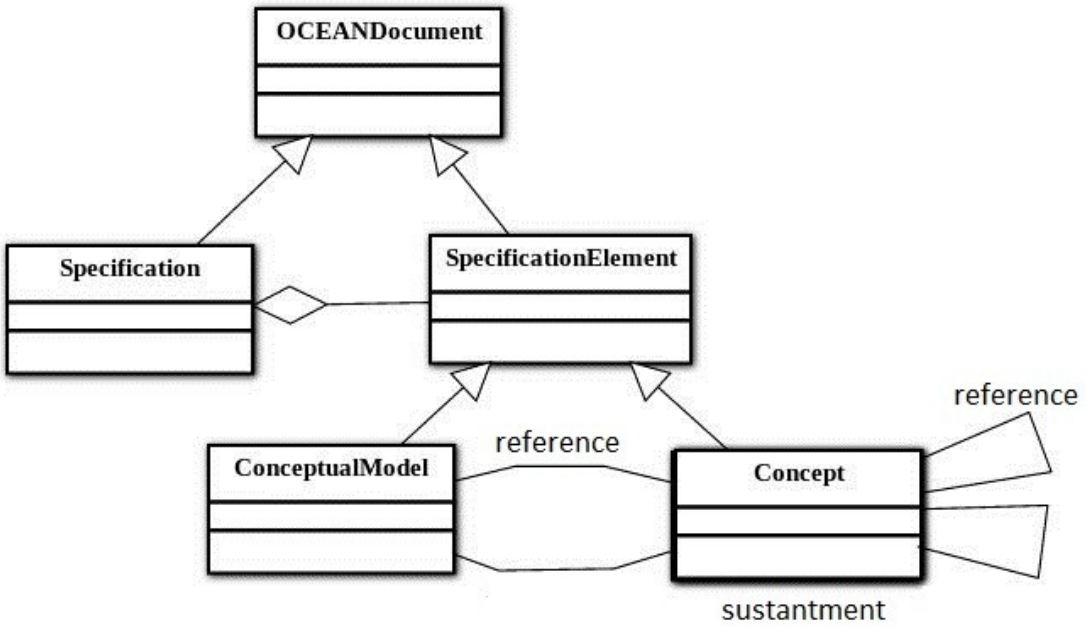

Figura 6: Metamodelo do framework OCEAN

Definições dos elementos do metamodelo da figura 6:

- OCEANDocument - Documento Genérico;
- Specification - “A estrutura de uma especificação é definida pelos tipos de modelos e conceitos tratados e também pelo registro das associações semânticas entre estes elementos” (SILVA, 2000);
- SpecificationElement - Podem ser modelos ou conceitos;
- Concept - Exemplo de instâncias: classe, caso de uso;
- ConceptualModel - Exemplo de instâncias: Diagrama de classe, Diagrama de Máquina de Estado.

A base para a modelagem de dados da ferramenta BR-Dia é baseada no metamodelo do framework OCEAN, o qual ajudou a fornecer um ponto de partida para a construção das superclasses da ferramenta e generalização dos conceitos relacionados com a construção dos diagramas.

5 Tecnologias e Ferramentas

Este capítulo apresenta as tecnologias, recursos e ferramentas que foram utilizados para a elaboração do software construído neste trabalho. Também é apresentado a fundamentação e motivação das escolhas para a produção do software. No Apêndice C deste trabalho tem um texto que também aborda sobre tecnologias para aplicações gráficas na Web, mas de forma mais generalizada.

5.1 HTML5

HTML(Hypertext Markup Language) é linguagem de marcação utilizada para produzir páginas na Web. Foi desenvolvida para ser independente de plataforma. O mesmo código HTML deve ser lido por diversos dispositivos, ao invés do código ter que se adaptar a cada dispositivo.

HTML5 é a quinta versão do HTML. A versão anterior, a 4.01, foi desenvolvida em 1999 e a internet mudou significativamente desde então (W3C, 2014). O HTML4 não tem suporte para semântica e também não facilita a manipulação dos elementos via JavaScript ou CSS. Considerando isto, podemos entender a importância do HTML5. O padrão incorpora diversos elementos como vídeo, áudio, gráficos vetoriais e renderização 3d, recursos anteriormente restrito à plugins como o Flash. O HTML5 foi desenvolvido levando em consideração a possibilidade de executar em dispositivos de baixo consumo de energia como smartphones e tablets.

Segundo a W3C, consórcio internacional voltado à padronização da Web, HTML5 incorpora as seguintes novidades:

- **Novos elementos:** *Tags* para exibição de diversos tipos de conteúdos como audio, vídeo, imagens, entre outros;

- **Novos atributos:** Novos atributos para formulários e outros elementos, como por exemplo, o atributo type, que ganhou novos elementos para formulários como: *range, e-mail, tel, week, datetime e placeholder*;
- **Semântica:** Suporte maior para a semântica dos elementos, por exemplo, ao especificar que um campo de entrada em um formulário é do tipo Data, o browser pode exibir um calendário quando o usuário entrar com o conteúdo;
- **Suporte completo ao CSS3:** Suporte à nova versão da folha de estilos da Web, o CSS (Cascading Style Sheets), o qual possibilita diversos efeitos e estilos como sombra em texto e outros elementos, efeitos de transição e quadros com cantos arredondados;
- **Gráficos 2D/3D:** Gráficos 2D e 3D com o Canvas ou CSS3 e gráficos 2D vetoriais com SVG;
- **Armazenamento local:** Dado é armazenado localmente no navegador e fica disponível sem depender da sessão, contrapondo o armazenamento por sessão;
- **Banco de dados SQL local:** O navegador utiliza SQLite3 como a engine de banco de dados, criando um banco de dados SQL localmente no navegador;
- **Aplicações Web:** Pensado para permitir a construção de aplicações Web complexas com suporte à recursos avançados como *workers*, armazenamento no cliente, acesso à arquivo local e SQL no cliente.

Gradualmente o HTML5 vai ganhando adesão, as implementações nos *browsers e* o seu uso em sites grandes como a BBC ou a Apple são fatores que impulsionam o interesse dos desenvolvedores.

A ferramenta produzida neste trabalho faz uso do HTML5 principalmente na parte de gráficos vetoriais utilizando SVG(Scalable Vector Graphics), que pode ser traduzido como gráficos vetoriais escaláveis e na especificação de estilos com CSS3.

No início deste trabalho houveram duas alternativas do padrão HTML5 para desenhar os diagramas, o SVG e o Canvas. Optou-se pelo SVG porque seus desenhos não perdem a qualidade com o zoom e é ele indicado para desenhar figuras simples como ícones e elementos de diagramas e o Canvas é mais indicado para gráficos complexos como jogos e aplicações 3d.

5.2 A linguagem Ruby

Ruby é uma linguagem de programação dinâmica, open source e focada em simplicidade e produtividade. Tem uma sintaxe elegante de leitura natural e fácil escrita. O criador da linguagem, Yukihiro “Matz” Matsumoto, se inspirou em suas linguagens favoritas (Perl, Smalltalk, Eiffel, Ada, e Lisp) para formar uma linguagem que equilibra a programação funcional com a programação imperativa (RUBY.ORG, 2014). Ruby é puramente orientada a objetos, mas permite programação funcional e tem poderosos recursos voltados para meta-programação, os quais podem ser utilizados para criar DSLs(linguagens específicas de domínio).

Segundo Flanagan e Matsumoto (2008) Ruby é uma linguagem dinâmica com uma gramática complexa, mas expressiva e possui uma biblioteca de classes com uma API rica e poderosa. Apesar de se inspirar em outras linguagens, Ruby é facilmente entendível por programadores Java e C. A seguir, o detalhamento de algumas características de Ruby.

- **Multiplataforma**

Ruby é independente de plataforma, ou seja, seu interpretador possui implementação para diversos sistemas operacionais, entre eles Linux, Windows, Unix e Mac OS X.

- **Tipagem dinâmica**

Ruby utiliza tipagem dinâmica, não é preciso declarar tipos de variáveis e métodos, a verificação de tipo ocorre em tempo de execução, exemplo, quando houver a chamada de um método que não existe na classe, é retornado um erro dizendo que o método é *“undefined”*.

- **Linguagem interpretada**

A linguagem Ruby é executada por um interpretador, e é possível também executar o código de forma interativa, a qual consiste na execução de cada linha enquanto o código é digitado em um console.

Linguagens interpretadas têm reputação de serem lentas, devido ao fato de terem de inspecionar e compilar o código em tempo de execução. Mas interpretadores podem ser otimizados. O interpretador de Ruby tem evoluído a cada versão, tendo inclusive alcançado desempenho superior, em algumas situações, ao jRuby, implementação de Ruby que roda sobre a máquina virtual do Java.

- **Programação funcional**

A linguagem Ruby é multi-paradigma, além de ser puramente orientada a objetos (não há tipos primitivos). Embora não seja uma linguagem puramente funcional, suporta o paradigma, funções e blocos anônimos de código podem ser manipulados como qualquer outro objeto.

Nas linguagens orientada a objetos mais tradicionais, como Java, geralmente para iterar sobre um elemento, você precisa codificar duas lógicas: a da iteração e a lógica do bloco de código que será executado repetidas vezes. Mas a iteração é apenas um detalhe que pode ser generalizado, o problema que se quer resolver é aplicar a mesma lógica para cada elemento de uma coleção. Em Ruby, você pode passar um bloco de código para um método que implementa a lógica de iteração nos bastidores.

Por exemplo:

*[1, 2, 3, 4].each {| n | print n * 2 } # Essa linha de código imprime “2468”*

- **Metaprogramação**

Metaprogramação nada mais é que código que escreve código, um programa que altera a si mesmo em tempo de execução ou que gera parte de seu próprio código em tempo de compilação. Java, por exemplo, permite metaprogramação até certo nível, através das *annotations,* recurso adicionado na versão 5, mas não é algo trivial. Em linguagens dinâmicas, como Ruby, isso se dá de forma mais facilitada. Em Ruby é possível inserir novos métodos em classes em tempo de execução e criar apelidos para métodos existentes e montar métodos dinamicamente conforme o banco de dados, como o framework Rails faz, por exemplo: Uma classe chamada *User* está relacionada com a tabela *users,* a qual tem um campo chamado *name.* Neste caso o Rails criará automaticamente, ao carregar a classe *User* no servidor, um método estático chamado *find_by_name,* o qual filtra os usuários pelo nome e retorna um array de objetos da classe *User*.

- **Ruby Gems**

A comunidade Ruby utiliza um sistema unificado e eficiente de distribuição de *Gems* (bibliotecas em Ruby)*,* basta especificar a biblioteca no arquivo *Gemfile* e executar o comando *bundle install,*que a *Gem* é instalada na versão estável mais nova, também pode ser especificado a versão da *Gem.*

Ruby foi escolhida para este projeto pela codificação ser ágil e elegante, pela familiaridade do autor deste trabalho e por ser a linguagem de implementação do framework Ruby on Rails. Um dos objetivos do trabalho é possibilitar que outras pessoas possam programar novos diagramas facilmente. A experiência prática do autor

deste trabalho comprovou que Ruby possibilita o desenvolvimento ágil com poucas linhas de código, possibilitando construir protótipos e sistemas em poucos meses.

5.3 Ruby on Rails

Ruby on Rails é um framework MVC de código aberto e otimizado para a produtividade no desenvolvimento de sites orientados a banco de dados (*database-driven Web sites*). A Figura 7 mostra o funcionamento de uma aplicação Rails interagindo com o cliente. *Models* representam os dados. *Controllers* são carregados pelo *Router,* o qual é responsável pelas rotas que o cliente fará as requisições, ou seja, as *urls*. Os *Controllers* carregam os *Models*, e podem aplicar os dados nas *View* e entregar no formato HTML para o cliente ou outros formatos como o JSON, que é o caso da aplicação deste trabalho.

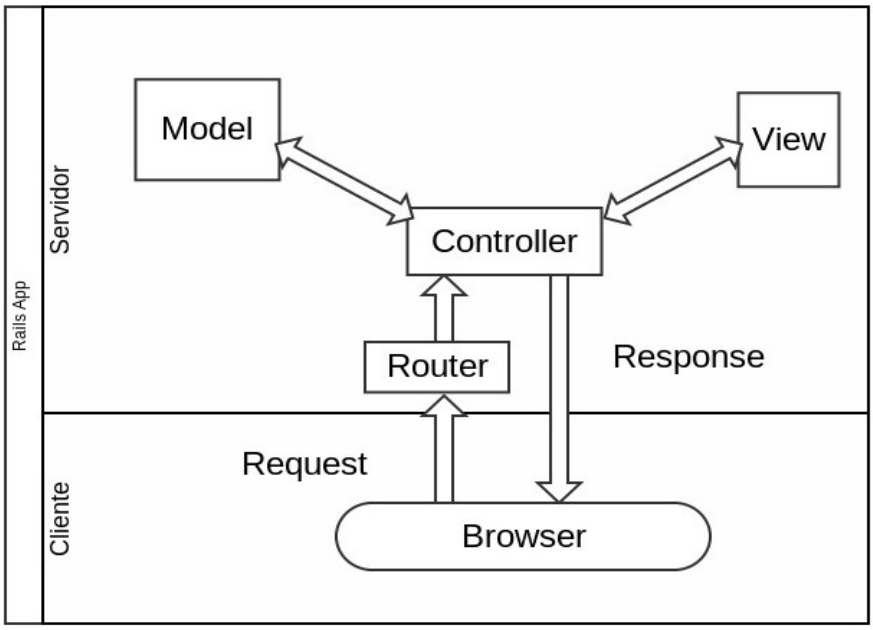

Figura 7: Representação básica da estrutura MVC do Rails com o cliente

Rails permite a codificação de forma elegante e ágil e conta com recursos que favorecem convenção ao invés de configuração (RAILS.ORG, 2013). Programação por convenção(também conhecido por convenção sobre configuração) é um paradigma que visa diminuir a quantidade de configurações e o número de decisões dos desenvolvedores a fim de ganhar simplicidade sem necessariamente perder a flexibilidade. Em programação por convenção não é preciso especificar como o *framework* se comportará, pois o mesmo assume algumas premissas e o programador irá configurar somente se houver necessidade de alterar essas premissas.

Um exemplo simples do uso da convenção em Rails seria desenvolver uma página de postagem de conteúdo em um blog, em Rails essa tarefa consiste em codificar um *model,* um *controller,* uma *view* e deixar o model disponível no arquivo de rotas, no nosso caso, o seguinte trecho de código: *resource :post.* Esse trecho irá gerar as rotas contidas na seguinte tabela 2:

Verbo HTTP	**Path**	**Controller#Action**	**Usado para**
GET	/post/new	posts#new	Retornar uma página HTML para criação de um novo post
POST	/post	posts#create	Criar um novo post
GET	/post	posts#show	Exibir um post
GET	/post/edit	posts#edit	Retornar uma página HTML para edição de um post
PATCH/PUT	/post	posts#update	Atualizar um post
DELETE	/post	posts#destroy	Apagar um post

Tabela 2: Rotas geradas pelo Rails

Considerando as rotas exibidas na tabela anterior, o framework espera que você crie um model *app/models/post.rb*, um controller *app/controllers/posts_controller.rb*

com os métodos *new, index, create, edit, update* e *destroy,* e as necessidades que saírem deste padrão, podem ser configuradas. Esse é apenas um de vários exemplos do uso de convenção no framework.

Ruby on Rails foi escolhido para este trabalho por ser focado em desenvolvimento Web, por proporcionar uma API REST que se comunica facilmente com o Backbone (cliente). Outro fator determinante para a escolha, é o fato do Redmine ser escrito com o Rails, o que possibilita uma fácil integração através de seu sistema de plugins, um dos objetivos básicos deste trabalho.

5.4 Redmine

Redmine é uma aplicação Web flexível para gerenciamento de projetos. Foi escrito em Ruby on Rails, é multi-plataforma, suporta diversos bancos de dados, integra com outros sistemas e tem suporte à criação de plugins o que vai de encontro com a proposta deste trabalho, pois a aplicação resultante poderá ser utilizada como um *plugin,* e se futuramente for implementada em outro sistema que utilize Rails, pode ser desacoplado e adaptado facilmente. Segundo Redmine (2013), o sistema conta com as seguintes características:

- Suporte a vários projetos;
- Controle de acesso baseado em papéis de usuário;
- Sistema flexível de acompanhamento de problemas;
- Gráfico de Gantt e calendário;
- Notícias, documentos e arquivos de gestão;
- Feeds e notificações por e-mail;
- Páginas de edição coletiva de documentos por projeto;
- Fóruns por projeto;

- Controle de tempo;
- Campos personalizados para problemas, entradas de tempo, projetos e usuários;
- Integração SCM (SVN, CVS, Git, Mercurial, Bazaar e Darcs);
- Suporte a vários idiomas;
- Apoio múltiplos bancos de dados.

5.3 Javascript

JavaScript é uma linguagem interpretada com tipagem fraca e dinâmica, com suporte a programação funcional e programação orientada a objetos que utiliza protótipos ao invés de classes para criar objetos. JavaScript é a linguagem de *script* da Web, geralmente utilizada para programar o comportamento das páginas e manipular os seus elementos. É utilizada pela maioria dos sites e vem ganhando importância atualmente motivada pela adoção do HTML5, pois muitas aplicações antes feitas em flash agora podem ser construídas em JavaScript seguindo a nova especificação do HTML, eliminando assim, a necessidade de plugins instalados no sistema ou navegador. JavaScript é uma das três principais tecnologias que os desenvolvedores Web devem saber: HTML para conteúdo, CSS para apresentação e JavaScript para o comportamento das páginas. Todos os navegadores modernos para desktop possuem um interpretador de JavaScript, também presente em consoles de games, smartphones e tablets, o que possibilita afirmar que JavaScript é uma das linguagens mais utilizadas.

Segundo Flanagan (2011), JavaScript há muito tempo superou suas raízes de linguagem de *script* para se tornar uma linguagem de propósito geral robusta e eficiente . Na versão mais recente foram adicionados novos recursos para suportar o desenvolvimento de software de grande escala.

A seguir a tabela de comparação de JavaScript com Java.

	JavaScript	Java
Execução	Interpretada	Compilada para *bytecode* e interpretada pela VM
Tipagem	Dinâmica e fraca	Estática e forte
Paradigma	Multi-paradigma	Orientação a objetos
Surgido em:	1995	1995

Tabela 3: Comparação de JavaScript com Java

Devido à interface do sistema desse trabalho ser baseada nos padrões do HTML5 para navegadores modernos, foi feito o uso de JavaScript para a interação com o usuário e comportamento da aplicação. Como a parte mais ampla no desenvolvimento foi feita com JavaScript, diversas bibliotecas foram necessárias para facilitar o processo de desenvolvimento. A seguir são detalhadas as bibliotecas JavaScript utilizadas neste projeto.

5.3.1 Backbone.js

“Backbone.js é uma biblioteca livre que provê estrutura para aplicações Web, fornecendo modelos com ligação de chave-valor, eventos personalizados, coleções com uma API de funções enumeráveis, *views* com o tratamento de eventos declarativo, e conecta tudo à sua API através de uma interface RESTful JSON” (BACKBONEJS.ORG, 2013).

Backbone.js possibilita obter representações dos dados do servidor através de seus *Models*, fornecendo busca, validação, edição, exclusão, e gravação no servidor. Uma *View* é o elemento que manipula o *Model* e aplica os dados do mesmo nos

templates e apresenta no formato HTML para o usuário. A *View* define funções de *callback* para os eventos do Model, os quais podem estar relacionados com mudanças nos atributos, assim a *View* mantém o estado da representação dos *models* sempre atualizado para o usuário.

Backbone.js foi escolhida para este trabalho, pelo fato, de separar as chamadas do servidor da interface da aplicação *(views)*. A biblioteca é recomendada para aplicações em JavaScript que sejam extensas e façam bastantes chamadas Ajax ao servidor, o que é o caso desta aplicação, visto que o usuário salvará recorrentes vezes e sem redirecionamento como num formulário de uma aplicação Web simples. Backbone.js também ajuda o desenvolvedor organizar melhor o código, pois cria uma estrutura de código semelhante ao MVC, mas com algumas particularidades como a ausência de *controllers.* A função de controlador, é dividida entre as *Views* e as *Routes*.

As principais virtudes da biblioteca são sua leveza e a capacidade de propiciar organização e estrutura para aplicações de grande porte em JavaScript.

5.3.2 JointJS

JointJS é uma biblioteca Javascript de código aberto para a criação de diagramas renderizados em SVG. Pode ser utilizada para criar diagramas estáticos ou iterativos, a biblioteca permite a criação de diagramas através de links e elementos relacionados. JointJS foi construída utilizando Backbone.js como base, assim a mesma é uma dependência.

É possível utilizar JointJS para criar qualquer diagrama estático ou editores de diagramas interativos. JointJS facilita a criação de aplicações visuais de diversos tipos, a natureza orientada a eventos somada com a arquitetura MVC de JointJS facilita a ligação com qualquer ambiente de servidor.

Segundo Client.IO (2013), a biblioteca conta com as seguintes características:

- Elementos básicos de diagramas (retângulo, círculo, elipse, texto, imagem, *path*);
- Elementos de diagramas personalizados com base em SVG;
- Elementos e links interativos;
- Conexão entre elementos através de *link;*
- Links com setas e rótulos personalizáveis;
- Ligações de suavização (interpolação bezier);
- Ímãs (pontos de conexão para links);
- Diagramas hierárquicos;
- Serialização e desserialização de e para o formato JSON;
- Altamente orientado a eventos;
- Gerenciamento de zoom;
- Suporte à toques (tablets);
- Animações;
- Suporte à NodeJS;
- Capacidade de renderizar centenas de elementos e links com interação instantânea;
- Arquitetura MVC;

A biblioteca Joint.js tem como base o Backbone.js e tem um ótimo desempenho na manipulação de gráficos vetoriais, esses são os fatores mais importantes para sua escolha neste trabalho.

5.3.3 JQuery

Jquery é uma biblioteca JavaScript de propósito geral, pequena e rica em recursos que auxilia no desenvolvimento simples e *cross-browser.* Jquery facilita

tarefas como transferência e manipulação de documentos HTML, manipulação de eventos, animação e Ajax. Jquery tem uma API fácil de utilizar e que funciona na maioria dos navegadores (JQUERY, 2014). A biblioteca também oferece suporte para criação de plugins sobre ela.

5.3.4 Underscore

É uma biblioteca JavaScript que é dependência do Backbone e também fornece diversas funcionalidades de propósitos gerais que não existem no JavaScript nativo como *map, reduce, select, filter, foreach, etc*.

Outro ponto positivo da biblioteca é que os scripts procuram sempre utilizar recursos nativos do navegador, os quais são implementações otimizadas.

5.4 Conclusão

Todas as tecnologias utilizadas neste trabalho cumpriram o seu papel. O que é resultado de boa parte do tempo para realização deste trabalho ter sido empregado na pesquisa de tecnologias e bibliotecas.

6 Arquitetura do Sistema

Para o desenvolvimento deste trabalho procurou-se seguir um projeto planejado com boas práticas de programação e sempre dando ênfase no reuso de software e a utilização de bibliotecas e frameworks livres. Técnicas de convenção de nomes de arquivos e pastas também são utilizadas para agilizar o desenvolvimento de novos diagramas e evitar configurações repetitivas.

Como se trata de uma aplicação Web, o sistema é composto por duas partes, a parte que roda no servidor, a qual controla o acesso e interage com o banco de dados e outros serviços. A outra parte é a que roda no cliente, a qual contém a interface do sistema com os editores de diagramas e é responsável pela interação com o usuário.

A figura 8 exibe um *Deployment Diagram*, que representa a arquitetura da aplicação, a qual é composta por três nodos de hardware, o cliente, o servidor e o serviço GIT, abordados a seguir:

- **Server**

O nodo Server roda Linux com uma aplicação Redmine, a qual é feita em Ruby on Rails. A arquitetura Rails (MVC) do Redmine tem uma peculiaridade, que é a pasta *plugins*, a qual fica na raíz do projeto e contém os plugins instalados e que também são MVC. Então temos uma estrutura de arquivos MVC dentro de outra, esta estrutura de pastas pode ser vista na figura 9, a estrutura MVC da aplicação e do plugin, ficam nas suas respectivas pastas *app*.

Como podemos ver na figura 8, aplicação principal e plugin tem seus próprios arquivos *Gemfile*, os quais especificam as *Gems* (bibliotecas), que serão utilizadas. Para instalar o plugin do BR-Dia no Redmine, basta colocar a pasta do plugin na pasta plugins do Redmine e rodar o comando para gerar tabelas no banco de dados relativas ao plugin.

- **Client**

O nodo Client, é executado no browser e controlado pelo Backbone.js. Este nodo tem basicamente arquivos de *models, views* e *templates*. Seu código reside na pasta *assets do* plugin e é enviado para ser executado no cliente. Um detalhe importante é que essa é uma aplicação voltada para a parte do cliente. Então ao invés de compilar *templates* das *views* no servidor e enviar o HTML para o cliente, o Rails converte seus objetos em objetos JSON (menos pesado que HTML) e envia para o Backbone tratar e aplicar o JSON no HTML. O arquivo HTML com o código JavaScript é carregado somente uma vez, ao entrar na página e iniciar a aplicação. Assim mudanças nos dados não implica em carregar novos documentos HTML.

- **GIT Server**

O nodo GIT Server, representa um servidor de controle de versão GIT, o qual é acessado pela aplicação utilizando a biblioteca ruby-git, a qual é chamada pelo *controller* e implementa uma conexão HTTPS com o servidor GIT, utilizando chaves SSH armazenadas no servidor da aplicação.

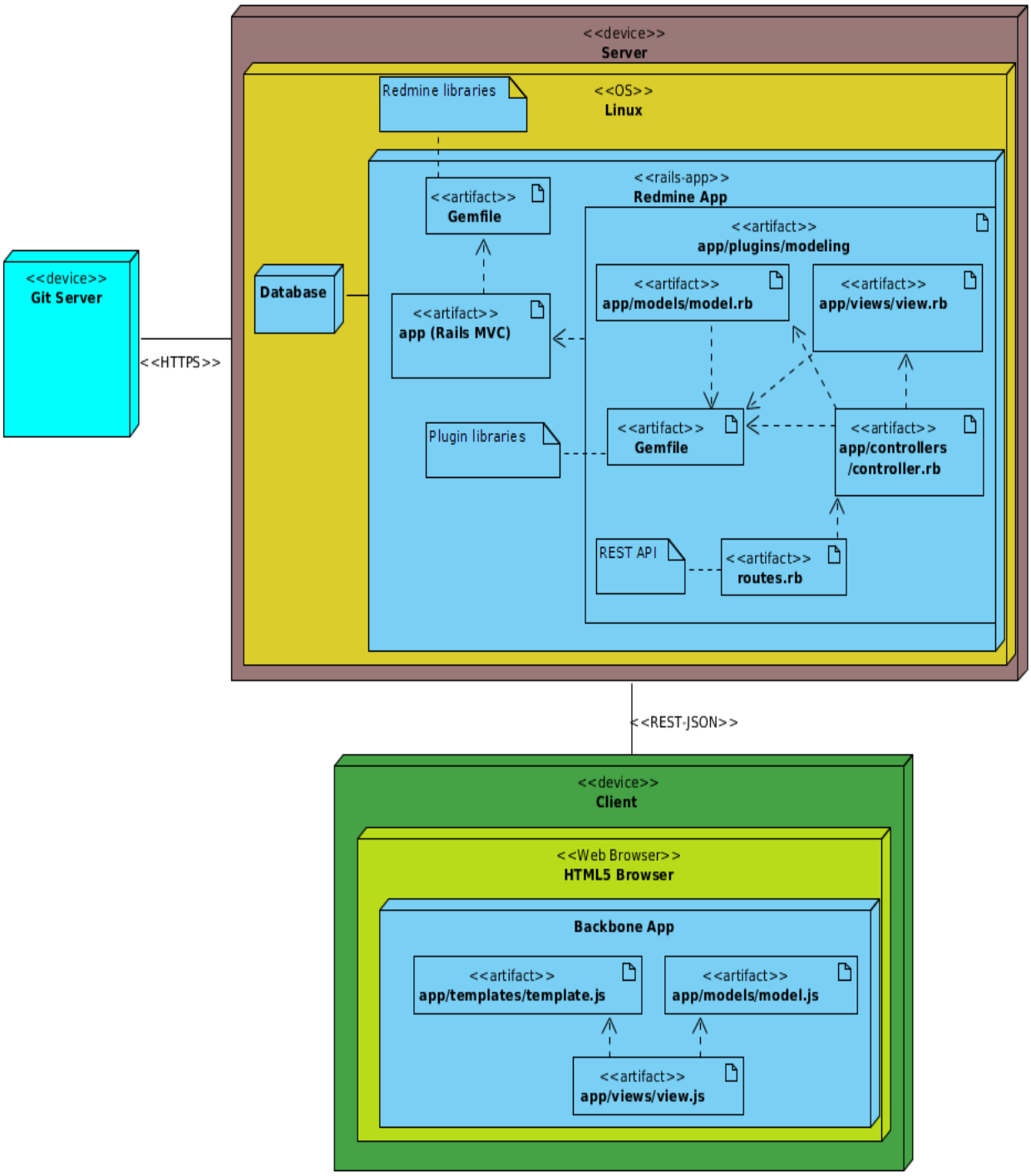

Figura 8: Deployment Diagram da Arquitetura Cliente Servidor da aplicação

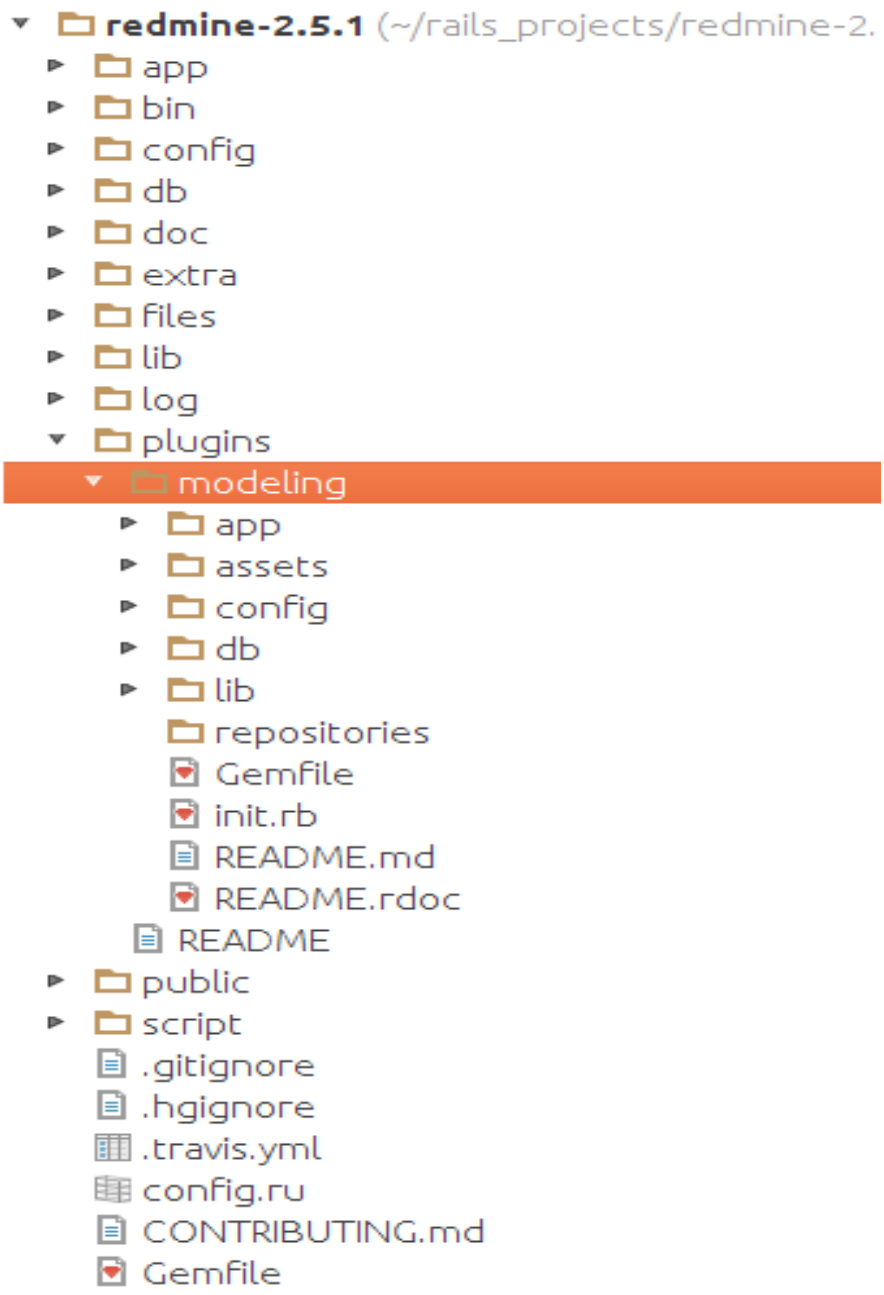

Figura 9: Estrutura de pastas da aplicação e o plugin

6.1 Fluxo de dados da aplicação

Para demonstrar como funciona o fluxo de dados na arquitetura e sua relação com a apresentação da aplicação, as figuras 10 e 11 Fall (2012), mostram o fluxo de dados de um sistema feito com Backbone.js, que é o caso da aplicação proposta.

A figura 10 exibe a ação de atualizar ou salvar um dado no servidor, o usuário acessa a *view* e executa alguma ação que altera o *model*, daí para salvar no servidor tem duas opções: pode ser programado para salvar a cada alteração do *model* ou através de um botão salvar, opção mais econômica em termos de consumo de recursos.

A figura 11 exibe a ação de buscar um dado no servidor. O usuário acessa as *views* que manipulam os *models* especificados esses modelos por sua vez notificam o servidor, que busca no banco de dados e retorna no formato JSON para os *models*. E por fim os models notificam as *view,* que alteram o seu conteúdo.

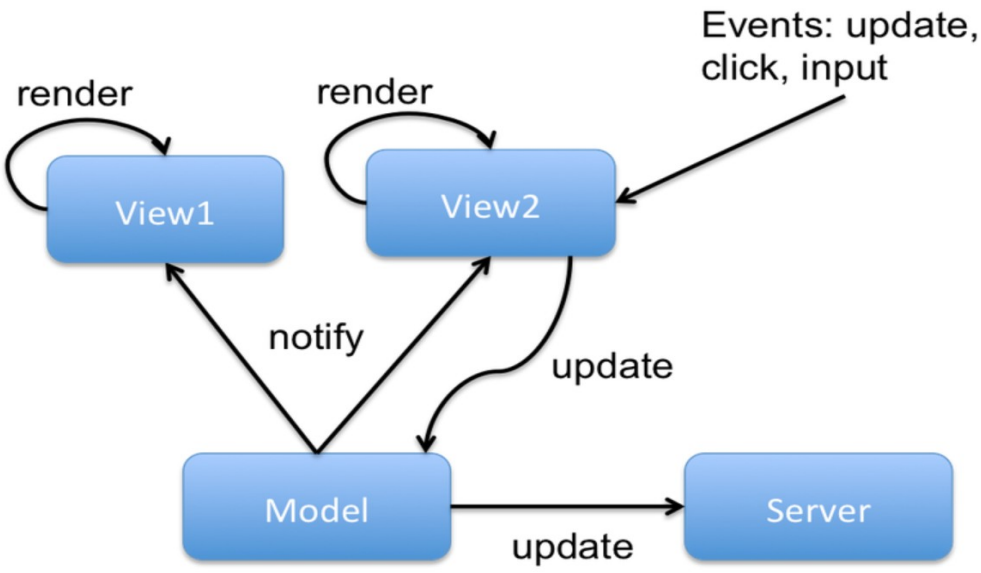

Figura 10: Ação de atualizar um dado no servidor (FALL, 2012)

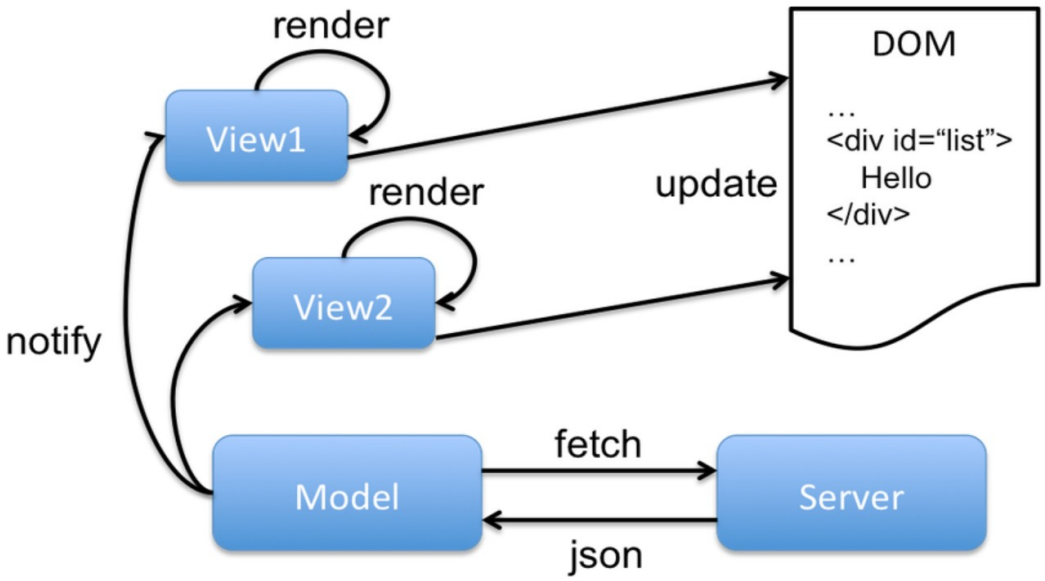

Figura 11: Ação de buscar um dado no servidor (FALL, 2012)

6.2 Servidor

A parte do servidor é responsável pelo armazenamento dos modelos no banco de dados, pela compilação dos templates *.html.jst* e pela interface com o lado do cliente. Também é responsável por interagir com serviços externos, no caso deste trabalho com repositórios GIT.

O servidor roda com uma aplicação do Redmine, como o objetivo inicial da aplicação é ser um plugin, a estrutura MVC da aplicação pode tanto ser portada para o Redmine, colocando a pasta do plugin do BR-Dia dentro da pasta plugins do Redmine, como pode ser usada sem o Redmine, simplesmente acrescentando um sistema de login e fazendo configurações recorrentes de uma típica aplicação Web com Rails. A seguir serão mostrados os dois módulos que compõe a ferramenta na parte do servidor.

6.2.1 Módulos da ferramenta no servidor

Para apoiar o desenvolvimento do *back-end*, foi criado 2 módulos de classes, BrDiaModule e DocumentModule, os quais são integrados com o framework Rails e ficam na pasta *lib* do plugin. A seguir a definição desses módulos:

- **BrDiaModule**

Este módulo é responsável pelas atividades de apoio da ferramenta e por interagir com ferramentas ou serviços externos como o GIT. Também é responsável pelo compilador de templates *.jst.html*, que são enviadas para o cliente e pelo gerador de código-fonte, o qual pode ser utilizado para gerar qualquer tipo de documento textual através de criação, inspeção e manipulação de arquivos.

- **DocumentModule**

Como o trabalho se propõe a desenvolver uma ferramenta com um ambiente de desenvolvimento de software com notação de alto nível, a base para a modelagem dos modelos, é o metamodelo do *framework* OCEAN, o qual foi produzido por Ricardo Pereira e Silva (SILVA, 2000) em sua tese de doutorado. Este metamodelo é adequado, pois foi concebido após diversos estudos em ferramentas de modelagem, os quais resultaram em um metamodelo bastante flexível, ou seja, ideal para esta ferramenta que se propõe a criar uma estrutura que facilite a adição de novos diagramas. Este metamodelo ajudou a generalizar os *models* e também os *controllers,* o que facilita na hora de implementar novos diagramas, pois não precisa partir do zero. Então o módulo *Document* é baseado no framework OCEAN.

6.2.2 Persistência dos dados (*models*)

Como o framework OCEAN já é testado e estudado, objetivando robustez no projeto, a base para os modelos e regra de negócio é baseada em seu metamodelo. Como o framework foi feito em um contexto diferente, com a linguagem Smalltalk e era uma aplicação desktop, o seu modelo foi adaptado neste trabalho para trabalhar com os recursos do framework Rails.

No OCEAN as associações de sustentação e referência entre os elementos de especificação são registradas através de tabelas de sustentação e referência. Os conceitos de sustentação e referência podem ser feitos em um banco de dados relacional através do relacionamento entre as tuplas sem precisar de tabelas extras para isso, é necessário somente as tabelas para as entidades. O Rails tem uma gama enorme de recursos para persistência e cria, inclusive, as tabelas e relacionamentos automaticamente no BD sem redundância através do mapeamento das classes. Os geradores de estrutura e classe do Rails podem ser usados inclusive em um futuro editor de especificação, personalizando-os.

A figura 12 apresenta o diagrama de classes para o core do módulo Document, o qual foi baseado no OCEAN e tem por finalidade ser flexível e prover uma estrutura para a fácil adição de novos diagramas e especificações.

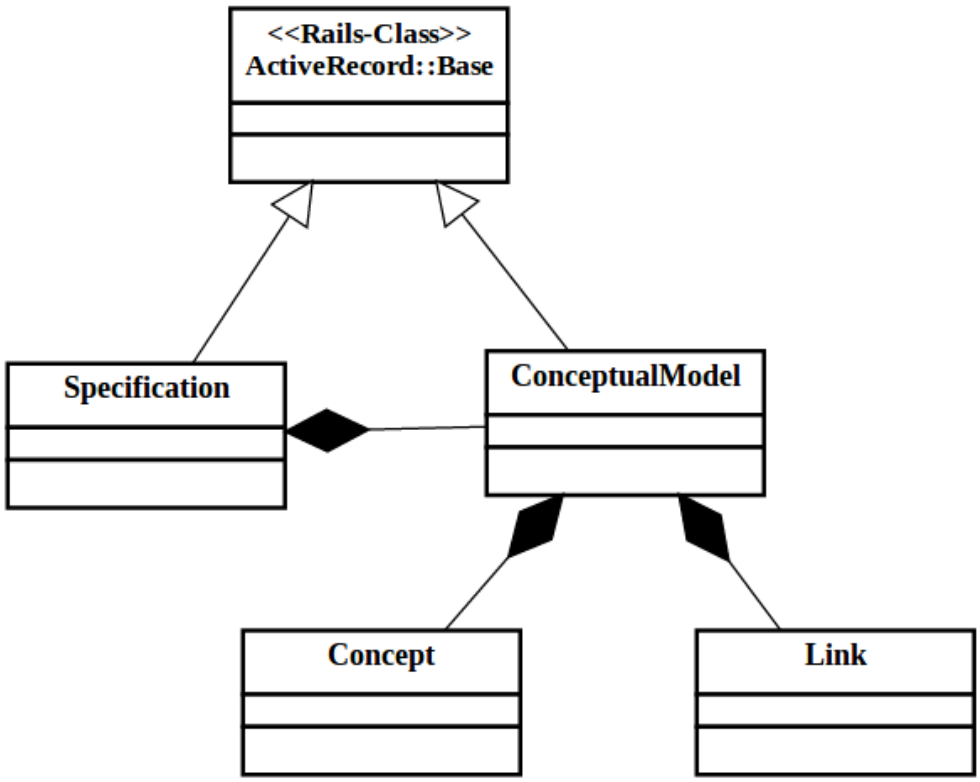

Figura 12: Meta-modelo adaptado

Em Ruby não existe classe abstrata, não há a necessidade de contratos, pois a tipagem é dinâmica. Isso pode ser emulado, mas não é considerado uma boa prática. Neste contexto, as classes do módulo Document (adaptado a partir do metamodelo) servirão de base para as outras classes, que não serão obrigadas a definir métodos da superclasse.

ActiveRecord::Base é a classe do Rails responsável por facilitar o acesso ao banco de dados. Specification e ConceptualModel estendem ActiveRecord::Base e representam uma tabela no banco de dados. As especificações e diagramas que estenderão essas duas classes, serão diferenciados no banco de dados através de seus tipos, ex: *OOSpecification* e *ClassDiagram.*

O diagrama de classes da figura 13 exemplifica o relacionamento do *core* da ferramenta com a implementação da estrutura do metamodelo do diagrama de classes e a relação da parte de *models* do framework Rails, o *ActiveRecord,* com as outras classes dos modelos. É importante notar que as classes *OOSpecification* e *ClassDiagram* não representarão as tabelas no banco de dados e sim *Specification* e *ConceptualModel* que são os agregadores.

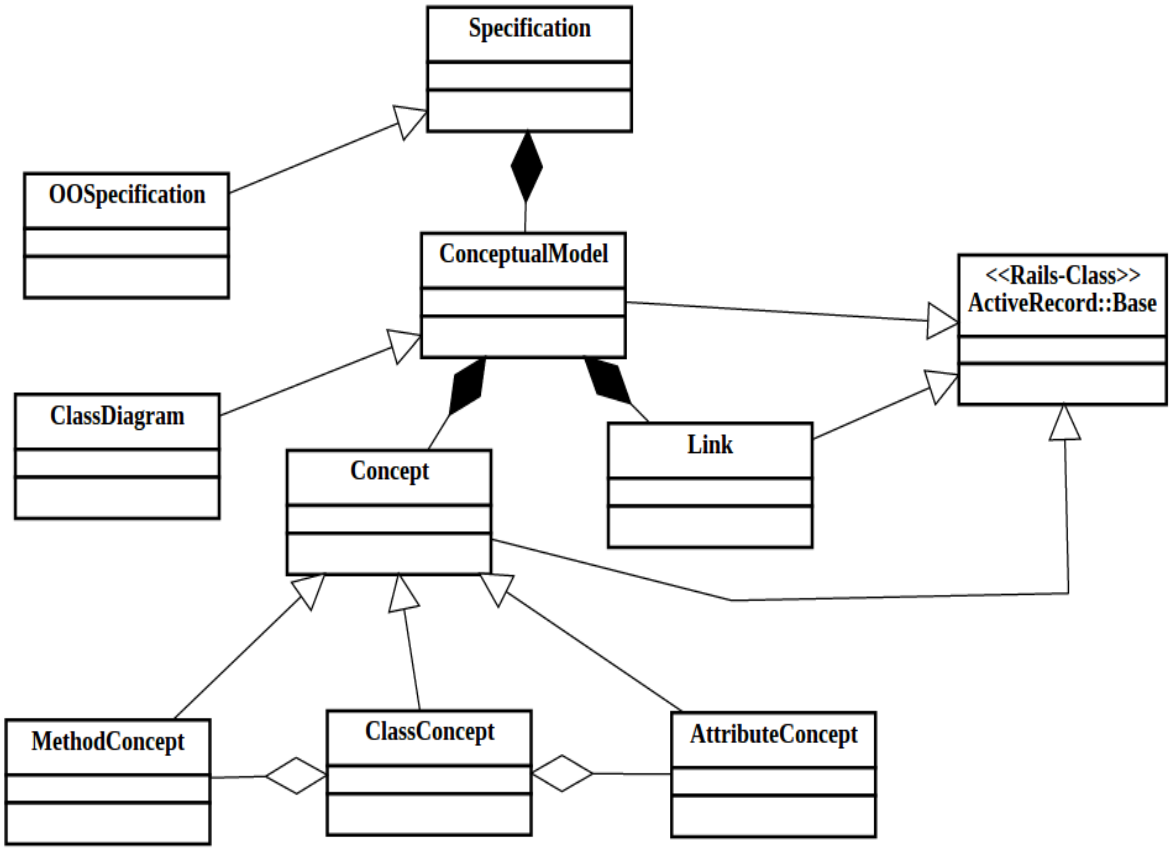

Figura 13: Relação do core da ferramenta com o diagrama de classes

6.2.3 Controllers

Os conceitos do metamodelo do framework OCEAN ajudaram também a generalizar os *controllers* (figura 14), possibilitando criar uma estrutura que possibilita reutilizar os *controllers* existentes. Assim ao criar novos *models* para novos diagramas, não é preciso criar novos *controllers* referentes a esses *models*. O que mantém a estrutura dos *controllers* bastante simples, facilitando a comunicação com o cliente que é feita com a API de REST do Rails.

Na figura 14, o método *index()* de *SpecificationsController*, é o único que retornará um elemento HTML, pois será carregado somente uma vez ao iniciar a aplicação, nele estarão todos os *templates* e arquivos JavaScript necessários para executar a aplicação. Assim, durante a execução da aplicação serão transferidos somente dados e não HTML.

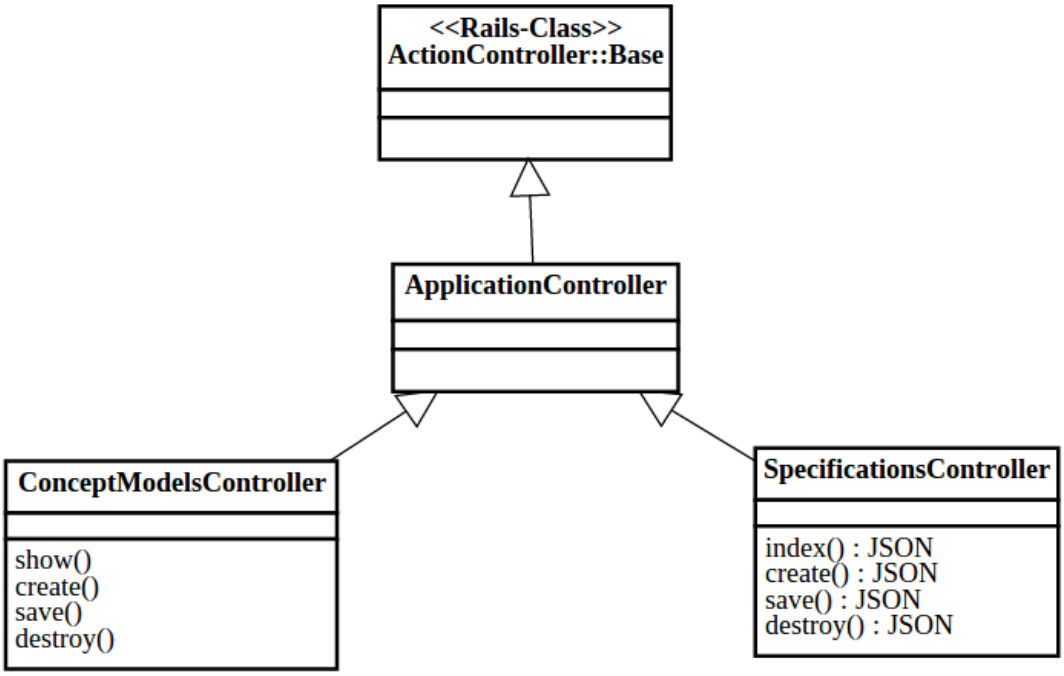

Figura 14: Controllers da aplicação no servidor.

6.3 Cliente

No lado do cliente, a interação com o usuário e comportamento da página é codificada com JavaScript e a apresentação é feita com HTML e SVG. A figura 15, mostra como funciona o sistema na parte do cliente, utilizando a biblioteca JointJS. Em JointJS, diagramas consistem em um conjunto de elementos conectados por links e são representados pelo modelo (*model*) joint.dia.Graph. Este modelo agrupa células, as quais podem ser um elemento ou um link. Em JointJS você manipula modelos e não elementos da interface (views), as views reagem, através da captura de eventos das modificações no modelo. Assim se garantirá que o modelo sempre refletirá o estado atual do dado.

A parte referente ao cliente na aplicação deste trabalho foi feita estendendo os protótipos de JointJS. Exemplo: um diagrama de classes é composto pelo modelo Class que estende Element, por associações que estendem o elemento Link e por ClassView, o qual estende ElementView. Nem todos os casos é necessário estender LinkView, grande parte pode ser suprida com o protótipo original da própria biblioteca.

A Figura 16 mostra um diagrama de classes simplificado que reflete o relacionamento das views relacionadas com o diagrama de classe. No sistema, um *Paper* é o manipulador da área de edição e visualização do diagrama, manipula o elemento marcado com a *tag <svg>.*

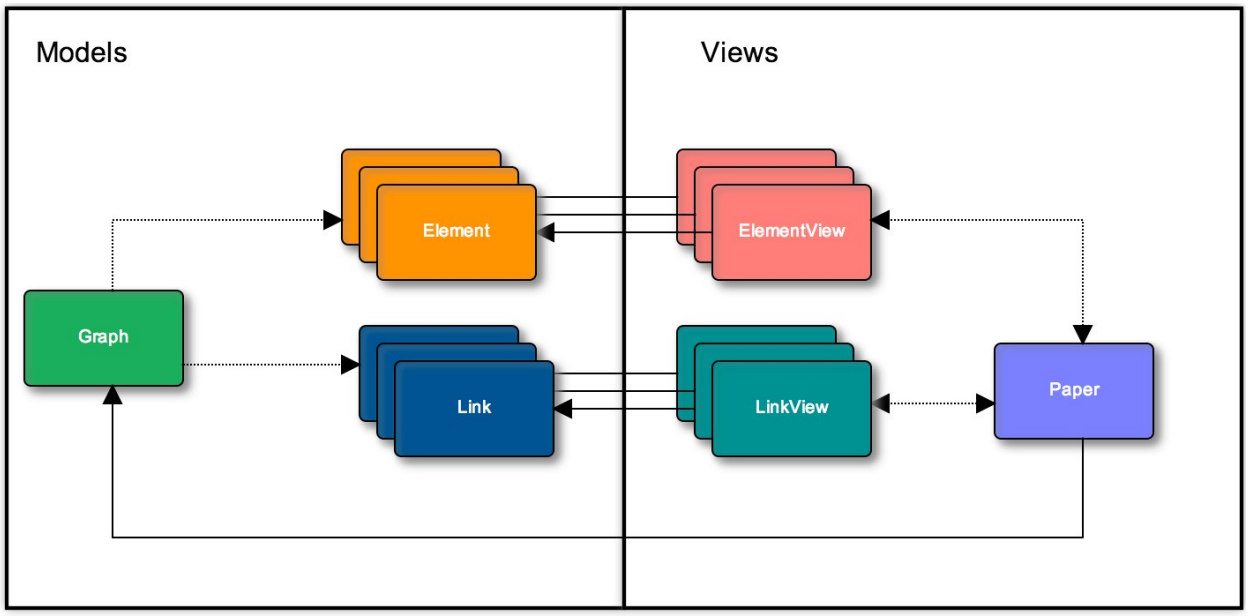

Figura 15: Funcionamento do sistema na parte do cliente

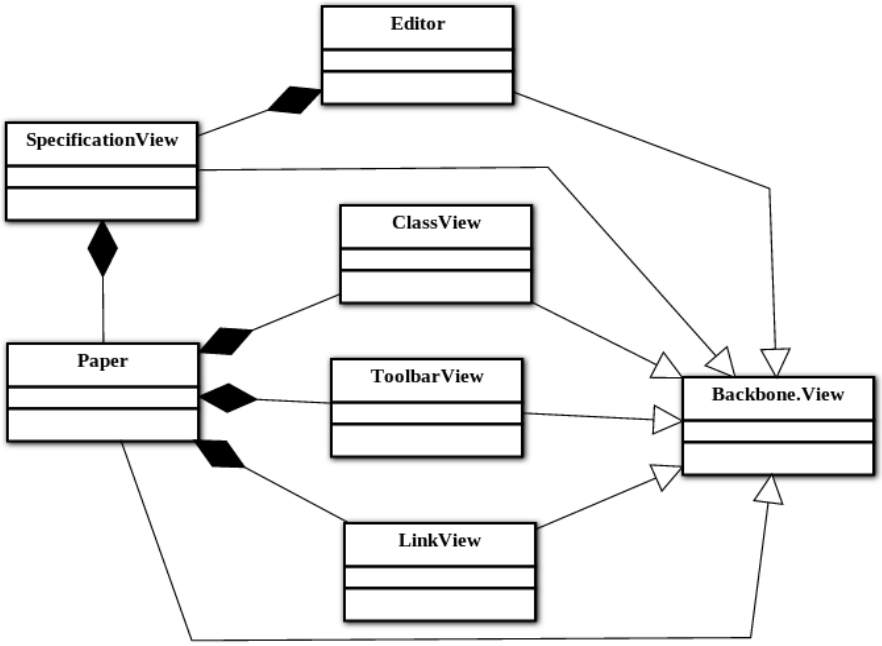

Figura 16: Diagrama de classes simplificado do relacionamento das views no cliente

6.5 Consistência na representação dos modelos.

Este trabalho utiliza UML para modelar os diagramas, e para a modelagem ser produtiva e proveitosa em termos de geração de código-fonte os diagramas devem ter consistência em suas representações. Por exemplo, um método do diagrama de sequência deve representar um método na diagrama de classe, e se o método for

apagado do diagrama de classe, o mesmo deve ser removido no diagrama de sequência.

A tarefa de garantir a consistência será do lado cliente. A classe dia.SpecificationView, registra os eventos dos elementos modificados ou removidos e notifica os diagramas, que por sua vez fazem as modificações necessárias. Para que os elementos disparem a notificação para *Specification* registrar, eles devem ter alguma ligação com elementos externos, a figura 30 do apêndice A exibe, no diagrama de máquina de estados de dia.Element, o momento em que o elemento passa a notificar Specification com o método *trigger().*

6.4 Integração com outras ferramentas

A integração da ferramenta com o Redmine é feita via plugin, o Redmine armazena seus plugins com uma estrutura MVC interna para cada plugin. Por exemplo: para criar a estrutura de um plugin, é só rodar no terminal: *rails generate redimine_plugin nome_do_plugin,* este comando cria todas as pastas, arquivos e configurações iniciais necessárias. Tudo ficará na pasta *plugin.* Para instalar o plugin da ferramenta BR-Dia, é só colocar a pasta do plugin na pasta *plugins* do Redmine. Quando o servidor do Redmine iniciar, será executado o arquivo de configuração do plugin, o qual faz as alterações necessárias para integrar o plugin.

A integração da ferramenta com o GIT, é feita com uma *gem* chamada ruby-git, a qual estabelece uma conexão HTTPS com um repositório GIT, mas para isso ocorrer, deve ser gerado uma chave pública SSH e uma chave privada, e enviado a chave pública para a conta no repositório, que pode ser o *Github.com*, por exemplo.

7 A Ferramenta BR-Dia

BR-Dia é uma ferramenta CASE para modelagem UML dos diagramas de classe, máquina de estados, casos de uso, sequência e atividades. Permite a geração de código-fonte Java e envio para repositórios GIT e é integrada com o Redmine tendo as suas entidades ligadas à entidade “Projeto”, do Redmine. A ferramenta é livre, e tem uma arquitetura voltada para facilitar a adição de novos tipos de diagramas.

7.1 Principais funcionalidades

A seguir os principais recursos conseguidos com a ferramenta BR-Dia, produzida neste trabalho.

- Diagramas interativos
- Edição com menu rápido ao passar o mouse sobre o elemento
- Função para avançar e voltar no histórico de alterações
- Links com formato editável ou automático
- Edição de qualquer elemento do diagrama
- Zoom até o limite máximo que o navegador suporta sem perder a nitidez
- Suporte à toque
- Integração com o Redmine
- Integração com o Github
- Compatível com as últimas versões dos navegadores modernos, incluindo os navegadores móveis.

7.2 Integração

A integração com o Redmine foi feita através de seu sistema de plugins. Como o Redmine utiliza o framework Rails, um plugin é uma espécie de mini aplicação rails dentro da pasta plugins que fica na raíz do projeto do Redmine.

A integração com o Github é feita com a biblioteca ruby-git, a qual provê funcionalidades de leitura e escrita em repositórios GIT. O usuário especifica um endereço de repositório e o nome da sub pasta e ao apertar o botão de commit, é gerado o esqueleto das classes do diagrama de classes e é enviado a pasta com o código para o repositório GIT, iniciando assim execução do projeto do código. A figura 17 mostra a tela de criação de especificação, onde pode ser configurado o repositório. A figura 18 mostra a tela de criação de diagramas, onde é configurado o código que vai para o repositório.

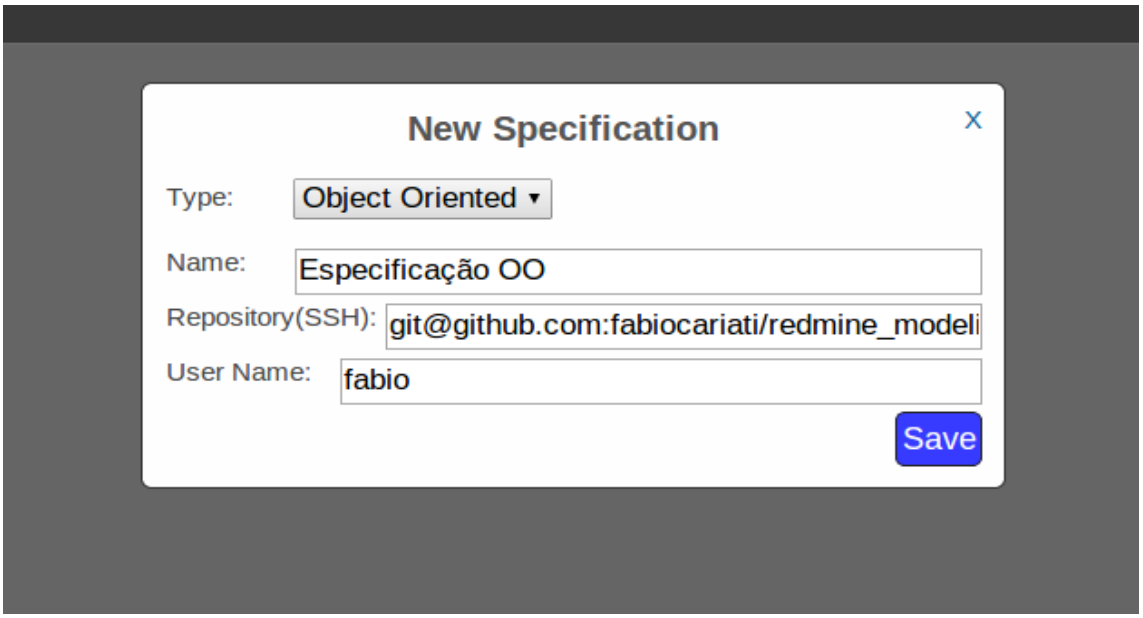

Figura 17: Criação de especificação

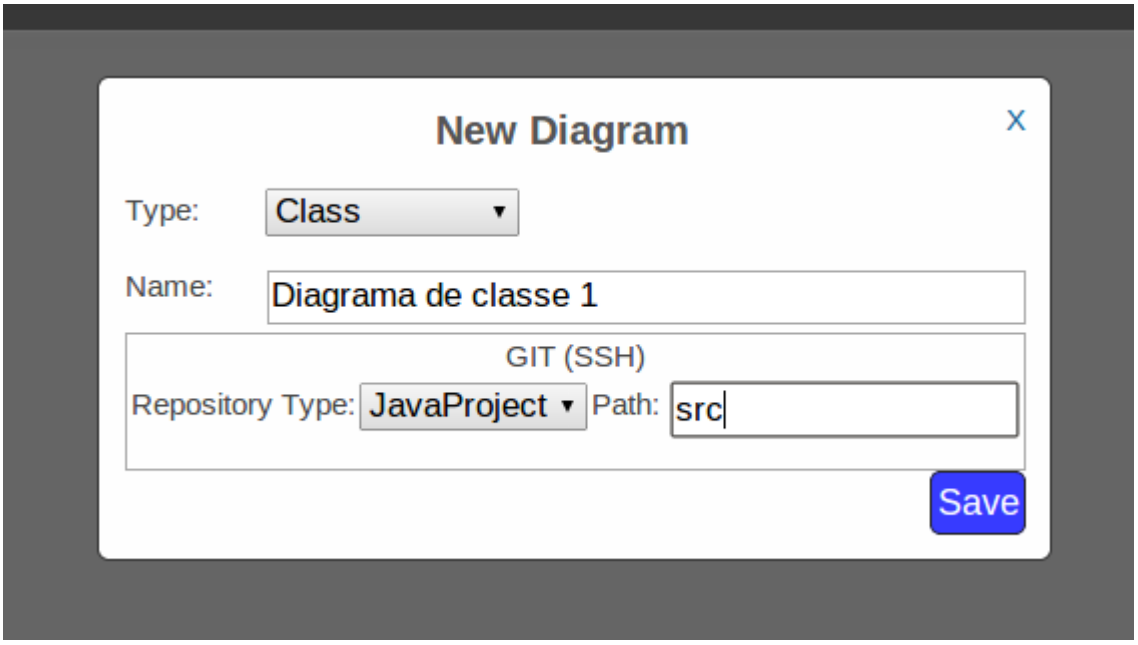

Figura 18: Criação de diagrama de classes

7.3 Portabilidade e compatibilidade

O sistema utilizado para testar a aplicação localmente foi o Ubuntu 13.10 e o serviço para publicar a aplicação e testar a mesma rodando na internet foi o serviço Heroku na modalidade gratuita, o qual roda em Linux com banco de dados PostgreSQL e tem um interpretador Ruby. O software funciona em todos os browsers modernos e atualizados, embora tenha desempenho superior no Chrome, o que se deve ao seu interpretador de javascript ter desempenho superior na renderização de SVG, o que pode ser superado aumentando o valor do *grid,* que é o parâmetro que indica o salto que o elemento dá ao se movimentar. Mas o Firefox tem melhorado o seu desempenho a cada nova versão, o que tende acontecer em os outros browsers também, visto que sites famosos, como o da Apple, passaram a adotar imagens SVG, que por serem

vetoriais tem nitidez em qualquer tamanho de tela, fator altamente relevante, considerando o contexto de computadores e dispositivos móveis de todo o tipo de tamanho e padrões de telas.

7.4 Uso

Para utilizar a ferramenta é necessário criar um projeto dentro do Redmine. Com o projeto criado, no menu principal terá uma aba chamada *Modeling,* a qual é responsável pela edição de diagramas do projeto e tem um menu superior, exibido na figura 19, com opções para carregar e editar especificações.

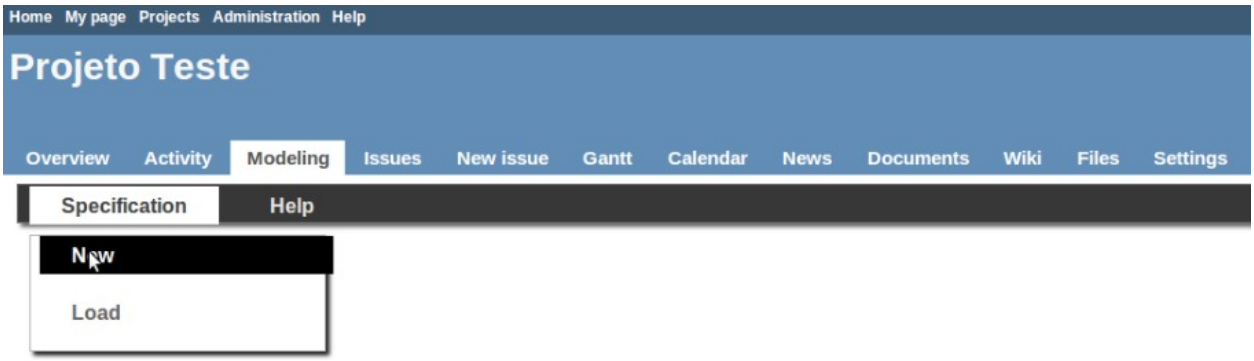

Figura 19: Menu superior da ferramenta

As seções a seguir irão detalhar o uso do editor de diagramas e especificações e as funcionalidades da ferramenta resultante deste trabalho.

7.4.1 Criação e edição de especificação

Uma especificação agrega diagramas, para criar a mesma, é necessário ter um projeto criado no Redmine, depois disso, é necessário acessar o menu principal do projeto na aba “modeling” e acessar no menu principal da ferramenta o item

Specification e sub-item *New* (figura 19), o que leva até a tela que já foi da mostrada na figura 17.

7.4.2 Criação e edição de diagramas

Para criar um diagrama, utiliza-se o menu lateral correspondente à estrutura de arquivos da especificação. A barra de ferramentas (figura 20) para cada diagrama fica no topo do mesmo.

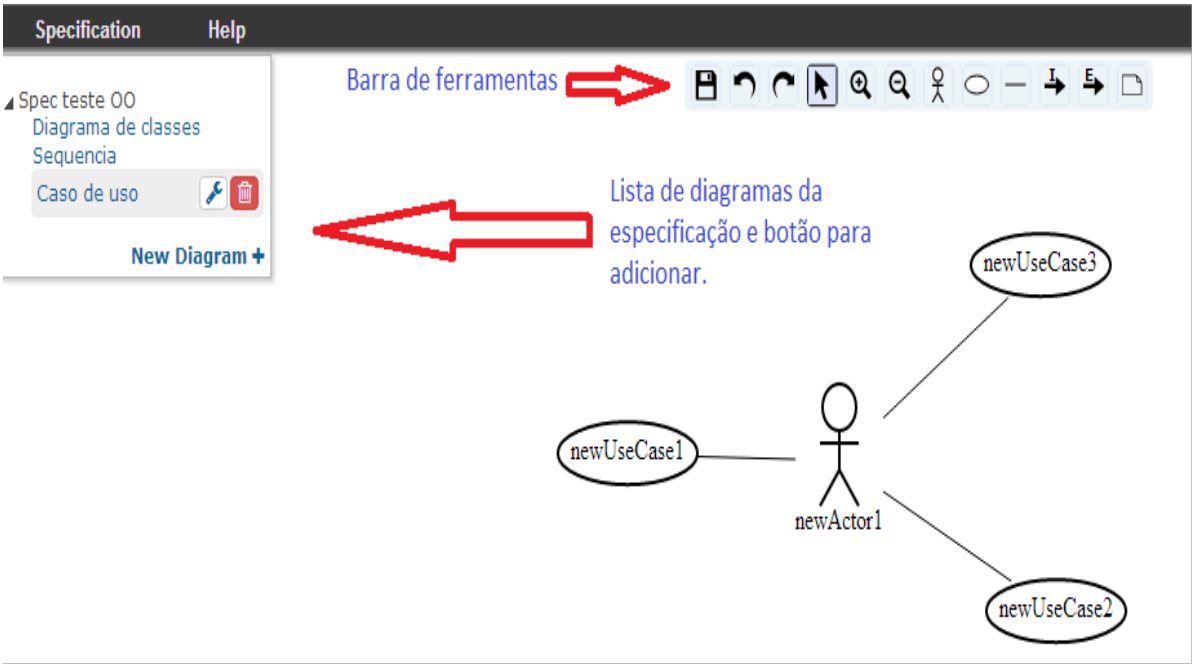

Figura 20: Edição de diagramas

7.4.3 Edição de elementos

Para se editar um elemento, basta passar o mouse sobre o ele, ou dar um toque, no caso de ser um *tablet.* Ao focar no elemento, é exibido um menu com opções específicas de cada elemento. A figura 21 exibe uma classe sendo focada, exibindo as opções de apagar, adicionar estereótipo, adicionar atributo e adicionar método.

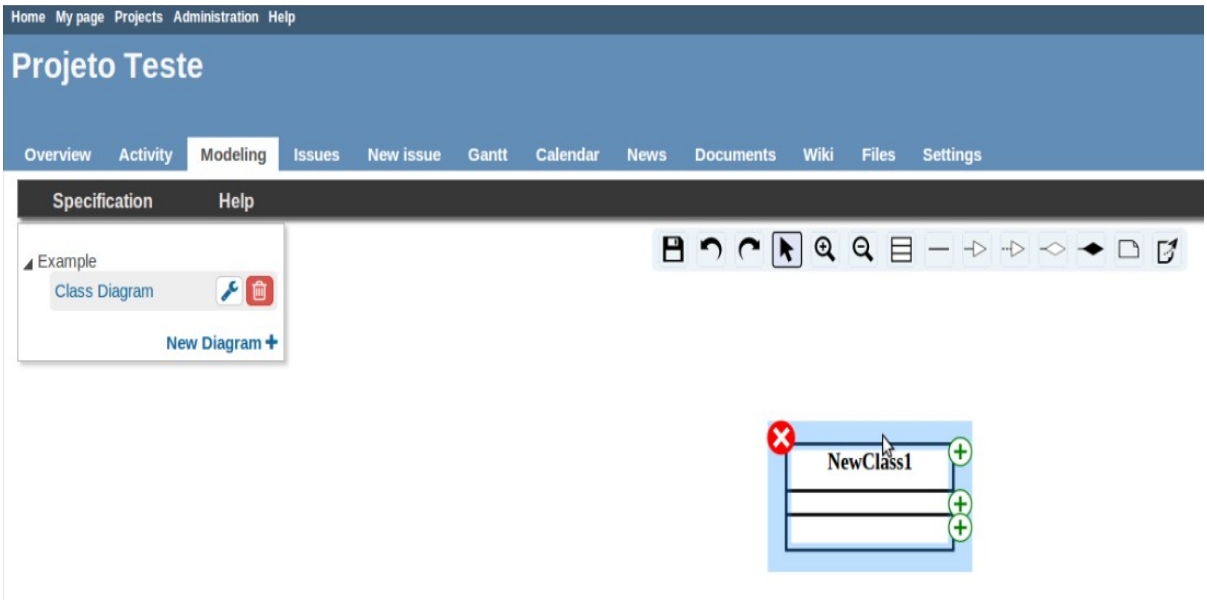

Figura 21: Criação e edição de diagramas

7.4.4 Edição de Links

Links podem ser editados no formato da ligação, no texto (para os que tiverem), na origem e no destino. A figura 22 mostra o formato de uma ligação sendo editada.

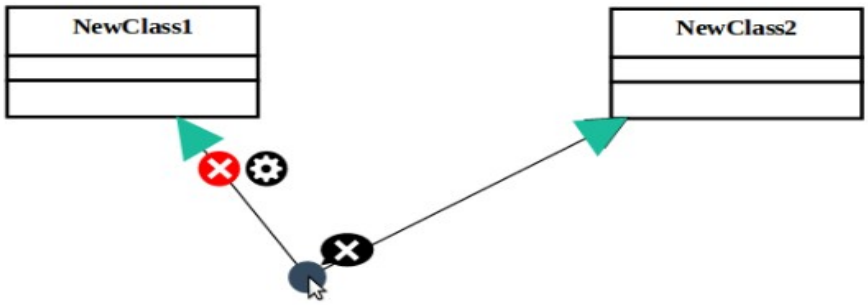

Figura 22: Edição de link

7.4.5 Geração de código e repositório

Para diagramas de classes, existe a funcionalidade de gerar código Java a partir do diagrama e mandar para um repositório GIT já com a estrutura de pasta especificada na criação do diagrama. Para poder esta funcionalidade, deve se gerar uma chave pública e privada no servidor onde fica a aplicação e depois especificar a chave pública no repositório GIT.

Dependendo do diagrama, na tela de criação há opções de pastas no repositório em que é gerado o código e na barra de ferramentas há opções de *commit* para enviar o código para o repositório.

7.5 Possibilidade de integração com outro back-end

A parte do cliente foi feita totalmente desacoplada do servidor, isso significa que é possível utilizá-la em outro back-end facilmente precisando somente configurar as rotas, se as mesmas forem diferentes das já estabelecidas. No desenvolvimento Web geralmente se utiliza uma linguagem de back-end e javascript para front-end. Como o trabalho pesado na construção do BR-Dia foi feito com javascript, este código pode ser aproveitado em outro back-end, embora o autor deste trabalho aconselhe a utilizar o Ruby pela agilidade, pela estrutura já pronta, pelas facilidades do framework Rails e possibilidade de criação de geradores, os quais são suportados pelo Rails.

8 Avaliação

Avaliação de software é verificar o cumprimento de requisitos através de termos quantitativos e qualitativos. Avaliar uma ferramenta CASE é verificar se ela está de acordo com os requisitos estabelecidos pelas necessidades no(s) processo(s) de desenvolvimento de software em que a ferramenta se propõe a suprir. No caso deste trabalho, é uma ferramenta para modelagem de software com UML.

Segundo Suryn (2014), as partes essenciais da avaliação de qualidade de software são o modelo de qualidade, o método de avaliação, métrica de software e ferramentas de apoio.

Avaliação é uma tarefa complexa e deve seguir um processo bem definido e ter um plano. Este trabalho se baseará em algumas partes da norma ISO/IEC 9126, abordada por Suryn (2014), o qual define um modelo de qualidade interna e externa do software. A figura 23 detalha a características desta norma. As ferramentas a serem avaliadas serão a deste trabalho, BR-Dia, Gliffy (ferramenta online), Astah e Violet UML Editor. Os critérios para a escolha desses softwares foram:

- Gratuidade;
- Suporte à UML;
- Ser online ou, no caso de uma aplicação desktop, ter algum recurso avançado;
- Ser relativamente conhecida e estar bem posicionada nos resultados da pesquisa do Google.

Como o foco desta monografia é o desenvolvimento da aplicação, esta avaliação será focada na relevância da ferramenta no contexto estabelecido, ou seja, uma ferramenta CASE flexível, portável, livre e útil ao ambiente acadêmico principalmente. A medição, terá critério binário e todos os testes terão o mesmo peso. A seguir é abordado cada teste feito com as ferramentas mencionadas.

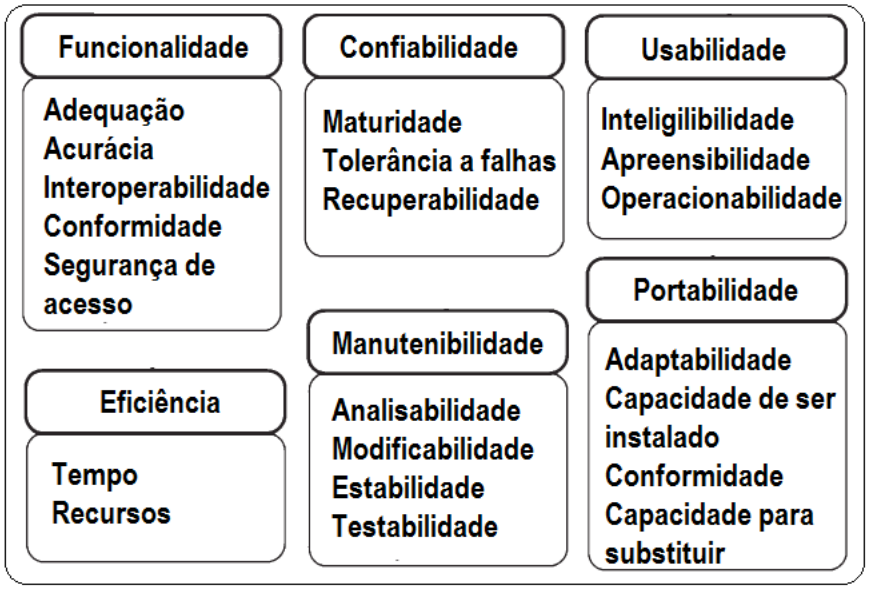

Figura 23: Características de qualidade

8.1 Funcionalidade

Destaques:

- Acurácia: o BR-Dia até a presente realização deste trabalho apresentou alguns erros no diagrama de sequência, podendo gerar inconsistência nos dados.
- Conformidade: o Violet UML Editor e o BR-Dia tem alguns diagramas com representações faltando e o Gliffy oferece somente suporte à UML 1.0 em sua versão gratuita.
- Segurança de acesso: o Violet UML Editor não pontuou, pois permitiu editar um arquivo que estava aberto.

	Pergunta	BR-Dia	Gliffy	Astah	Violet UML Editor
Adequação	Faz aquilo que se propõe?	X	X	X	X
Acurácia	Faz de maneira adequada?		X	X	X
Interoperabilidade	Interage com os sistemas especificados	X	X	X	X
Conformidade	Está de acordo integralmente com o padrão da UML2?			X	
Segurança de acesso	Restringe o acesso aos dados?	X	X	X	
Total		3	4	5	3

Tabela 4: Comparação de funcionalidade

8.2 Usabilidade

Em todas as ferramentas foi possível aprender e editar facilmente os diagramas UML suportados. Como houve empate, sinaliza a necessidade de ser fazer um teste de usabilidade mais aprofundado, utilizando as 10 heurísticas de Nielsen, por exemplo. Como esse não é o escopo deste trabalho, isso certamente pode ser um trabalho futuro. Então este item não será somado ao resultado total.

8.3 Portabilidade

Destaques:

- Capacidade para ser instalado: Astah e Violet UML Editor não pontuaram, pois por serem em Java precisam que o ambiente possua Java também, o que inviabiliza o uso em alguns tablets e os os coloca em desvantagem sobre o Glify e o BR-Dia, que basta estarem instalados em um servidor que rodam em diversos dispositivos que tenham algum navegador que implemente o HTML5. Como o critério é binário, o fato da necessidade da instalação da máquina virtual do Java representar um passo a mais para a execução do software, já é o suficiente para colocar as ferramentas em Java em desvantagem e desempatar este teste. Pois as ferramentas Web necessitam apenas do acesso e funcionam em uma gama maior de dispositivos.

	Pergunta	BR-Dia	Gliffy	Astah	Violet UML Editor
Adaptabilidade	Fácil de adaptar?	X	X	X	
Capacidade para ser instalado	É fácil instalar em outros ambientes?	X	X		
Conformidade	Está de acordo com os padrões de portabilidade?	X	X	X	X
Capacidade para substituir	É fácil para usuários de outros softwares usarem?	X	X	X	X
Total		4	4	3	2

Tabela 5: Comparação de portabilidade

8.4 Manutenibilidade, confiabilidade e eficiência

A manutenibilidade não foi testada porque nem todas as ferramentas analisadas tem o código-fonte aberto e pela falta de conhecimento interno relacionado com o desenvolvimento das ferramentas avaliadas, é inviável fazer um teste confiável.

O item confiabilidade não foi testado pelo fato de a ferramenta BR-Dia ser resultado de um trabalho acadêmico com limitações técnicas e de recursos, então esteve fora do escopo deste trabalho testes aprofundados que verifiquem com robustez todos as possíveis falhas da aplicação, o que inviabiliza realizar um teste de confiabilidade robusto.

O item eficiência não foi tabelado pelo fato da complexidade de testar a eficiência e consumo de recursos de ferramentas sobre diferentes ambientes, algumas executando em máquina virtual, outras no navegador e com apoio do servidor. E sem contar em fatores externos como oscilações da rede. Tudo isso dificulta a obtenção de um resultado confiável.

8.5 Tabela de comparação de características específicas

A tabela 6 apresenta quais as características da ferramenta BR-Dia estão presentes nas outras ferramentas submetidas nos testes anteriores. Destaca-se o fato de nenhuma ter suporte a geração de código-fonte, recurso geralmente encontrado nas versões pagas.

Carcterísticas	Ferramentas			
	BR-Dia	Gliffy	Astah	Violet UML Editor
Software Livre	Sim	Não	Não	Sim
Geração de código-fonte	Sim	Não	Não	Não
Integra com outras ferramentas?	Sim	Sim	Sim	Sim
Funciona em tablets?	Sim	Sim	Não	Não

Tabela 6: Comparação com características específicas

8.6 Resultado da avaliação

A tabela 7 mostra a soma dos resultados das avaliações de funcionalidade e portabilidade. Pelo resultado podemos ver que o Violet UML Editor teve o pior resultado, mas se fossemos aplicar os testes descartados, ele teria um resultado melhor que o BR-Dia, pois é um produto, tem uma comunidade ativa e sólida e o BR-Dia é um protótipo. A avaliação foi direcionada para avaliar os objetivos deste trabalho.

O Gliffy teve o melhor resultado, mas se colocássemos um peso maior sobre o requisito de suportar UML 2, o Astah poderia empatar ou ter um resultado melhor, porque é o único dentre as ferramentas da avaliação que tem suporte para diagramas da UML 2, o que é um diferencial dentro do escopo deste trabalho.

	BR-Dia	**Gliffy**	**Astah**	**Violet UML Editor**
Soma dos resultados	7	8	8	5

Tabela 7: Total das avaliações

No geral a Ferramenta BR-Dia teve um resultado satisfatório, durante os testes destacou-se o fato de sua interface ser bastante rápida perante as outras aplicações e como o item eficiência ficou de fora da avaliação, isso não somou no resultado final. O Violet UML Editor teve um desempenho inferior perante o Gliffy e o Astah, os quais é possível perceber facilmente que são superiores. Embora esta avaliação não seja sobre todos os aspectos das ferramentas, serviu para evidenciar os recursos faltantes e erros na ferramenta BR-Dia que seguem na lista a seguir:

- **Capacidade para salvar mudanças no formato das ligações:** é feito no cliente, mas não salva, os links voltam a serem retos sem formato.
- **Espaço para notificação de erros:** os erros são gerados, mas não tem uma área de exibição.
- **Capacidade para salvar palavras ligadas a links:** pode ser criada no cliente, mas ainda não é salva
- **Diagrama de sequência incompleto:** este diagrama não está completo, só possível fazer diagramas simples e limitados.

9 Conclusão

Este trabalho consistiu na produção deste documento e de uma ferramenta para modelagem de software extensível, integrada com um repositório GIT e com a ferramenta de gerência de projetos Redmine, a interface com o usuário é totalmente desacoplada do servidor. A arquitetura da ferramenta foi construída voltada à adição de novos tipos de diagramas e especificações. Embora seja um protótipo, a ferramenta foi capaz de modelar os diagramas UML utilizados neste trabalho.

9.1 Resultados Obtidos

- **Software livre para modelagem:** Suporte para Cinco diagramas da UML;
- **Geração de código-fonte:** Código Java gerado a partir do diagrama de classes em que for executado a função de *commit;*
- **Integração com o Redmine:** A ferramenta é um plugin do Redmine e tem suas especificações ligadas a projetos do Redmine;
- **Integração com o Repositórios GIT:** É possível gerar diagramas de classe e enviar diretamente para repositórios GIT;
- **Compatibilidade com os navegadores modernos, incluindo tablets:** A ferramenta é compatível com *tablets.* Neste trabalho foi testado somente no ipad;
- **Base para a construção de novos diagramas:** Conjunto de superclasses e módulos para auxiliar a criação de novos tipos de diagramas.

9.2 Avaliação

O desenvolvimento de um aplicativo Web para modelagem não é algo trivial, visto que existem poucos exemplos e o modo de desenvolver foge um pouco da zona de conforto do desenvolvimento Web tradicional. A arquitetura e escolha de tecnologias ocuparam boa parte do tempo de planejamento e o desenvolvimento da parte gráfica e de interação com o usuário consumiu cerca de 60% do tempo de desenvolvimento.

Na parte de conceitualização, generalização e base de projeto os conceitos e o meta-modelo do *framework* OCEAN trouxeram uma boa base para o projeto, o que pode ser notado pela facilidade que houve para organizar a arquitetura da aplicação.

O resultado, em termos de software, é uma ferramenta para modelagem que tem bom desempenho e qualidade gráfica, e tem alguns recursos que a diferencia das outras como geração de código-fonte diretamente no sistema de controle de versão e integração com o Redmine.

Embora tenha alguns recursos faltando ainda, principalmente no diagrama de sequência, a ferramenta foi capaz de modelar os diagramas utilizados neste trabalho, tanto para ilustrar conceitos como para detalhar o projeto da aplicação. E teve também um resultado satisfatório nos testes, visto que as outras ferramentas eram softwares consolidados no mercado.

9.3 Trabalhos Futuros

- **Extração de um *framework* orientado a objetos:** Embora a ferramenta BR-Dia tenha um conjunto de superclasses que oferece funcionalidades para fácil adição de novos diagramas, essas superclasses ainda não estão organizadas como um *framework*, seria interessante a extração de um *framework* neste contexto.

- **Criação de geradores de código-fonte:** Rails fornece poderosos recursos para a construção de geradores, agilizaria bastante o desenvolvimento de novos diagramas se a ferramenta agregasse um gerador que construísse toda a estrutura e esqueleto de código necessário para adicionar novos diagramas.
- **Adição de novas especificações e diagramas:** Visto que uma base de desenvolvimento já foi construída, nada mais natural que a adição de um grande número de novas especificações e diagramas.
- **Realização de teste de usabilidade:** Como não foi escopo deste trabalho, para melhorar a qualidade da ferramenta, seria interessante a aplicação de um teste de usabilidade completo que aplique as heurísticas de Nielsen, por exemplo.
- **Outras integrações:** Como se trata de uma aplicação Web, BR-Dia pode ser integrado nos mais variados sistemas Web.

9.4 Considerações Finais

O objetivo e motivação deste trabalho desde a sua concepção foi trazer alguma contribuição em termos de ferramentas utilizadas no meio acadêmico e na comunidade de software livre e ao mesmo tempo provar que se pode construir uma ferramenta eficiente em um curto espaço de tempo utilizando tecnologias e métodos adequados ao contexto e objetivo da ferramenta.

O software produzido e este documento constituem subsídio para outros trabalhos acadêmicos, softwares e pesquisas. Embora a ferramenta ainda precise amadurecer como software, o principal valor deste trabalho é o seu projeto e a criação de um ponto de partida para a criação de um produto, requisito fundamental para a aceitação do conceito e da ferramenta na comunidade e meio acadêmico.

- O endereço do repositório do plugin BR-Dia é: **https://github.com/fabiocariati/brdia**

APÊNDICE A – Diagramas

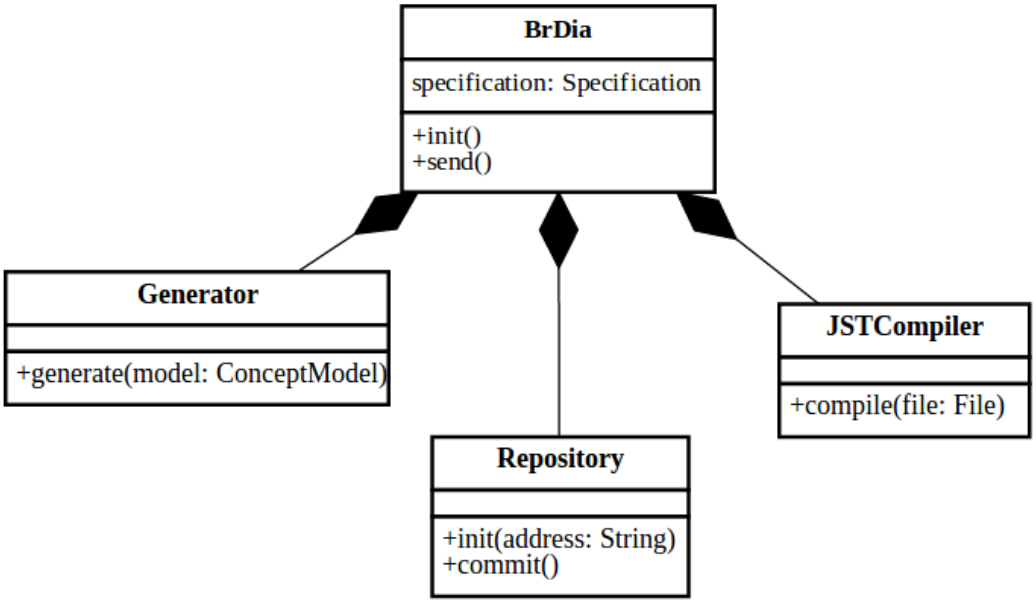

Figura 24: Diagrama de Classes do módulo BrDia

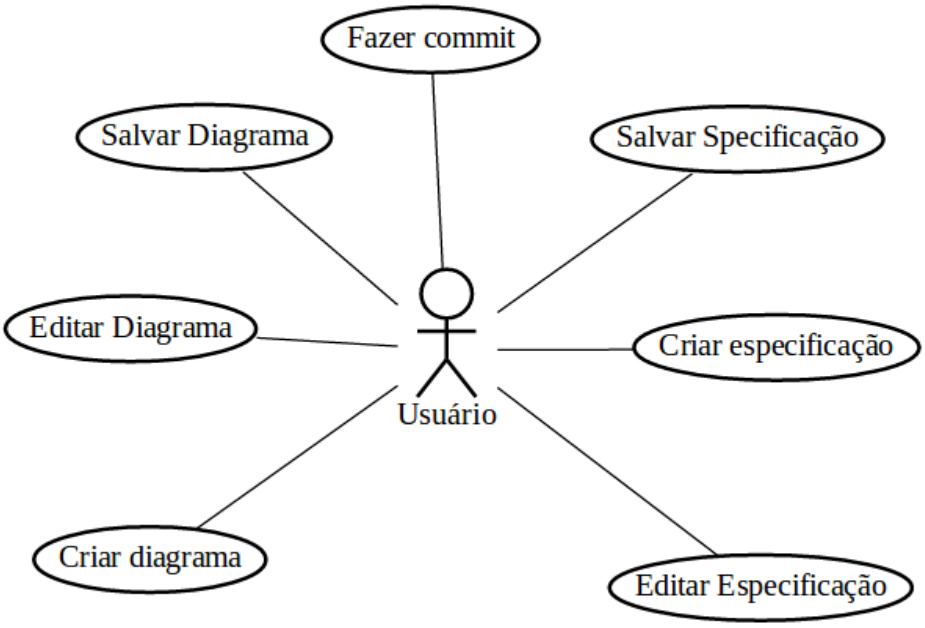

Figura 25: Diagrama de caso de uso para interação com a ferramenta

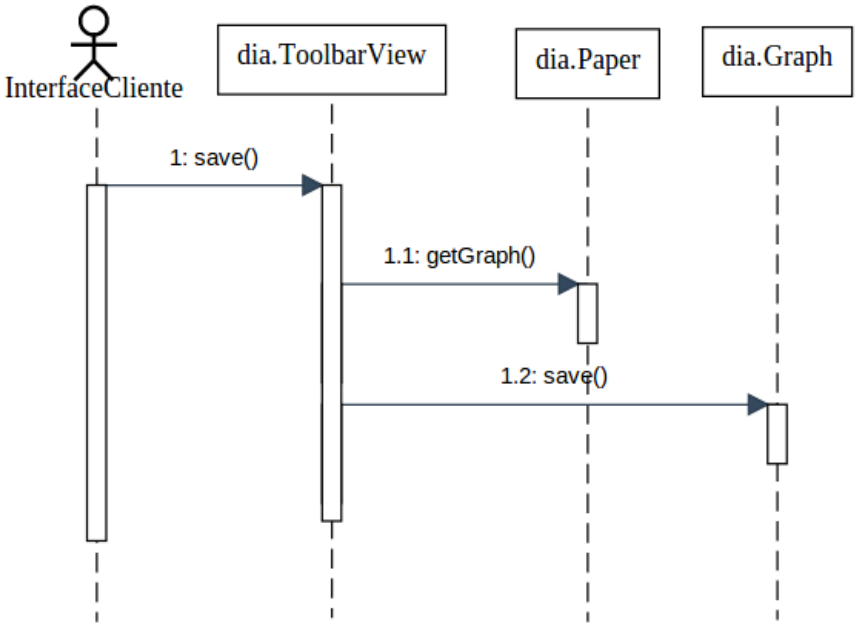

Figura 26: Diagrama de sequência da ação de salvar

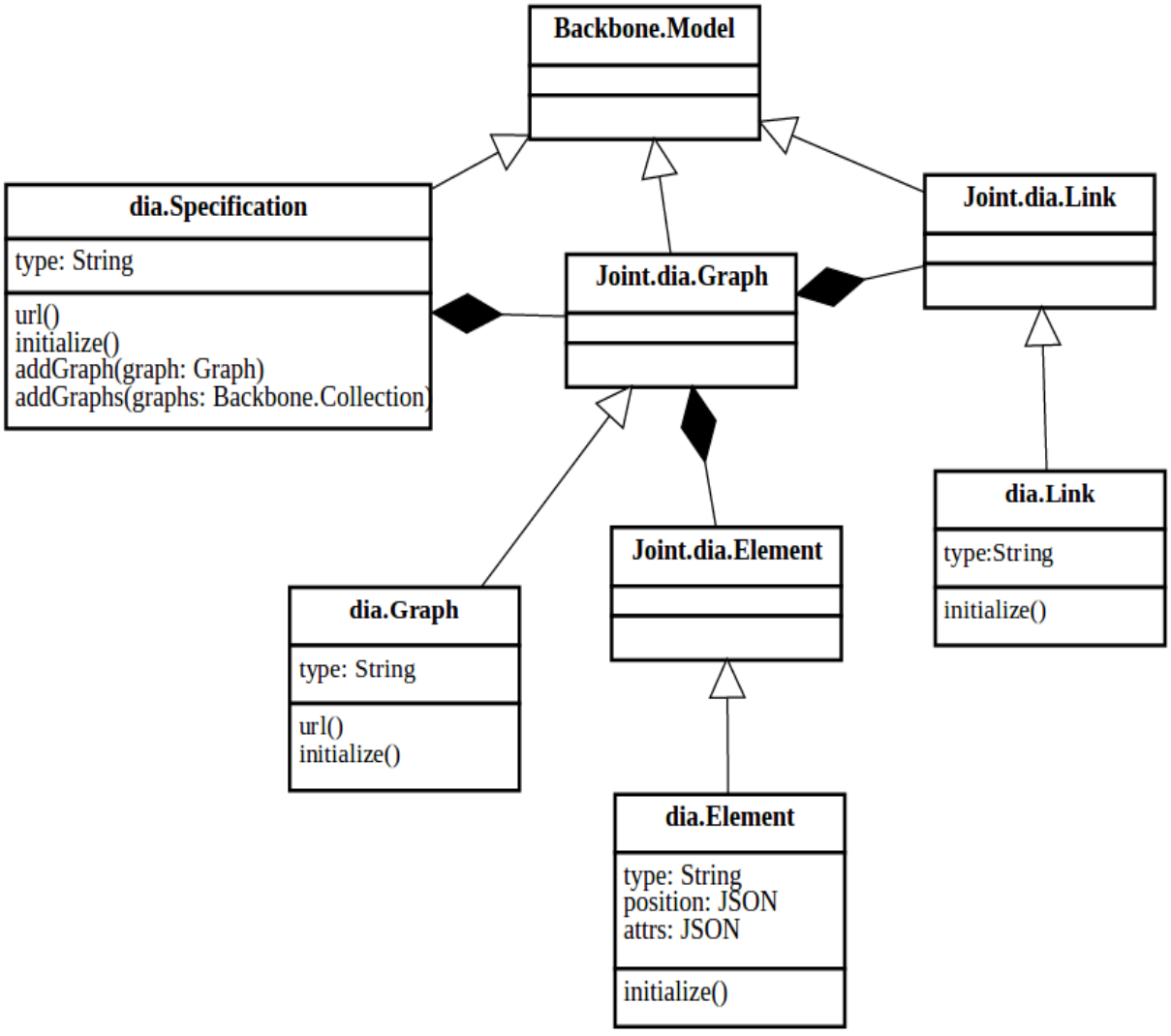

Figura 27: Diagrama de classes dos models no cliente

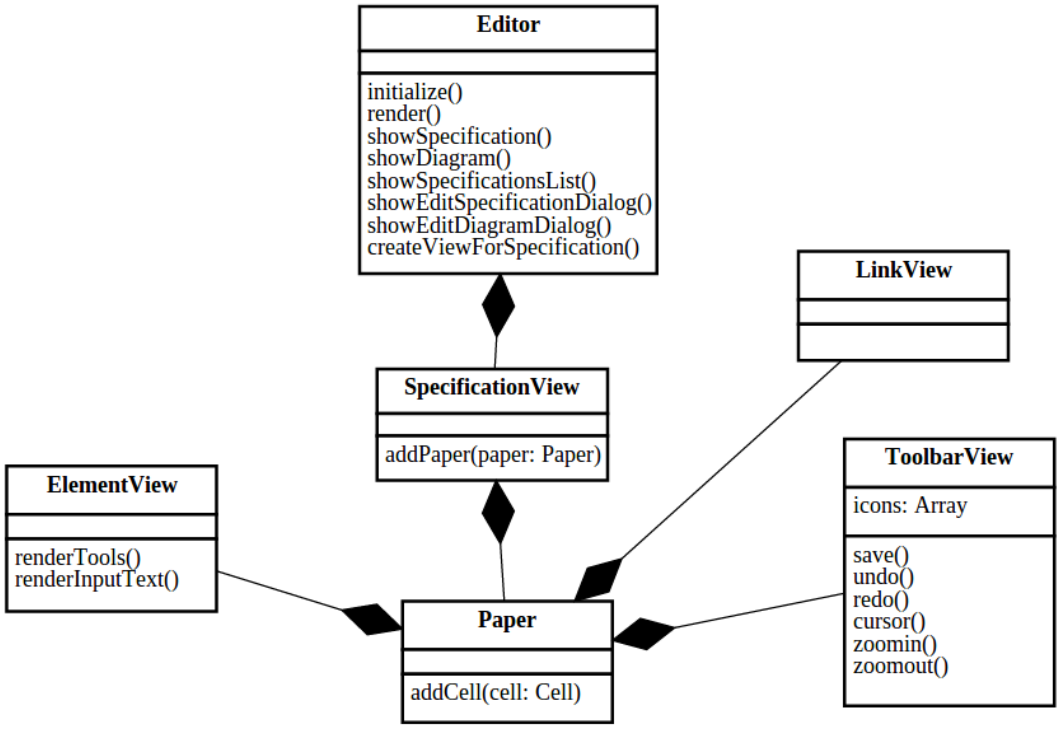

Figura 28: Diagrama de classes das views no cliente

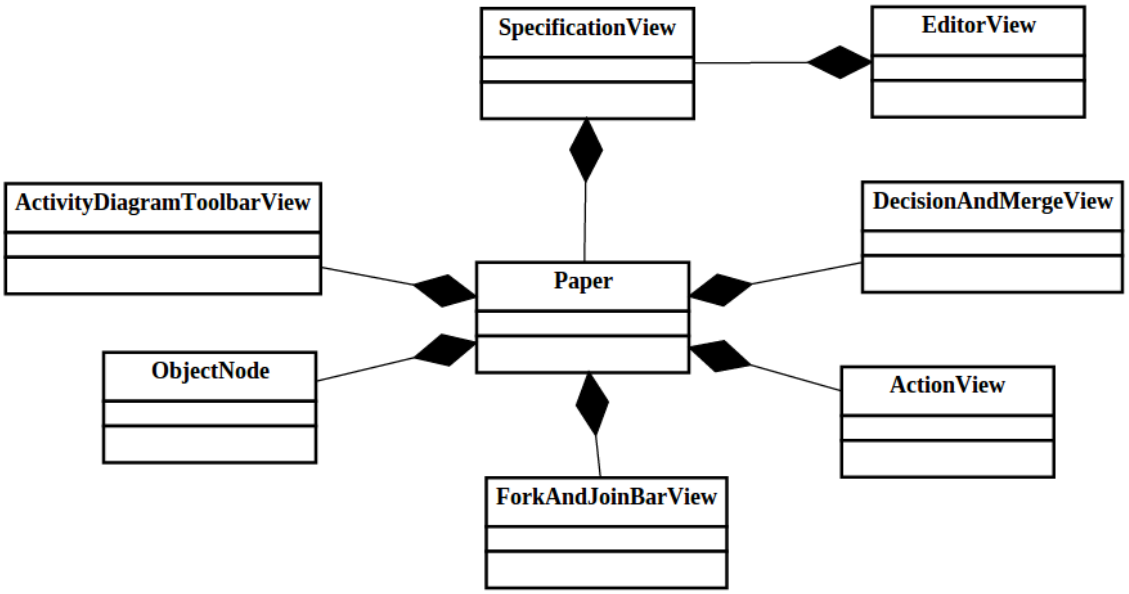

Figura 29: Exemplo da composição de views de um diagrama

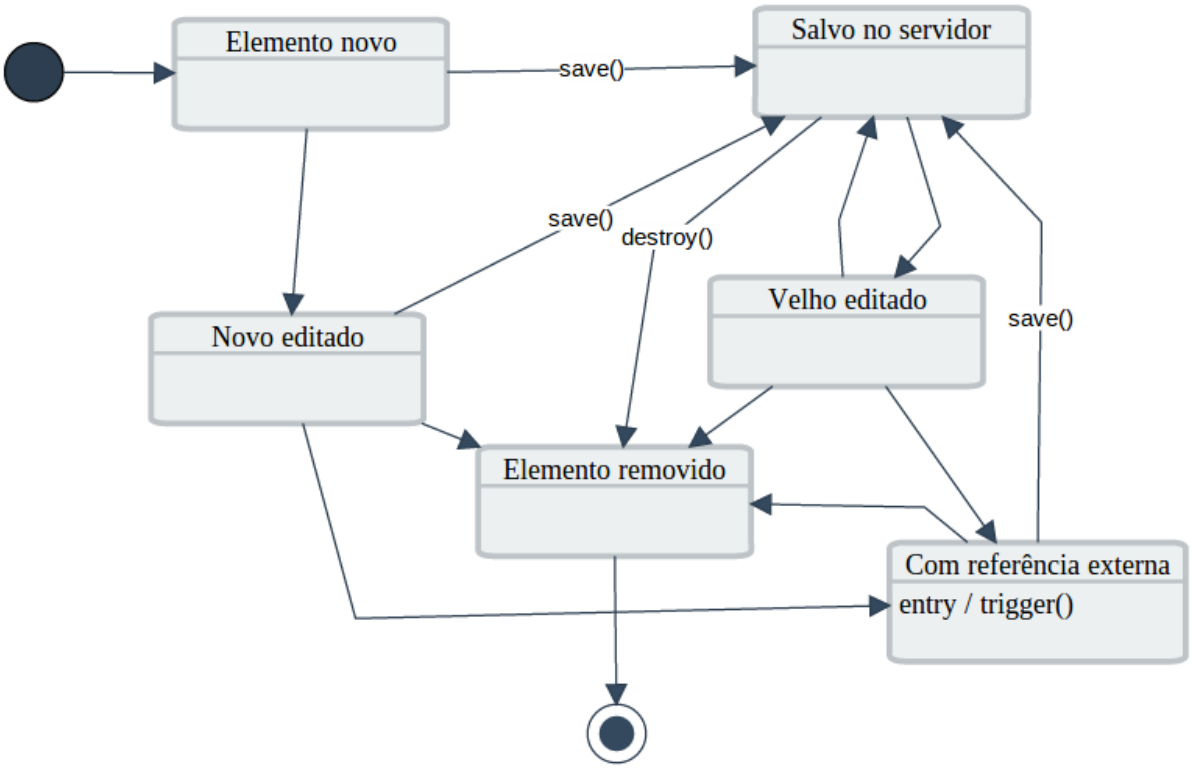

Figura 30: Diagrama de máquina de estados dos estados de um elemento

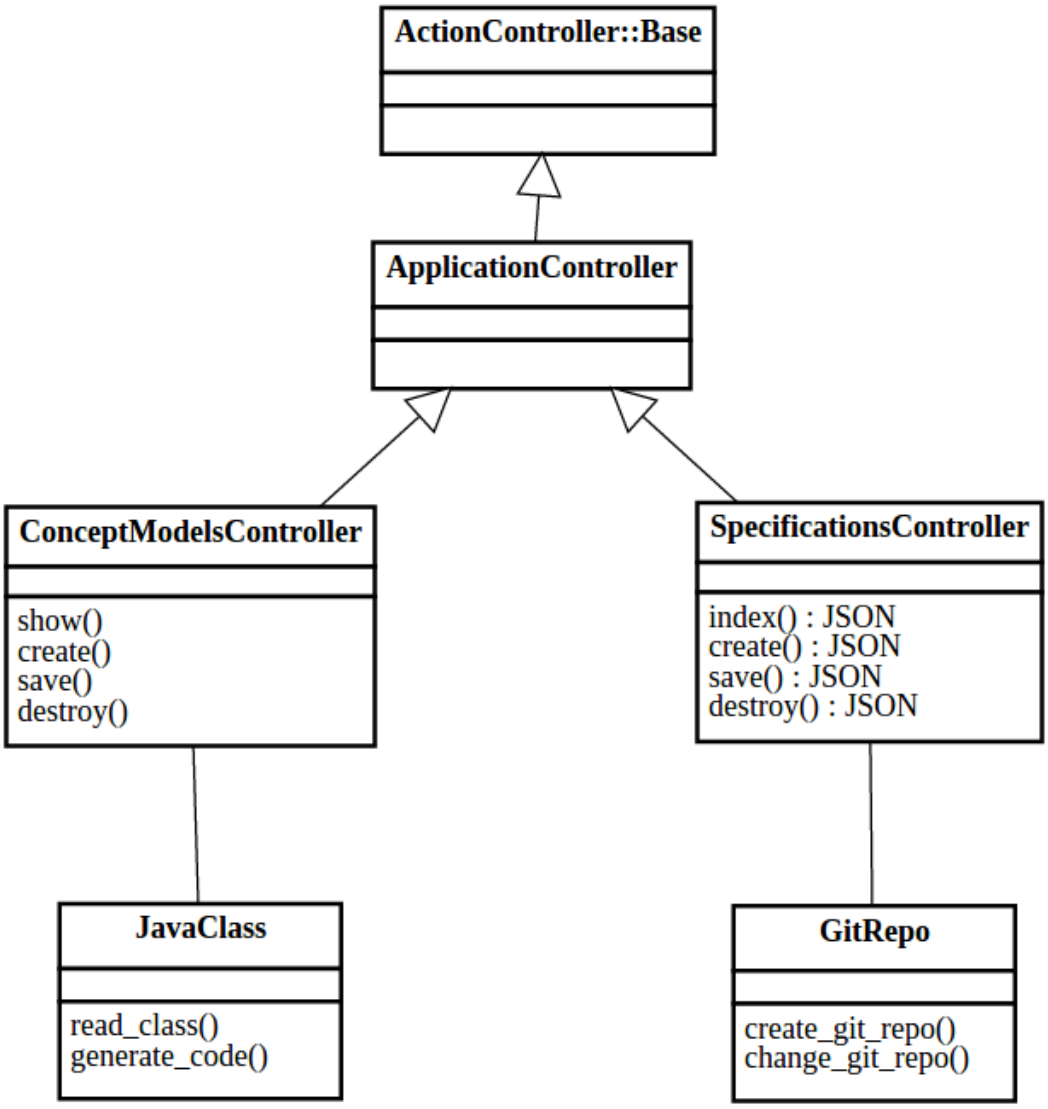

Figura 31: Diagrama dos controllers do lado do servidor.

APÊNDICE B – Construção de aplicações gráficas e interativas com HTML5

B.1 Introdução

A construção de uma aplicação Web consiste basicamente em determinar como será o servidor e como será o cliente. A tendência atual é que a parte do cliente seja responsável por toda a lógica e interface da aplicação e o servidor pelo armazenamento e transferência de dados para o cliente. O BR-Dia utiliza a linguagem Ruby no servidor e JavaScript no cliente. Os dados são armazenados em um banco de dados SQL, o qual é acessado pelo framework Rails.

B.2 Persistência e transferência dos dados

Uma aplicação Web que persiste dados em um servidor, precisa definir como esses dados são enviados do cliente para o servidor e como esses dados são recuperados do servidor e enviados para o cliente.

Em uma aplicação focada em *client-side*, basicamente a principal função do servidor é o armazenamento de dados. A renderização de páginas, comportamento e interação com o usuário, é responsabilidade do lado do cliente.

No servidor do BR-Dia, foi utilizado Ruby on Rails e os dados são armazenados em um banco de dados SQL e transferidos para o cliente através de uma arquitetura REST, a qual envia os dados em formato JSON para o cliente. No cliente, os dados são recebidos pela bilioteca Backbone.js através do roteamento para seus *models* (instância de *Backbone.Model*).

B.3 MV* frameworks

Com o aumento da complexidade das aplicações Web e as novidades do HTML5, surgiram os *MV* frameworks,* os quais são bibliotecas e frameworks JavaScript que têm estrutura semelhante ao MVC, cada um com suas adaptações, mas todos possuem *model e view* (por isso MV*). Alguns exemplos são: Backbone (utilizado neste trabalho), Ember, AngularJS e muitos outros. Para ajudar na escolha do framework, foi criado o site TodoMVC (2013), um projeto que oferece a mesma aplicação denominada *Todo,* implementada na maioria frameworks MV * da atualidade (TodoMVC, 2013).

B.4 Recursos gráficos do HTML5

O modo mais tradicional para se apresentar páginas na Web é utilizar a marcação HTML com a folha de estilos CSS, mas o HTML 5 tem novos recursos para a apresentação de páginas, como o Canvas e o SVG.

O elemento Canvas é responsável por estabelecer uma área para desenhar dinamicamente imagens em uma página. O gráfico é desenhado e animado através da linguagem JavaScript. Este normalmente é utilizado para aplicações gráficas complexas e jogos 3D.

O SVG é um XML que representa uma linguagem para gráficos vetoriais, os quais permitem figuras em alta resolução. O SVG tem a vantagem de ser aberto, gratuito, e padronizado pela W3C, com o apoio de empresas como Apple, IBM, Microsoft e Adobe. O Joint.js, utilizado neste trabalho, é uma biblioteca JavaScript especializada em diagramas desenhados com SVG.

A ferramenta BR-Dia utiliza o templates HTML e SVG para renderizar seus diagramas, as *views* Backbone carregam os templates e compilam aplicando os dados

dos *models*. A figura 32 mostra um template de um elemento SVG que representa um círculo, os atributos *box* e *width*, vem da *view* que representa o elemento no cliente.

```
<g class="element" transform="translate(<%= box.x %>, <%= box.y %>)">
    <circle r="<%= width / 2 %>" stroke="black" stroke-width="2" fill="white"/>
</g>
```

Figura 32: Template SVG de um elemento círculo

B.5 Fontes de informação

A tabela 8 exibe as principais fontes de dados da Web para se produzir uma aplicação gráfica e interativa com HTML5 e Ruby on Rails.

Descrição	Link
Especificação HTML5	http://www.w3.org/TR/html5/
Tutorial de SVG da W3C	http://www.w3schools.com/svg/
Documentação do Rails	http://rubyonrails.org/documentation/
Documentação do Ruby	https://www.ruby-lang.org/en/documentation/
Tutorial de JavaScript da W3C	http://www.w3schools.com/js/
Projeto TodoMVC – lista de frameworks MV*	http://todomvc.com/

Tabela 8 - Fontes de informação na Web

B.6 Conclusão

Para se construir uma aplicação Web para edição gráfica, algumas técnicas fogem da maneira tradicional de se desenvolver para a Web, visto que editores gráficos profissionais normalmente residem na plataforma desktop. Mas é uma área promissora e ainda há uma carência de ferramentas gráficas em HTML5. A maioria das grandes empresas de tecnologia apoia o HTML5, o que garante compatibilidade com os navegadores atuais.

APÊNDICE C - Expandindo a ferramenta BR-Dia

O core da aplicação foi desenvolvido para se adicionar facilmente novos diagramas e elementos. A ideia é que se possa criar perfis de linguagens e frameworks para se ter diagramas personalizados e mais relevantes ao contexto, padrão e metodologia de desenvolvimento. Também pode-se adicionar outros tipos de diagramas relacionados com o processo de desenvolvimento de software como o Diagrama Entidade Relacionamento, por exemplo.

Este capítulo é dedicado para descrever a estrutura da aplicação em funcionamento através dos passos básicos para se adicionar novos tipos de diagramas. A seguir é detalhado o roteiro para se criar um novo tipo especificação e um editor de Grafos orientados.

C.1 Exemplo de criação de tipo de especificação

Criar uma especificação consiste em estabelecer quais modelos irão fazer parte da mesma, no nosso caso a especificação exemplo se chamará Generic Specification, que conterá o Grafo orientado, representado no retorno do método *concept_types,* o qual retorna um array de strings que representam os diagramas que fazem parte desta especificação. Esse é um exemplo simples, mas em casos mais complexos essa classe teria funções relacionadas ao tipo de modelagem, como funções de geração de código-fonte, repositórios e outras customizações. A figura 33 mostra o código da classe que representa os dados da especificação *GeneralSpecification*.

```
class GeneralSpecification < Specification
  def concept_types
    ['DirectedGraph']
  end
end
```

Figura 33: Código da especificação

C.2 Exemplo de criação de tipo de diagrama

O exemplo aqui é bem simples e objetiva passar uma visão geral do processo de desenvolvimento de novos diagramas. Trata se de um editor de Grafo Orientado, o qual consiste em círculos com um texto dentro conectado com outros círculos através de setas direcionadas. O diagrama de classes simplificado da figura 34 ilustra o *models* desse diagrama.

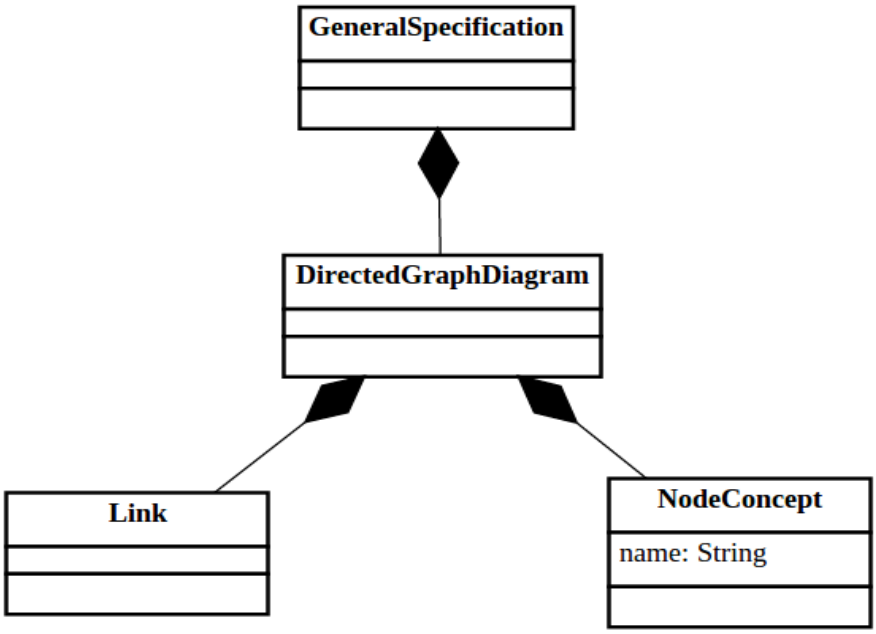

Figura 34: Diagrama de classes simplificado dos modelos do editor de grafo

A seguir os passos para criar o editor para esse diagrama:

OBS: Todos os comandos de geração de código foram executados no terminal do Ubuntu 13.10 de dentro da pasta do projeto.

1 – Criar a classe *DirectedGraphDiagram*, que estende *ConceptModel*

Essa classe conterá a informação de quais tipos de modelos farão parte do diagrama, no nosso caso *NodeConcept e Link*. E também conterá a informação de qual especificação o gráfico pertence. O código-fonte desta classe pode ser visto na figura 35.

A criação de uma classe de modelo de diagrama se resume nos seguintes passos:

- Especificar qual a entidade que estende *Specification* que o diagrama pertence, que pode ser genérica, no caso do diagrama ser usado em mais de uma especificação
- Especificar os elementos
- Especificar os links
- Definir a descrição do diagrama

```
class DirectedGraphDiagram < ConceptModel
  belongs_to :object_oriented_specification
  has_many :node_concepts, class_name: 'NodeConcept',
           foreign_key: :concept_model_id, dependent: :destroy
  has_many :links, class_name: 'Link', foreign_key: :concept_model_id, dependent: :destroy

  def self.description
    'Directed Graph'
  end
end
```

Figura 35: Código de DirectedGraphDiagram

obs: Métodos como has_many e belongs_to são métodos de classe herdados de ActiveRecord::Base, que geram outros métodos quando a classe é carregada no servidor.

2 – Gerar os modelos que farão parte do diagrama

Assim como Rails, o Redmine possui um gerador de modelos, que cria inclusive as tabelas no banco de dados. O comando a seguir irá gerar o modelo do nodo:

rails g redmine_plugin_model modeling node_concept x:integer y:integer name:string concept_model:references.

Após a execução deste comando, deve se editar a classe gerada, *NodeConcept* para estender a classe *Concept* e funcionar corretamente na estrutura de modelos. Quando a classe *DirectedGraphDiagram* for manipular os elementos, ela irá chamar os métodos herdados da classe *Concept*.

Criar uma classe de conceito consiste nas seguintes etapas

- Definir os atributos
- Gerar um modelo com os atributos definidos
- Modificar a classe para estender *Concept*

3 – Executar o comando para criar as tabelas no banco de dados:

rake redmine:plugins:migrate

Este comando executa as classes de migração geradas no comando do passo 2. Após a execução deste comando, as tabelas referentes aos modelos serão criadas no banco de dados e adicionadas ao *Schema.*

4 – Criar um elemento do tipo *dia.Element* no cliente

Um elemento do tipo *dia.Element* é um *model* da biblioteca Backbone.js e é responsável por especificar os atributos do elemento e e quais destes serão buscados e enviados para o servidor salvar e buscar no banco de dados. No nosso caso, o elemento é general.Node, especificado no código da figura 36.

Como a especificação deste diagrama só tem um diagrama, este elemento não precisará notificar o elemento *Specification* sobre suas mudanças, apenas notificará para o seu diagrama, que já é o comportamento padrão. Mas se fosse necessário, seria feito com o método *trigger()* do Backbone.js.

Criar um elemento consiste nas seguintes etapas:

- Criar um protótipo que estende dia.Element
- Definir os atributos que representam dados no servidor
- Definir os atributos utilizados somente no cliente(geralmente relacionado a estilos e atributos gráficos).

```
general.Node = dia.Element.extend({

    defaults: joint.util.deepSupplement({

        type: 'general.Node',
        attrs: {
            'circle': { fill: '#FFFFFF', stroke: 'black' },
            'text': { 'font-size': 14},
            '.general-node-name': {
                'font-size': 16, 'font-family': 'Times New Roman', fill: '#000000'
            }
        },
        name: ''
    }, dia.Element.prototype.defaults)
});
```

Figura 36: Código do elemento Node

No código da figura 36, o atributo *type* representa qual a tabela no banco de dados que será armazenado o modelo, neste caso pela convenção de nomes

determinada no projeto, a partir de *'general.Node'* é inferido a tabela da classe *'NodeConcept'* no servidor. Dentre os outros atributos *attrs* define os atributos do estilo e é utilizado somente no cliente e não é enviado para o servidor e *name* é o texto do nodo e representa um campo no banco de dados.

5 – Criar uma *view* no cliente do tipo *dia.ElementView*

A *view*, no cliente, é responsável por renderizar os templates e tratar os eventos da interface. No nosso caso, como é um exemplo simples, toda a parte de interação com o usuário já é suprida por *dia.ElementView.* Precisamos somente configurar qual o atributo que corresponde ao campo textual que será especificado pelo template no passo 6.

```
general.NodeView = dia.ElementView.extend({

    initialize: function() {
        var self = this;
        this.createTextFieldFor('name');

        dia.ElementView.prototype.initialize.apply(this, arguments);

        this.model.on('change:name', function() {
            self.trigger('change:attrs');
        });
    }
});
```

Figura 37: Código do elemento NodeView

6 – Criar um *template* com o visual dos elementos do diagrama.

Templates são compilados ao inicializar a aplicação no servidor em um arquivo único carregado somente uma vez no cliente. A separação dos templates em arquivos diferentes serve para organizar o código, pois o que é enviado para o cliente é o

arquivo único gerado. É importante observar que para carregar os templates é necessário seguir duas regras de convenção de nomes na sua construção:

- Elemento *div* externo com a classe “jst-template” com o atributo *name* que representa a *view* que carregará este elemento;
- Elementos *div* internos que representam itens gráficos;

A figura a seguir mostra o código-fonte do arquivo *directedgraph.jst.html,* o qual deve armazenar os templates do diagrama de Grafo Direcionado.

```
<div class="jst-template" name="Node">
    <div class="item" name="markup">
        <circle r="20" stroke="black" stroke-width="2" fill="white"/>
        <text class="general-node-name" />
    </div>
</div>
```

Figura 38: Código do template

7 – Criar *view* da barra de ferramentas

Nesta *view* (figura 39) é configurado as ações específicas deste diagrama. Para a transição de um nodo para o outro, foi reaproveitado o elemento e o ícone de *uml.Transition* do diagrama de máquina de estados. A classe *dia.ToolbaView* oferece ícones e ações padrões como zoom, selecionar, voltar, avançar e salvar, vistos na figura 40.

```
general.DirectedGraphDiagramToolbarView = dia.ToolbarView.extend({
    initialize: function(){
        dia.ToolbarView.prototype.initialize.apply(this, arguments);
        this.nodeCount = 0;

        this.icons = _.union(this.icons, ['node', 'transition']);
    },

    node: function() {
        var newNode = new general.Node({
            position: { x: 30, y: 20 },
            name: 'new' + (++this.nodeCount),
            age: "new"
        });
        this.paper.model.addCell(newNode)
    },

    transition: function() {
        this.paper.tool = uml.Transition;
    }

});
```

Figura 39: Código da view de barra de ferramentas

Figura 40: Barra de ferramentas padrão

8 – Criar um ícone para adição de nodos

Um ícone da barra de ferramentas é um elemento SVG com a descrição e um identificador, que deve seguir o padrão: "*nome_do_icone-icon".* Este padrão deve ser respeitado para que o ícone seja carregado pela barra de ferramentas. O SVG neste caso (figura 41) está sendo representado por uma String, mas poderia estar em um arquivo de template, eliminando a necessidade de aspas.

```
node: toolbarIcon('Add Node', 'node-icon',
    '<g transform="translate(10,10)">' +
        '<circle r="7" stroke="black" stroke-width="1.5" fill="white"/>' +
    '</g>'
)
```

Figura 41: Código do ícone

O código-fonte da figura 41 é acrescentado no arquivo *editor/templates/lib/icons.js*

Resultado

A figura 42 exibe o resultado do novo diagrama criado, o Editor de Grafo Orientado, na página principal de edição.

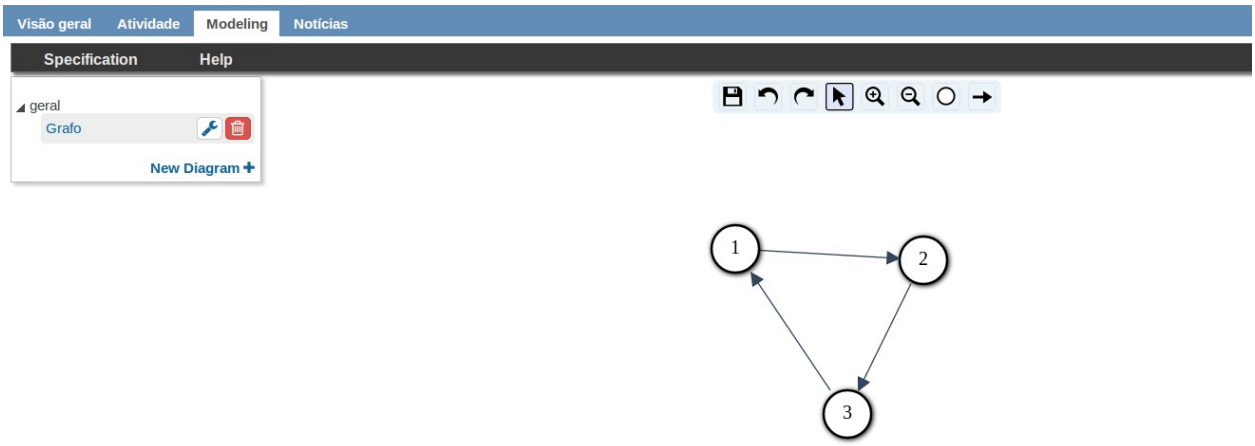

Figura 42: Resultado do novo diagrama

Referências bibliográficas

ALMEIDA, Giselle T. de; RAMOS, Bruno A.; F. NETO, Michelle M.. Projeto QUALI-EPT – Instituto Federal Fluminense (IFF). In: CBSOFT 2010, 4., 2010, Salvador. **Anais... .**Salvador: Ufbs, 2010. p. 14 – 20.

BACKBONEJS.ORG. **Backbone.js.** 2013. Disponível em: <http://backbonejs.org/>. Acesso em: 02 nov. 2013.

CASE-TOOLS.ORG**. Index CASE Tools**. Disponível em: <http://case-tools.org>. Acesso em: 04 ago. 2013.

CHINUBHAI, Aneesh. **Efficiency in Software Development Projects.** International Journal Of Software Engineering And Its Applications. Nadiad, p. 171-180. out. 2011. Disponível em: <http://www.sersc.org/journals/IJSEIA/vol5_no4_2011/13.pdf>. Acesso em: 04 dez. 2013.

CLIENT.IO. **JointJS**. 2013. Disponível em: <http://www.jointjs.com/> Acesso em: 02 nov. 2013.

Fall 2012. **Mobile Web Appplications Development with HTML5.** 2012. Disponível em: <http://aaltoWebapps.com/lecture3_2012_2.html> Acesso em: 03 dec. 2013.

FLANAGAN, David. **JavaScript:** The Definitive Guide, Sixth Edition, Sebastopol: O'reilly, 2011.

FLANAGAN, David; MATSUMOTO, Yukihiro. **The Ruby Programming Language.** Sebastopol: O'reilly, 2008.

Gliffy – **Online Diagram Software.** 2013. Disponível em: <http://www.gliffy.com> Acesso em: 10 nov. 2013.

JQuery. **Jquery.** Disponível em: <http://jquery.com/>. Acesso em: 01 maio 2014.

LARMAN, Graig. **Applying UML and Patterns Third Edition**: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition. 3. ed. Indianapolis: Addison Wesley Professional, 2004.

OMG. **Uml 2.4.1: Superstructure**. , 2011. Disponível em: <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF>. Acesso em: 02 dec. 2013.

PI, Shih-ming; SU, Shou-chian; WANG, Kai. **Promotion and Adoption of CASE Technology.** 1998. Disponível em: <http://disc-nt.cba.uh.edu/chin/digit98/second.pdf>. Acesso em: 10 jun. 2014.

PRESSMAN, Roger S.. **Software Engineering:** A PRACTITIONER'S APPROACH. Boston: Mcgraw-hill, 2001.

RAILS.ORG. **Ruby on Rails.** , 2013. Disponível em: <http://rubyonrails.org> Acesso em: 02 nov. 2013.

REDMINE. **Overview - Redmine.** 2013. Disponível em: <http://www.redmine.org> Acesso em: 02 nov. 2013.

REDMINE. **Plugin Tutorial.** 2013. Disponível em: <http://www.redmine.org/projects/redmine/wiki/Plugin_Tutorial> Acesso em: 02 nov. 2013.

RUBY.ORG. **Sobre o Ruby.** Disponível em: <https://www.ruby-lang.org/pt/about/>. Acesso em: 24 maio 2014.

SILVA, R. P. e. **UML 2 em Modelagem Orientada a Objetos**. Florianopolis: Visual Books, ´2007.

SILVA, R. P. e. **Suporte ao desenvolvimento e uso de frameworks e componentes.** Tese de Doutorado, Universidade Federal do Rio Grande do Sul: [s.n.], 2000.

Smart Draw – **Create Diagrams and Charts Automatically.** 2013. Disponível em: <http://www.smartdraw.com> Acesso em: 10 nov. 2013.

SURYN, Witold. **Software Quality Engineering:** A Practitioner's Approach. Hoboken, New Jersey: John Wiley & Sons, 2014.

TodoMVC. **Helping you select an MV* framework.** Disponível em: <http://todomvc.com/>. Acesso em: 19 jul. 2013>.

W3C. **A vocabulary and associated APIs for HTML and XHTML.** Disponível em: <http://www.w3.org/TR/html5/>. Acesso em: 19 jul. 2013>.

W3C. **HTML5 Introduction.** Disponível em: <http://www.w3schools.com/html/html5_intro.asp>. Acesso em: 01 maio 2014.