CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

Modelagem de sistemas utilizando a UML: Aplicando a engenharia reversa.

Rafael de Pieri Barbosa

Carlos Henrique Rodrigues Sarro

Americana/SP
2013

Faculdade de Tecnologia de Americana
Curso de Bacharelado em Análise de Sistemas e Tecnologia da Informação

Modelagem de sistemas utilizando a UML: Aplicando a engenharia reversa.

Rafael de Pieri Barbosa

rafaelpieribarbosa@gmail.com

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Americana como parte das exigências do curso de Análise de Sistemas e Tecnologia da Informação para a obtenção do título de Bacharel em Análise de Sistemas.

Orientador: Mestre Carlos Henrique Rodrigues Sarro

Americana/SP
2013

Barbosa, Rafael de Pieri
Modelagem de sistemas utilizando a UML: aplicando a engenharia reversa. / Rafael de Pieri Barbosa. – Americana: 2013.
84f.

Monografia (Graduação em Análise de Sistemas e Tecnologia da Informação) - Faculdade de Tecnologia de Americana - Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Ms. Carlos Henrique Rodrigues Sarro

1. Modelagem de sistemas 2. UML - linguagem de modelagem I. Sarro, Carlos Henrique Rodrigues II. Centro Estadual de Educação Tecnológica Paula Souza - Faculdade de Tecnologia de Americana.

CDU: 681.5.01
681.3.061

BANCA EXAMINADORA

Prof. Ms. Carlos Henrique Rodrigues Sarro

Prof. Ms. Leandro Halle Najm

Prof. Esp. Nivaldo Tadeu Marcusso

À

Minha família que sempre esteve ao meu lado nos momentos difíceis.

À

Uma amiga especial que não está mais entre a gente, mas sem ela não seria possível ter chegado até aqui. Em memória de Susana Daniel Scapinello.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por ter me guiado até aqui e por ter me dado forças para superar os momentos difíceis enfrentados.

Aos meus pais Antônio e Angela, que sempre lutaram para que eu pudesse estudar. Mesmo nos momentos difíceis sempre me incentivaram e em momento nenhum deixaram de acreditar em mim.

Ao meu irmão, que mesmo não tendo participação direta na minha formação, sempre esteve ao meu lado todos esses anos.

Aos meus grandes companheiros que fiz na faculdade, Samuel, Guilherme e Gabriel, como também a todos os outros alunos que estiveram ao meu lado todo esse tempo.

Ao meu orientador Carlos Henrique Rodrigues Sarro, que sem sombra de dúvida foi fundamental no desenvolvimento deste trabalho.

A todos os professores da FATEC que de alguma forma contribuíram para a minha formação.

Por fim, um agradecimento especial para Ana Carolina Scapinello e Susana Daniel Scapinello. Sem o apoio incondicional destas duas grandes pessoas, não seria possível ter chegado até aqui.

O que prevemos raramente ocorre; o que menos esperamos geralmente acontece.

(Benjamin Disraeli)

RESUMO

Atualmente a modelagem de sistemas tem sido tratada com grande descaso pela maioria das empresas de TI. Muitos profissionais da área não conhecem ou simplesmente não entendem o porquê de se utilizar esta técnica. Em sistemas complexos é imprescindível o uso deste recurso para que os participantes do projeto tenham uma visualização única de suas funcionalidades. Este trabalho tem como objetivo apresentar os motivos de se utilizar a modelagem em sistemas, bem como indicar uma ferramenta eficiente para este processo. Neste caso a UML foi escolhida por se tratar da linguagem padrão de modelagem adotada pela indústria de TI. Além de apresentar os recursos que esta linguagem oferece, será mostrada na prática a utilização dela através da aplicação de engenharia reversa.

Palavras-chave: UML, modelagem, engenharia reversa.

ABSTRACT

Currently the modeling of systems has been neglected by most IT companies. Many IT professionals don't know or just don't understand the reason of using this technique. In complex systems is indispensable the use of this resource, so that the project participants can have a single view of their functionalities. This work aims to present the reasons to use the modeling systems and indicate an efficient tool for this process. In this case the UML was chosen because it's the standard modeling language adopted by the IT industry. In addition to presenting the resource this type of language offers, it will be shown in practice the use of it, by applying reverse engineering.

Keywords: *UML, modeling, reverse engineering.*

LISTA DE ABREVIATURAS E SIGLAS

UML	*Unified Modeling Language*
TI	Tecnologia da Informação
OMT	*Object-Modeling Technique*
OOSE	*Object Oriented Software Engineering*
RFP	*Request for Proposal*
OMG	*Object Management Group*
RTF	*Revision Task Force*
BSD	*Berkeley Software Distribution*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*
JPEG	*Joint Photographic Experts Group*
PNG	*Portable Network Graphics*
JVM	*Java Virtual Machine*
SPL	*Sun Public License*
MPL	*Mozilla Public License*
IDE	*Integrated Development Environment*
JDK	*Java Development Kit*
JRE	*Java Runtime Environment*
BPMN	*Business Process Modeling Notation*
TDD	*Test Driven Development*

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A engenharia de *software* surgiu por volta de 1967 com o objetivo de combater a crise do *software*. Este termo foi utilizado para indicar as dificuldades existentes no desenvolvimento de sistemas computacionais, como a crescente demanda, a complexidade dos processos a serem atendidos e a ausência completa de qualquer técnica estabelecida para o desenvolvimento de *software* que funcionassem adequadamente (SCHACH, 2010).

O papel da engenharia de *software* é fornecer mecanismos que auxiliem os profissionais de TI a desenvolverem sistemas de qualidade, com baixo custo e dentro do prazo estimado. Além de fornecer técnicas para o gerenciamento do processo de desenvolvimento de um *software*, esta engenharia tem em seus fundamentos científicos a utilização de modelos abstratos que permitem especificar, projetar, implementar e manter sistemas computacionais, avaliando e garantindo a sua qualidade (MAGELA, 2004).

Segundo Guedes (2009, p.21) o processo de desenvolvimento de *software* pode ser dividido em seis fases:

1. Levantamento de requisitos;
2. Análise de requisitos;
3. Projeto;
4. Codificação;
5. Testes;
6. Implantação.

A modelagem de sistemas normalmente é aplicada na fase de análise de requisitos e principalmente na fase de projeto. Porém, ela também pode ser utilizada durante o processo de escrita do código ou até mesmo após a fase de implantação do projeto (GUEDES, 2009, p. 21).

Na etapa de análise de requisitos, segundo Guedes (2009, p. 23) a *"[...] modelagem auxilia a levantar questões que não foram concebidas durante as entrevistas iniciais"*. Guedes (2009, p. 23) ainda alerta dizendo que:

> Tais questões devem ser sanadas o quanto antes, para que o projeto do *software* não tenha que sofrer modificações quando seu desenvolvimento já estiver em andamento, o que pode causar significativos atrasos no desenvolvimento do *software*, sendo por vezes necessário remodelar por completo o projeto.

Com a modelagem já iniciada na fase de análise de requisitos, na etapa de projeto procura-se detalhar ao máximo as funcionalidades e restrições do *software* a ser desenvolvido. Sendo assim, o grande trabalho da modelagem acontece neste momento, pois os artefatos[1] gerados servirão de base para a implementação de todo o sistema, portanto, devem ser precisos, transmitindo detalhadamente o que deve ser feito (GUEDES, 2009, p.27).

Normalmente a modelagem é utilizada nas fases supracitadas (análise de requisitos e projeto), tendo como função auxiliar os profissionais de TI a construírem o sistema através de engenharia de produção. No entanto, pode-se aplicar a modelagem com o projeto em andamento ou após a conclusão do mesmo através de engenharia reversa. Neste processo, utiliza-se uma ferramenta de modelagem que com base no código-fonte do sistema, cria artefatos (diagramas) que detalham a estrutura do sistema e transmitem em linhas gerais o seu comportamento.

Porém, a modelagem de sistemas não é vista com bons olhos pela indústria de TI. Muitas empresas acreditam que o tempo gasto com a modelagem poderia ser utilizado para o desenvolvimento de *software*. O resultado de tal prática pode acarretar em retrabalho e desperdício de recursos durante o projeto que poderiam ser tranquilamente evitados com a aplicação da modelagem desde o início do processo. Com isso, o orçamento e prazo estabelecido normalmente não são cumpridos, causando insatisfação por parte do cliente e prejuízos financeiros para a empresa de *software* (LOBO, 2009, p. 6).

O objetivo deste trabalho é contrariar este pensamento, mostrando a importância da modelagem no processo de desenvolvimento de *software*, sobretudo, nos que são desenvolvidos no paradigma de orientação a objetos. Além disso, o trabalho tem o intuito de apresentar uma linguagem de modelagem eficiente que

[1] São subprodutos gerados no processo de desenvolvimento de software, por exemplo, classes e objetos gerados em sistemas que foram programados no paradigma de orientação a objetos.

atenda a esta necessidade. No caso a escolhida foi a UML por se tratar da linguagem-padrão adotada internacionalmente pela indústria de engenharia de software.

A UML é uma linguagem de modelagem, que tem como objetivo auxiliar os engenheiros de *software* a definirem as características do sistema, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado (GUEDES, 2009, p.19). Neste trabalho, além de apresenta-la, será aplicado a engenharia reversa em um sistema de gerenciamento de torneios de xadrez com o intuito de demonstrar na prática a utilização desta linguagem de modelagem.

2. CONTEXTO HISTÓRICO DA UML

Com o passar do tempo é inevitável o surgimento de novas tecnologias com o intuito de oferecer recursos que visam facilitar a vida do profissional de TI no desenvolvimento de sistemas cada vez mais complexos e que exigem um número maior de requisitos a serem preenchidos. Dentro deste contexto, a programação orientada a objetos foi desenvolvida para atender a esta nova necessidade de mercado, possibilitando que em apenas uma entidade pudesse ser representados os dados e as funções que operam sobre eles.

Segundo Silva Filho (2011, p. 2) *"a programação orientada a objetos é uma abordagem de programação que serve de elo entre os problemas existentes e as soluções computacionais apresentadas no campo da programação"*. Em outras palavras, a programação orientada a objetos possibilitou aos programadores quebrar a barreira conceitual que existia entre a adaptação das entidades reais às limitações existentes nas linguagens e técnicas de programações tradicionais.

Nas décadas de 1960 e 1970 surgiram as primeiras linguagens de programação orientada a objetos. A primeira normalmente reconhecida é a *Simula-67*, desenvolvida pelos noruegueses Ole-Johan Dahl e Kristen Nygaard, em 1967. Se tratando da origem desse paradigma de programação, não pode esquecer-se de Alan Kay, criador da linguagem *Smaltalk*. O cientista difundiu os termos programação orientada a objetos e computação pessoal, e teve grande contribuição para as ideias do PC moderno quando trabalhou na *Xerox PARC* (LARMAN, 2005, p. 42-43).

Porém o conceito de orientação a objetos ao longo daquele período permanecia informal, até que em 1982 o termo se tornou tema de um artigo escrito por Grady Booch (um dos fundadores da UML) intitulado *Object-Oriented Design.* Durante a década de 1980 muitos outros pioneiros da análise de programação orientada a objetos escreveram sobre o tema expondo suas ideias, bem como os fundadores da UML Jim Rumbaugh e Ivar Jacobson, dentre outros como Peter Coad, Steve Mellor, e Bertrand Meyer; destaque para este último que escreveu um dos livros mais influentes da época, *Object-Oriented Software Construction* publicado em 1988 (LARMAN, 2005, p. 43).

Com todo este apoio, a linguagem *Smaltalk* se tornou amplamente disponível na década de 1980, sendo precursora de outras linguagens como o *Objective* e *Eiffel*, além das linguagens C e C++, bastante utilizadas até os dias de hoje.

Diante de um novo paradigma de programação e de aplicações cada vez mais complexas, os profissionais de TI envolvidos em metodologia, começaram a testar novos métodos de análise e projeto, dando origem as primeiras linguagens de modelagem orientada a objetos, já nos anos 80. Na década seguinte, foram publicados três livros muito populares de metodologia. Em 1991, o livro *Object Oriented Modeling and Design* de Rumbaugh descrevia o método OMT, enquanto o livro *Object Oriented Design with Applications* de Grady Booch descrevia o método de Booch. Um ano mais tarde, em 1992, Jacobson publicou o livro *Object Oriented Software Engineering* que além de abordar a análise de programação orientada a objetos com a metodologia OOSE, continha casos de uso de requisitos (LARMAN, 2005, p. 43).

Estes três métodos surgiram com um claro destaque, onde cada um continuam pontos fracos e fortes. O método Booch focava as fases de projeto e construção do sistema; já o método OMT se encaixava melhor na fase de análise e sistemas de informações que despendia de um volume maior de dados; enquanto o método OOSE tinha como seu ponto forte o fornecimento de suporte nos casos de uso, facilitando a captura dos requisitos do sistema (BOOCH, JACOBSON, RUMBAUGH, 2006, p. XVI).

Em 1994, a UML começou a tomar forma quando Booch e Rumbaugh uniram esforços não apenas com o objetivo de criar uma notação comum, mas de alinhar seus dois métodos (Booch e OMT) em apenas um, dando origem ao primeiro rascunho público da UML conhecido como o Método Unificado (*Unified Method*). Logo em seguida, Ivar Jacobson, o criador do método OOSE, se uniu a Booch e Rumbaugh já na *Rational*[2], completando o grupo que veio a ser conhecido como “os três amigos”. A partir desse momento os esforços foram com o intuito de criar uma notação comum de diagramação ao invés de um método comum (LARMAN, 2005, p. 43).

[2] É uma empresa que foi fundada com o intuito de produzir ferramentas aplicando as melhores práticas da engenharia de software. Em 20 de fevereiro de 2003, ela foi adquirida pela IBM.

Em janeiro de 1997 nasce a UML 1.0, fruto dos esforços dos “três amigos” juntamente com um consórcio de UML com várias empresas de grande porte, que queriam uma definição completa da UML. Dentre as empresas que participaram da definição da UML 1.0, estavam a IBM, a *Oracle* e a *Microsoft*. O resultado foi uma linguagem de modelagem completa, podendo ser aplicada em várias situações e tipos de problemas. Mary Loomis teve grande contribuição ao convencer a OMG (*Object Management Group*), grupo responsável pelos padrões relacionados à orientação a objetos, a elaborar uma proposta (RFP) [3] de linguagem padrão de modelagem, onde a UML 1.0 foi oferecida em resposta a RFP.

Porém, somente em 14 de outubro de 1997 a OMG adotou a UML como sendo a linguagem padrão de modelagem para orientação a objetos, após um grupo de tarefas em semântica liderada por Cris Kobryn da *MCI Systemhouse* e administrado por Ed Eykhold da *Rational*, realizar uma revisão na linguagem com o propósito de formalizá-la e integrá-la a outros projetos de padronização.

Durante vários anos, a manutenção da UML foi realizada pela RTF (*Revision Task Force*) da OMG, onde foram produzidas as versões 1.3, 1.4 e 1.5. Um grupo de parceiros foi responsável pelas revisões e atualizações da UML de 2000 a 2003, originando a UML 2.0 que inclui mais recursos e alterações importantes que foram realizadas com base nos resultados obtidos nas versões anteriores. Essa versão foi adotada pela a OMG no início de 2005 (BOOCH, JACOBSON, RUMBAUGH, 2006, p. XVI).

A versão atual da UML é a 2.2.

[3] RFP é um convite enviado a um grupo de fornecedores para apresentarem propostas de venda de produtos ou serviços.

3. A IMPORTÂNCIA DA MODELAGEM DE SISTEMAS

Muitos profissionais de TI associam a modelagem apenas como um processo de documentação, porém, esta não é a sua função principal. Neste capítulo será apresentado o porquê da utilização da modelagem, bem como, as vantagens que a mesma pode trazer para o processo de desenvolvimento de *software*, com o objetivo de quebrar o estereótipo de que este recurso é perda de tempo.

3.1. Por que fazer a modelagem de sistemas?

As empresas de *software* bem-sucedidas são aquelas que desenvolvem soluções que atendam as necessidades dos seus clientes. Quando conseguem desenvolver um sistema dentro do prazo estimado e utilizando corretamente os recursos que dispõem, com certeza a empresa terá vantagem competitiva de mercado.

O foco principal das equipes de desenvolvimento não é a documentação do sistema, nem linhas de código organizadas ou a realização de reuniões sofisticadas. O foco principal é desenvolver um *software* que atenda as necessidades de negócio do cliente. Os demais detalhes do projeto pode-se dizer que é secundário (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 3).

O equívoco das empresas de software é tratar o “secundário” como “irrelevante” (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 4). Segundo Lobo (2009, p. 6) *“[...] muitas empresas ainda não utilizam a modelagem de software em seu processo de desenvolvimento por acharem que gastariam tempo [...] que poderia ser utilizado para o desenvolvimento do software”.* Esta linha de raciocínio aliada com uma modelagem que não atenda as reais necessidades do projeto, pode gerar retrabalho e desperdício de recursos que poderiam ser evitados caso a modelagem de software fosse aplicada no início do processo (LOBO, 2009, p. 6).

Para desenvolver um *software* de qualidade, é de suma importância realizar corretamente o processo de extração de requisitos, escolher as pessoas certas e as ferramentas que melhor se adequam as necessidades. Para que tudo ocorra dentro do previsto, principalmente no que diz respeito ao prazo do projeto e dos custos que

o envolvem, é fundamental um processo de desenvolvimento que seja seguro, dispondo de alternativas tanto para as necessidades de negócio como da tecnologia escolhida para o desenvolvimento do projeto (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 4).

Dentro deste contexto, a modelagem é um fator importante na implantação de um software de qualidade. Segundo Booch, Jacobson, Rumbaugh (2006, p. 3-4) construímos modelos para:

- Comunicar a estrutura e o comportamento desejados do sistema;
- Visualizar e controlar a arquitetura;
- Compreender melhor o sistema que estamos elaborando e;
- Gerenciar riscos.

Um modelo pode ser definido como uma simplificação da realidade. Eles nos fornecem uma cópia do projeto de um sistema. A abrangência dos modelos pode variar desde planos mais detalhados, como em planos mais gerais tendo uma visão completa do sistema. Bons modelos tendem a incluir componentes que tem grande relevância e omite os componentes de menor importância em determinado nível de abstração (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 6).

Os sistemas podem ser modelados sob diferentes visões, podendo se utilizar de modelos distintos, com o objetivo de abstrair da melhor maneira possível as informações do mundo real. Os modelos podem ser estruturais, onde o foco é a organização e especificação do sistema, ou podem ser comportamentais, tendo como objetivo mostrar a dinâmica do sistema (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 6).

Segundo Booch, Jacobson, Rumbaugh (2006, p. 6) com uma modelagem adequada, os modelos permitem alcançar quatro objetivos fundamentais em um projeto de software:

- Ajudam a visualizar o sistema como ele é ou como desejamos que seja;
- Permitem especificar a estrutura ou o comportamento de um sistema;
- Proporcionam um guia para a construção do sistema;

- Documentam as decisões tomadas.

A modelagem ajuda a delimitar o problema que estamos enfrentando, possibilitando dividir um problema difícil em partes menores, o que torna a solução mais fácil. Com isso, é possível ampliar o intelecto humano, permitindo pra quem usa a modelagem trabalhar com vários níveis de abstração (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 6-7).

O uso da modelagem só tende a trazer benefícios ao projeto em desenvolvimento, mas isso não quer dizer que o mesmo será utilizado. Normalmente quanto menos complexo for o sistema, maior a chance de não utilizá-la. De qualquer forma, com a modelagem, o projeto tende a ser mais rápido, facilitando o seu desenvolvimento. Além disso, o uso da modelagem minimiza os erros e a construção de itens errados, principalmente em projetos mais complexos. (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 6-7).

3.1.1. Modelagem orientada a objetos

Na visão tradicional, o foco são os algoritmos, onde o principal bloco de construção são as funções. Os desenvolvedores dentro desta perspectiva, tendem a se preocupar com o controle e decomposição de algoritmos maiores em outros menores, com o intuito de organizar o código. Com o crescimento dos sistemas e o surgimento de novos requisitos, a manutenção será complicada em sistemas construídos a partir do foco de algoritmos (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 11).

Na visão contemporânea, adota-se a perspectiva orientada a objetos, onde o principal bloco de construção é o objeto ou classe. Podemos dizer que um objeto é uma entidade estruturada que é abstraída do espaço do problema ou do espaço da solução, e uma classe é a especificação de um conjunto de objetos (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 11).

Para exemplificar a orientação a objetos, considere um sistema de cobrança com uma arquitetura relativamente simples composta por três componentes, uma interface com o usuário, uma camada intermediária e um banco de dados. Na interface, serão abstraídos objetos concretos, como botões, caixas de texto e

menus. A camada intermediária terá objetos que controlarão as operações de transação e toda a regra de negócio do sistema, além de visões de alto nível relacionadas às entidades do mundo real, como clientes, produtos e vendas. E no banco de dados, terá objetos concretos como as tabelas que serão responsáveis por armazenar os dados relacionados ao espaço do problema, como clientes, produtos e vendas (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 11).

Muitas das linguagens, sistemas operacionais e ferramentas contemporâneas têm sido desenvolvidas no paradigma de orientação a objetos, mostrando que a ideia de visão do mundo real em objetos não é uma tendência e sim uma realidade. Com isso novos desafios referentes à engenharia de *software* surgem a cada dia. As linguagens de modelagem tem a função de fornecer mecanismos para enfrentar estes desafios (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 11).

3.1.2. As linguagens de modelagem e a escolha da UML

Para desenvolver este trabalho, realizou-se um levantamento sobre as linguagens de modelagem disponíveis no mercado. Durante a pesquisa, foi possível notar que em sistemas desenvolvidos no paradigma estruturado, os fluxogramas são utilizados para fazer a modelagem, enquanto que em sistemas desenvolvidos no paradigma de orientação a objetos, a UML *(Unified Modeling Language)* é amplamente aplicada. Diante deste cenário, esta última foi escolhida por ser uma linguagem voltada para o paradigma de orientação a objetos, foco deste trabalho.

Além disso, a UML é adotada mundialmente pela engenharia de software, regulamentada pela OMG *(Object Management Group),* grupo responsável pelos padrões relacionados à orientação a objetos, e fruto de esforços realizados por inúmeros especialistas da área, sendo resultado da fusão de várias linguagens de modelagem conforme relatado no capítulo 2.

4. A UML E SEUS DIAGRAMAS

Após a explicação do porque se fazer a modelagem no processo de desenvolvimento de sistemas, é necessário conhecer uma linguagem que ofereça recursos para aplicá-la. Neste capítulo será apresentada a linguagem de modelagem UML, mundialmente utilizada pelos profissionais de TI. O que é? Onde e como pode ser aplicada? Quais recursos que ela oferece? Essas perguntas serão respondidas a seguir.

4.1. O que é UML?

A UML é uma linguagem de modelagem para visualização, especificação, construção e documentação de artefatos de sistemas de *software* complexos (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 13).

Essa linguagem tem como objetivo auxiliar os engenheiros de *software* a definirem as características do sistema, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado (GUEDES 2009, p.19).

É importante salientar que a UML não se restringe apenas à modelagem de sistemas computacionais, ela também pode ser utilizada na modelagem de fluxo de trabalho no sistema legal, na estrutura e comportamento de sistemas de saúde e em projetos de *hardware* (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 17).

A versão 2.0 da UML, que será o foco deste trabalho, é composta por 13 tipos de diagramas oficiais (a exceção é o diagrama de estruturas compostas) listados na Tabela 1 e devidamente classificados conforme indicado na Figura 1. Embora, este trabalho tenha sido dividido pelos tipos de diagramas, os autores da UML não tratam os diagramas como parte central da UML. Você pode utilizar elementos de um tipo de diagrama em outro, mesmo que o padrão UML indique que certos elementos são desenhados para determinados tipos de diagrama (FOWLER, 2004 p.33).

Tabela 1: Tipos de diagrama oficiais da UML **(**FOWLER, 2004, p. 33).

Diagrama	**Objetivo**	**Linhagem**
Classes	Classe, características e relacionamentos.	Na UML 1
Objetos	Exemplo de configurações de instâncias.	Extraoficialmente na UML 1
Componentes	Estrutura e conexão de componentes.	Na UML 1
Instalação ou implantação	Distribuição de artefatos nos nós.	Na UML 1
Pacotes	Estrutura hierárquica em tempo de compilação.	Extraoficialmente na UML 1
Estruturas compostas	Decomposição de uma classe em tempo de execução.	Novidade da UML 2
Casos de uso	Como os usuários interagem com um sistema.	Na UML 1
Gráficos de estados	Como os eventos alteram um objeto no decorrer de sua vida.	Na UML 1
Sequência	Interação entre objetos; ênfase na sequência.	Na UML 1
Atividades	Comportamento procedimental e paralelo.	Na UML 1
Visão geral da interação	Mistura de diagrama de sequencia e de atividades.	Novidade da UML 2
Comunicação	Interação entre objetos; ênfase nas ligações.	Diagrama de colaboração da UML 1
Tempo ou Temporização	Interação entre objetos; ênfase no sincronismo.	Novidade da UML 2

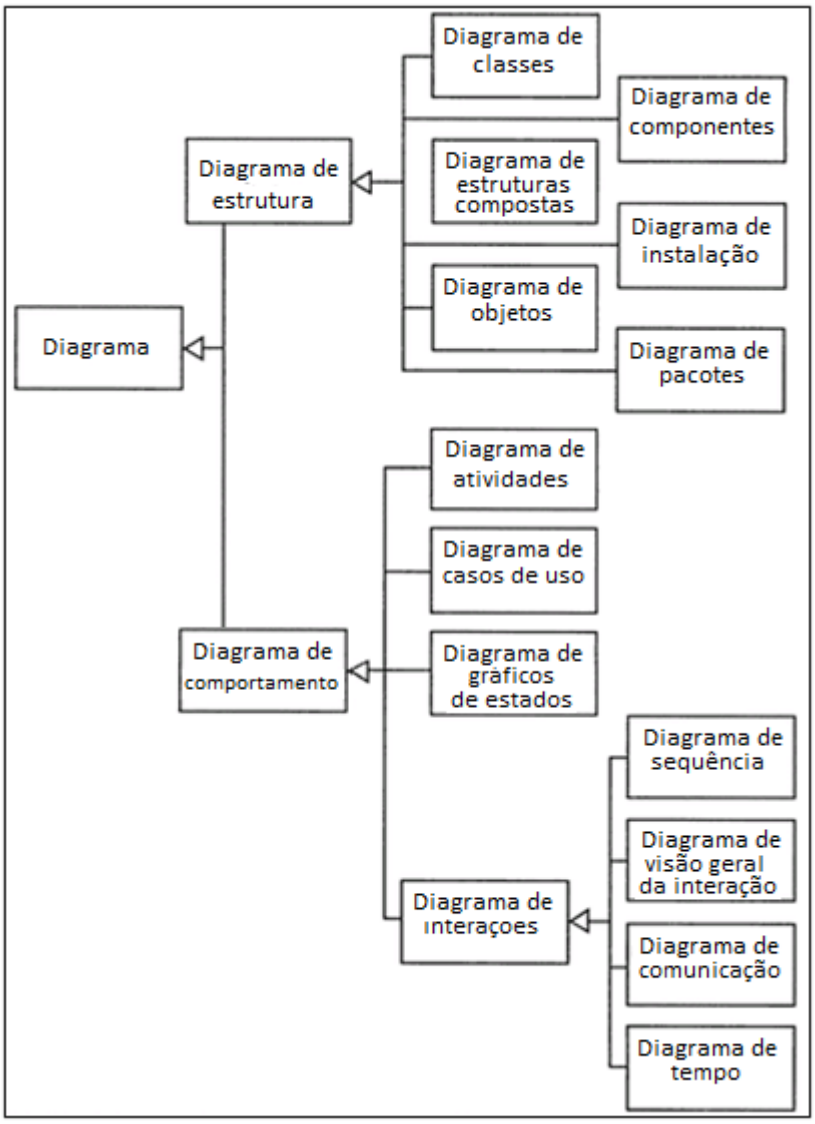


Figura 1: Classificação dos tipos de diagrama UML (FOWLER, 2004 p. 34).

4.1.1. Por que utilizar a UML?

A UML fornece ferramentas que otimizam a extração de requisitos, possibilitando mostrar de forma gráfica o problema do mundo real. As notações gráficas geradas são usadas principalmente para a comunicação entre os requisitos coletados na análise e a equipe de desenvolvimento. Dessa forma, o desenvolvimento do sistema partirá de um documento comum, diminuindo o tempo de implementação e otimizando a qualidade do *software* (SHALLOWAY, TROTT, 2002, p. 55).

Além disso, a UML provê se o entendimento da equipe em relação ao sistema é o mesmo. Como os sistemas normalmente são complexos e possuem diferentes

tipos de informação a serem transmitidas, ela fornece diversos diagramas especializados para tipos distintos de informação.

Por fim, a UML não é somente uma ferramenta eficaz pra descrever projetos orientados a objetos, ela também força o profissional de TI a pensar por meio de seu próprio enfoque (SHALLOWAY, TROTT, 2002, p. 55).

4.1.2. Maneiras de se utilizar a UML

A UML é utilizada de três formas distintas no desenvolvimento de *software*: esboço, projeto e linguagem de programação. Na UML como esboço os desenvolvedores utilizam a UML para transmitir alguns aspectos de um sistema. Dentro desta forma de aplicação, pode-se utilizar esboços no desenvolvimento e na engenharia reversa. No desenvolvimento, o diagrama UML é desenhado antes de se escrever o código, enquanto na engenharia reversa o diagrama UML é construído a partir de um código já existente, com o intuito de ajudar seu entendimento (FOWLER, 2004, p. 25-26).

Diferentemente da UML como esboço, a UML como projeto tem o foco detalhar a complexidade do sistema. Assim como no modelo de aplicação anterior, na UML como projeto pode-se aplicar esta abordagem no desenvolvimento e na engenharia reversa. No desenvolvimento, o objetivo é construir um projeto detalhado para um programador codificar. Esse projeto deve ser completo ao ponto de que todas as decisões estejam expostas, permitindo ao programador segui-lo como uma atividade simples e direta. Na engenharia reversa, os projetos devem transmitir de forma detalhada o código em documentos. Os projetos podem mostrar de forma gráfica cada detalhe de uma classe, facilitando o entendimento dos desenvolvedores (FOWLER, 2004, p. 26). Este tipo de abordagem da UML será tema de estudo no capítulo 5, onde será aplicada a engenharia reversa em um sistema de gerenciamento de torneios de xadrez.

Por fim, a UML pode ser usada como linguagem de programação. Neste contexto, os desenvolvedores constroem os diagramas UML que são compilados diretamente para código executável, fazendo com que a UML se torne o código-fonte (FOWLER, p. 27).

4.2. Diagramas Estruturais

Os diagramas estruturais são utilizados para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema. A UML 2.0 oferece seis diferentes tipos de diagramas estruturais, são eles os diagramas de classes, objetos, componentes, instalação, pacotes e de estruturas compostas. A seguir serão explicados os conceitos básicos e as funcionalidades de cada diagrama estrutural.

4.2.1. Diagrama de classes

O diagrama de classes é certamente o mais utilizado e sem dúvida um dos mais importantes diagramas da UML, servindo como base para a maioria dos demais diagramas (GUEDES, 2009, p. 33).

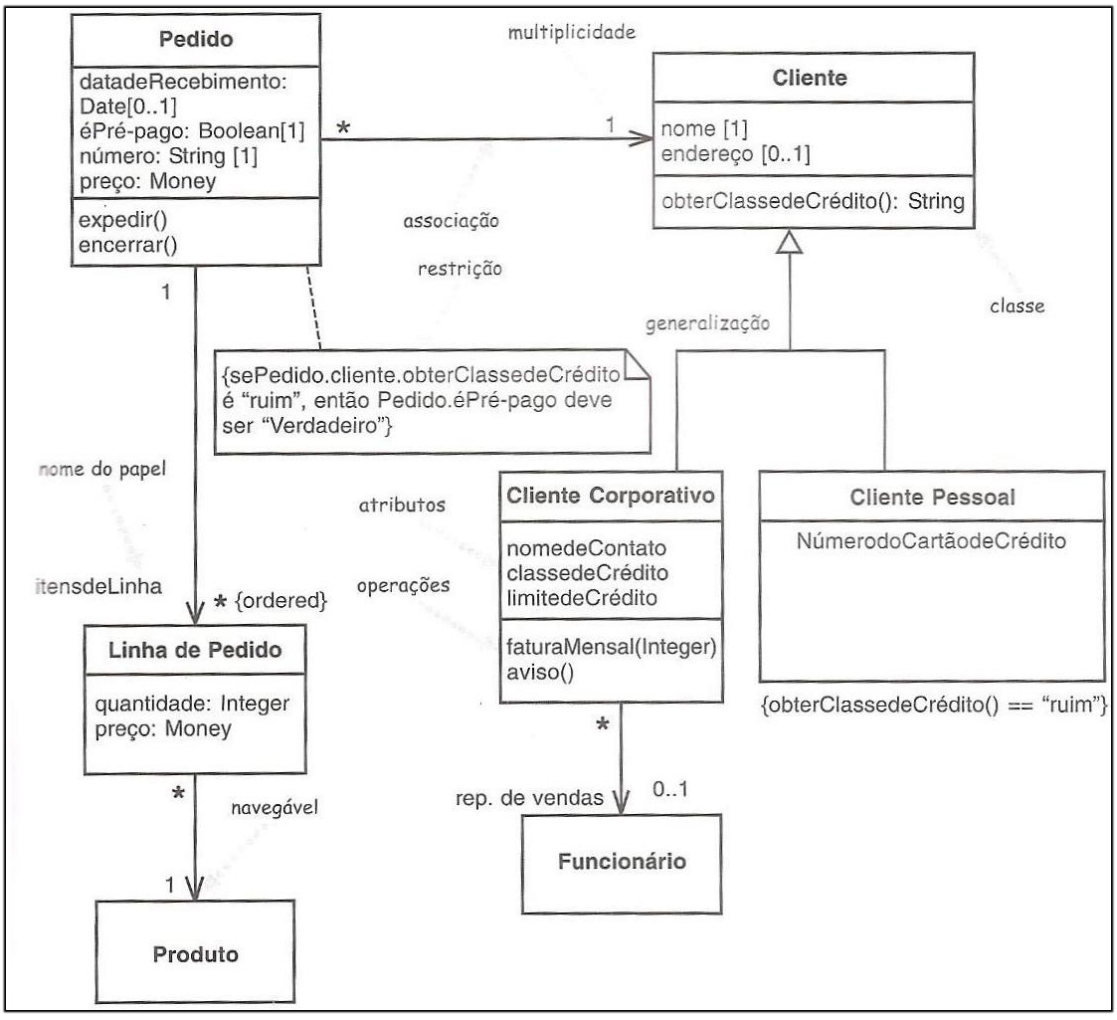


Figura 2: Exemplo de Diagrama de Classes: Estrutura de um Pedido (FOWLER, 2004 p. 34).

Um diagrama de classes descreve os objetos presentes no sistema e os tipos de relacionamentos estáticos que existem entre eles. Além disso, os diagramas de classes mostram as propriedades e as operações de uma classe, bem como as restrições aplicadas à maneira como os objetos estão conectados (FOWLER, 2004, p. 52).

É importante salientar que os diagramas de classes não são apenas importantes para visualizar, especificar e documentar modelos estruturais, mas também para construção de sistemas executáveis através da engenharia de produção e reversa (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 107).

A Figura 2 mostra um modelo de diagrama de classes simples, descrevendo a estrutura de um pedido. As caixas do diagrama de classes são divididas em três partes, sendo a primeira em negrito o nome da classe, a segunda os atributos e a terceira as operações. A Figura 2 também mostra dois tipos de relacionamentos entre classes: associações e generalizações. (FOWLER, 2004, p.52).

4.2.2. Diagrama de objetos

O diagrama de objetos tem como objetivo fazer a modelagem de instâncias de itens que estão em um diagrama de classes. Este tipo de diagrama mostra um conjunto de objetos e seus relacionamentos em um determinado ponto no tempo (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 188).

Os diagramas de objetos são usados para fazer a modelagem do processo de um sistema. Essa prática envolve a modelagem de um retrato do sistema em um período de tempo e a representação de um conjunto de objetos, com seus estados e relacionamentos (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 188).

A Figura 3 mostra um conjunto de classes e a Figura 4 mostra um conjunto de objetos associados. Os elementos da Figura 4 são instâncias das classes da Figura 3 porque os nomes estão sublinhados. Neste tipo de diagrama cada nome da classe tem a forma *nome de instância: nome de classe,* porém as duas partes do nome são opcionais, portanto, tanto *John: Pessoa* e *umaPessoa* são nomes considerados válidos. Quando utilizar apenas o nome da classe, é necessário incluir os dois

pontos. Além disso, você pode mostrar valores para atributos e vínculos, como mostra a Figura 3 (FOWLER, 2004, p. 94).

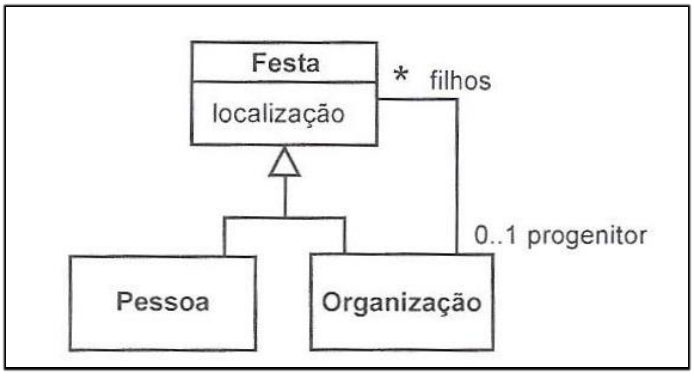


Figura 3: Diagrama de Classes da Estrutura de Composição Festa (FOWLER, 2004 p. 95).

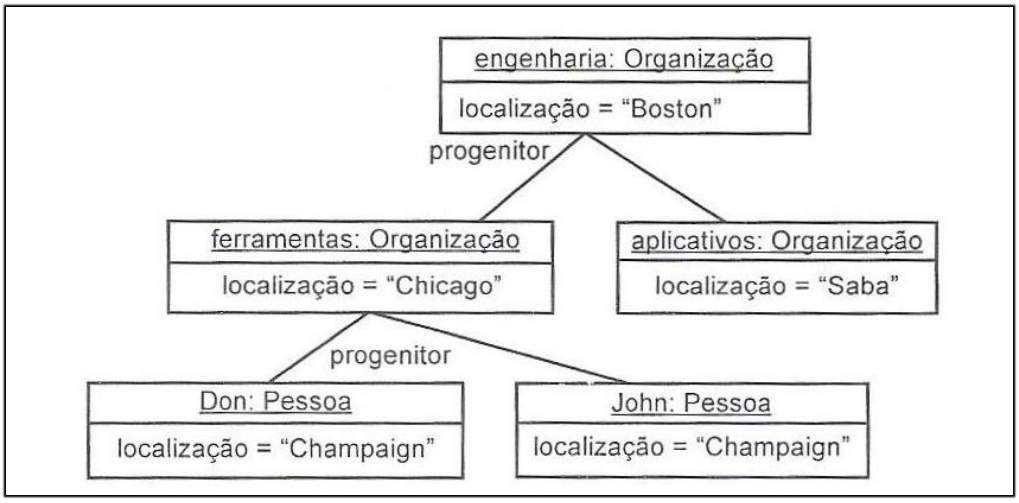


Figura 4: Diagrama de objetos mostrando exemplos de instâncias de Festa (FOWLER, 2004 p. 95).

4.2.3. Diagrama de componentes

O diagrama de componentes tem como foco evidenciar as partes lógicas de um sistema. Este tipo de diagrama representa os componentes do sistema a serem implementados em termos de módulos de código-fonte, bibliotecas, formulários e módulos executáveis, além de descrever como eles estarão estruturados e irão se comunicar para que o sistema funcione corretamente (GUEDES, 2009, p. 40).

Segundo Booch, Jacobson, Rumbaugh (2006, p. 195) *"um componente é a parte lógica e substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interfaces".* Em outras palavras, componentes construídos de maneira adequada definem abstrações com interfaces bem-definidas, tornando

possível substituir componentes mais antigos por outros componentes mais novos desde que sejam compatíveis.

As interfaces têm como função fazer a ligação entre os modelos lógico e de projeto, ou seja, a especificação de uma interface para uma classe pode ser usada em um modelo lógico, e ser transferida para algum componente do projeto (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 195).

Veja a aplicação dos diagramas de componentes na Figura 5.

> Neste exemplo, uma caixa registradora pode se conectar com um componente servidor de vendas, usando uma interface de mensagens de vendas. Como a rede não é confiável, um componente fila de mensagem é introduzido para que a caixa possa se comunicar com o servidor, quando a rede estiver ativa, e se comunicar com uma fila, quando a rede estiver desativada; a fila se comunicará então com o servidor, quando a rede se tornar disponível. Como resultado, a fila de mensagens fornece a interface de mensagens de vendas para se comunicar com a caixa e exige essa interface para se comunicar com o servidor. O servidor é dividido em dois componentes principais. O processador de transações implementa a interface de mensagens de vendas e o driver de contabilidade se comunica com o sistema de contabilidade (FOWLER, 2004, p. 134).

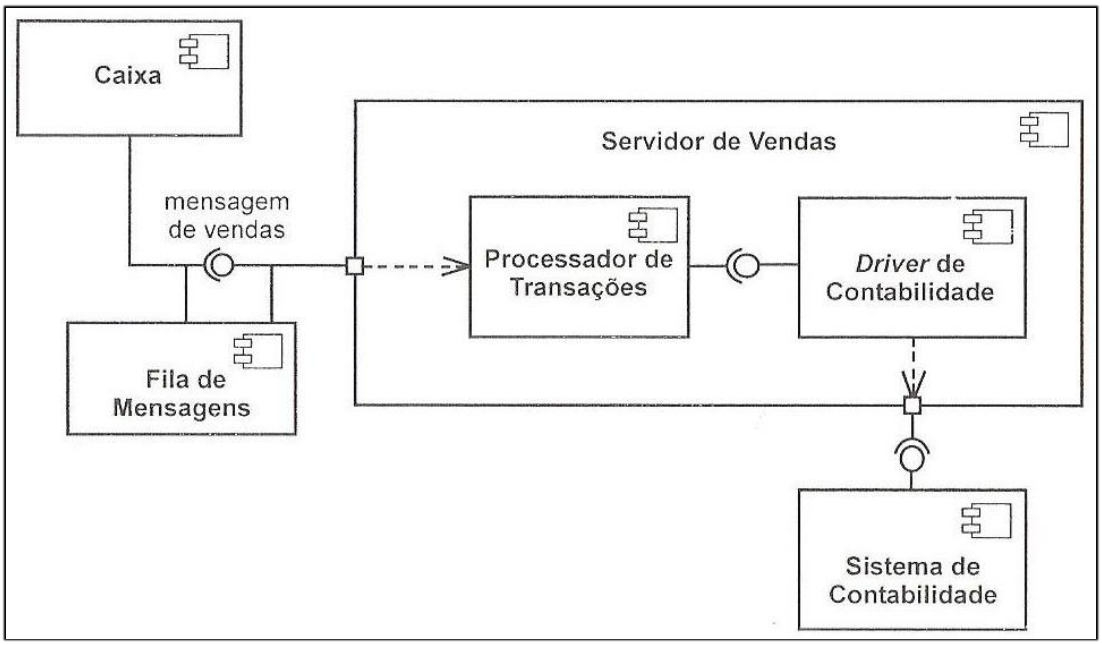


Figura 5: Exemplo de diagrama de componentes (FOWLER, 2004 p. 135).

4.2.4. Diagrama de instalação ou implantação

O diagrama de instalação ou implantação como também é conhecido, é utilizado para fazer a modelagem da visão estática da implantação de um sistema. Normalmente, a modelagem envolve a topologia do *hardware* em que o sistema é executado (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 411). Características físicas como servidores, estações, topologias e protocolos de comunicação, ou seja, toda a estrutura física sobre o qual o sistema terá que rodar, é descrito no diagrama de instalação (GUEDES, 2009, p. 41).

Os diagramas de instalação são importantes para a visualização, especificação e documentação de sistemas embutidos, cliente/servidor e distribuídos, bem como para o gerenciamento de sistemas executáveis por meio de engenharia de produção e reversa. (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 411).

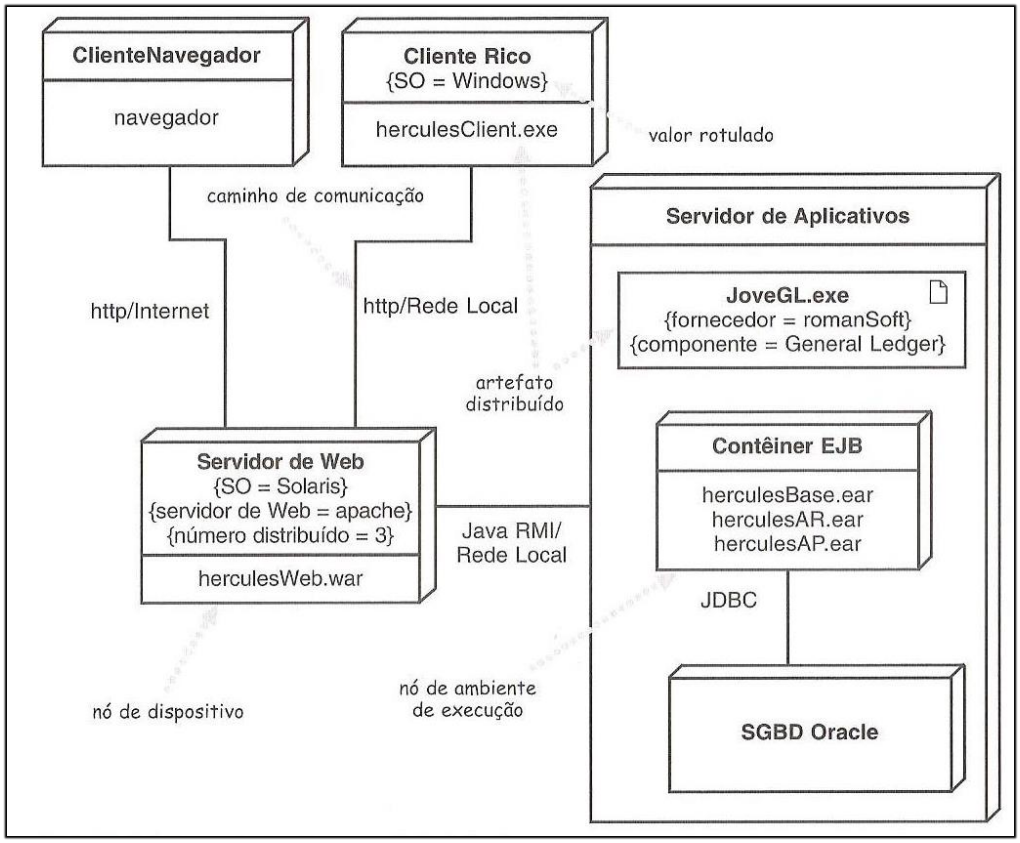


Figura 6: Exemplo de diagrama de instalação (FOWLER, 2004 p. 103).

A Figura 6 é um exemplo de um diagrama de distribuição. Os itens do diagrama são nós conectados por caminhos de comunicação. Cada nó pode conter algum *software*. Um dispositivo é *hardware*, sendo que ele pode ser um computador ou uma simples peça de *hardware* conectado a um sistema. Um ambiente de execução é um *software* que contém ele mesmo ou outro *software*, como por exemplo, um sistema operacional (FOWLER, 2004, p. 103).

4.2.5. Diagrama de pacotes

O diagrama de pacotes tem como objetivo representar subsistemas ou submódulos englobados por um sistema determinando as partes que o compõem (GUEDES, 2009, p. 35). Em outras palavras, o diagrama de pacotes é utilizado para agrupar os elementos da modelagem em grupos maiores permitindo que os mesmos sejam manipulados como grupos. O uso de pacotes é aplicado comumente para agrupamento de classes, porém este recurso pode ser aplicado pra qualquer outro elemento da UML (FOWLER, 2004, p. 96).

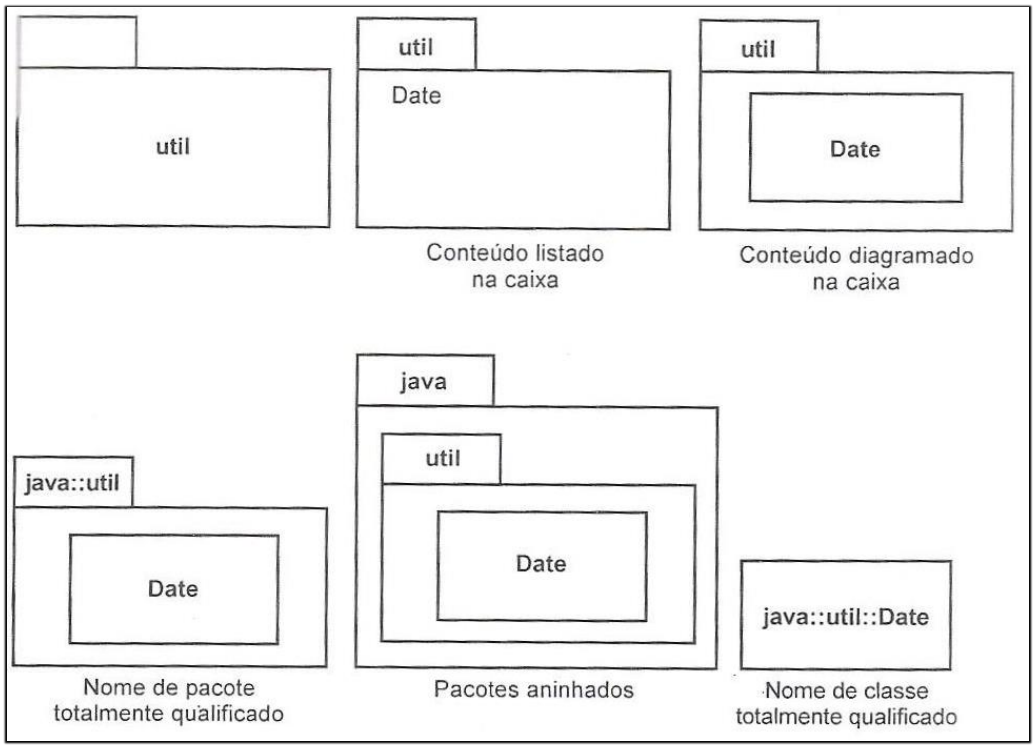


Figura 7: Maneiras de representar pacotes em diagramas (FOWLER, 2004 p. 97).

Os pacotes podem ser utilizados de diversas formas nos diagramas conforme pode ser visto na Figura 7. Pode-se mostrar somente o nome do pacote ou mostrar também seu conteúdo. A exibição deste último com ícones de classe possibilita a

visualização de todos os detalhes de uma classe. Outro recurso é a listagem dos nomes das classes quando o objetivo é indica-los que fazem parte do pacote. Os nomes dos pacotes podem ser qualificados ou simplesmente receber nomes normais (FOWLER, 2004, p. 96).

4.2.6. Diagrama de estruturas compostas

O diagrama de estruturas compostas tem como utilidade descrever a estrutura interna de um classificador, por exemplo, uma classe ou componente, detalhando as partes internas que o compõem, como estas se comunicam e colaboram entre si. Os diagramas de estruturas compostas também podem ser utilizados para descreverem uma colaboração em que um conjunto de instâncias cooperam entre si para executar uma tarefa (GUEDES, 2009, p.42). Vale ressaltar que o diagrama de estruturas compostas não corresponde a um diagrama oficial da UML 2.0 (FOWLER, p. 136).

Para exemplificar a aplicação deste diagrama, considere a noção de um leilão. Normalmente neste tipo de ambiente, devemos ter um vendedor, alguns compradores, lotes de mercadorias e ofertas para a venda. A Figura 8 demonstra este cenário através de um diagrama de classes.

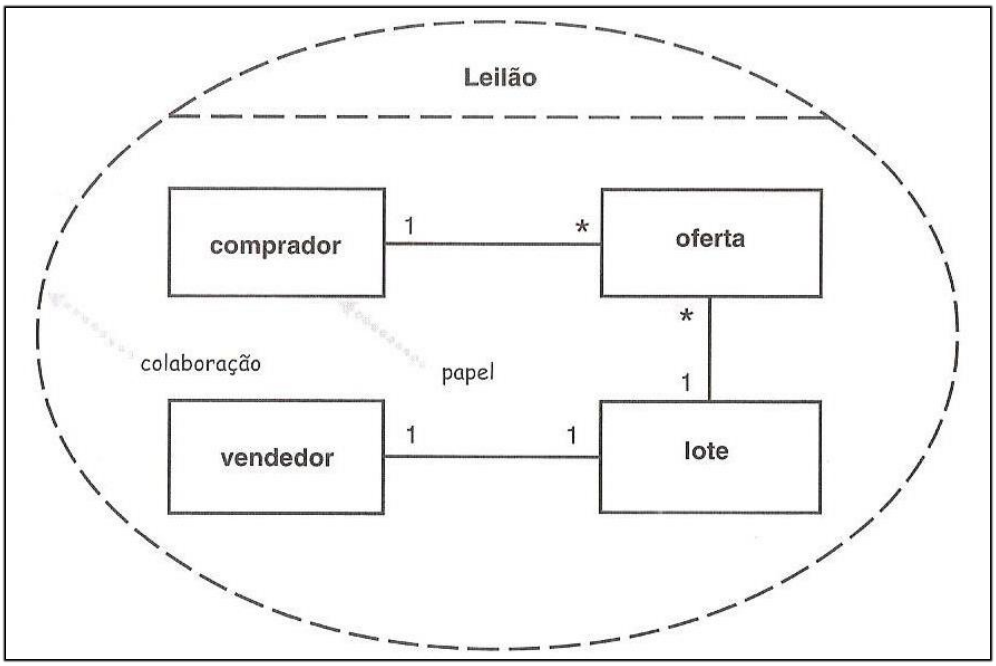


Figura 8: Empregando uma colaboração em um diagrama de classes (FOWLER, 2004 p. 136).

Note que o diagrama em questão não é um diagrama de classes normal. Ao redor dos diagramas há uma elipse tracejada, que representa a colaboração leilão. Dessa forma, as classes que estão na colaboração, não são classes propriamente ditas, mas sim papéis que serão desempenhados quando a colaboração for aplicada (FOWLER, 2004, p. 136).

4.3. Diagramas Comportamentais

Diagramas comportamentais são utilizados para evidenciar alterações que ocorrem no sistema em tempo de execução. A UML 2.0 oferece sete diferentes tipos de diagramas comportamentais, são eles os diagramas de casos de uso, atividades, gráficos de estados, sequência, visão geral da interação, comunicação e de tempo. Estes quatro últimos fazem parte de um subgrupo denominado de diagramas de interação. Basicamente os três primeiros diagramas descrevem as alterações em alto nível, enquanto os diagramas de interação representam interações que ocorrem internamente no sistema. A seguir serão explicados os conceitos básicos e as funcionalidades de cada tipo de diagrama comportamental.

4.3.1. Diagrama de casos de uso

O diagrama de casos de uso é utilizado para a modelagem do comportamento de um sistema, de um subsistema, ou de uma classe (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 241). Este tipo de diagrama procura identificar os atores (normalmente usuários, outros sistemas ou algum *hardware* especial) que utilizarão o sistema, como também os serviços, em outras palavras, o diagrama de casos de uso procura evidenciar as funcionalidades que o sistema disponibilizará aos atores (GUEDES, 2009, p. 31).

Os diagramas de casos de uso fazem com que sistemas, subsistemas e classes se tornem mais acessíveis e compreensíveis, pelo fato de apresentarem uma visão externa sobre como esses elementos podem se comportar dentro do contexto. Além disso, os diagramas de casos de uso também são importantes para realização de testes em sistemas executáveis tanto por meio de engenharia de produção como para compreendê-los por meio de engenharia reversa (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 241).

A Figura 9 mostra um exemplo da aplicação deste diagrama. Neste sistema, o principal processo é a locação de cópias, onde o sócio se identifica e informa as cópias que deseja fazer a locação. Caso o registro do sócio existir e ele não tiver cópias em atraso, a locação será autorizada. Ainda neste exemplo, é evidenciado o processo de emissão dos filmes mais locados, para que a empresa possa ter uma ideia da preferência dos seus clientes (GUEDES, 2009, p. 82).

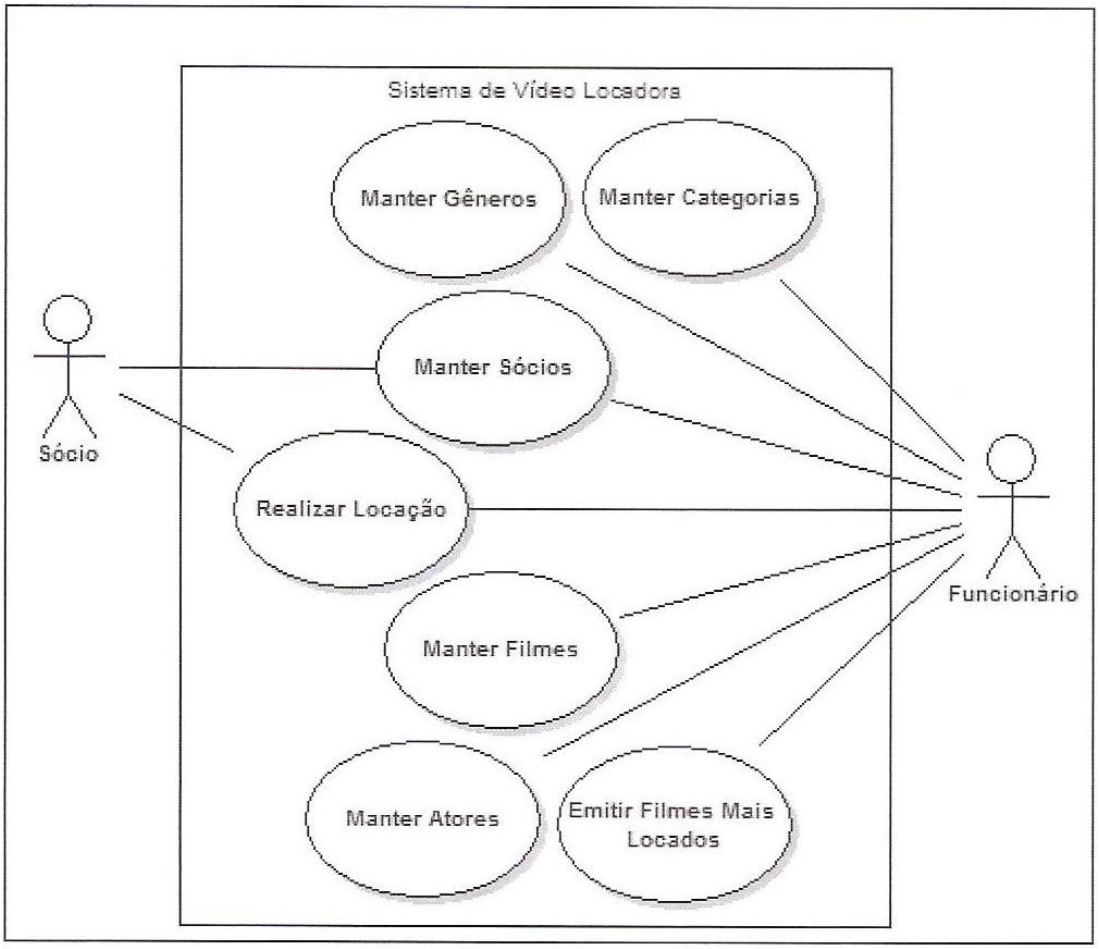


Figura 9: Diagrama de casos de uso – Sistema de Vídeo Locadora (GUEDES, 2009 p. 82).

4.3.2. Diagrama de atividades

Pode-se dizer que um diagrama de atividades é em sua essência um gráfico de fluxo, cuja finalidade é mostrar o fluxo de controle de uma atividade para outra. A diferença está no fato de um diagrama de atividades mostrar a concorrência e as ramificações de controle, enquanto um gráfico de fluxo tradicional não (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 268).

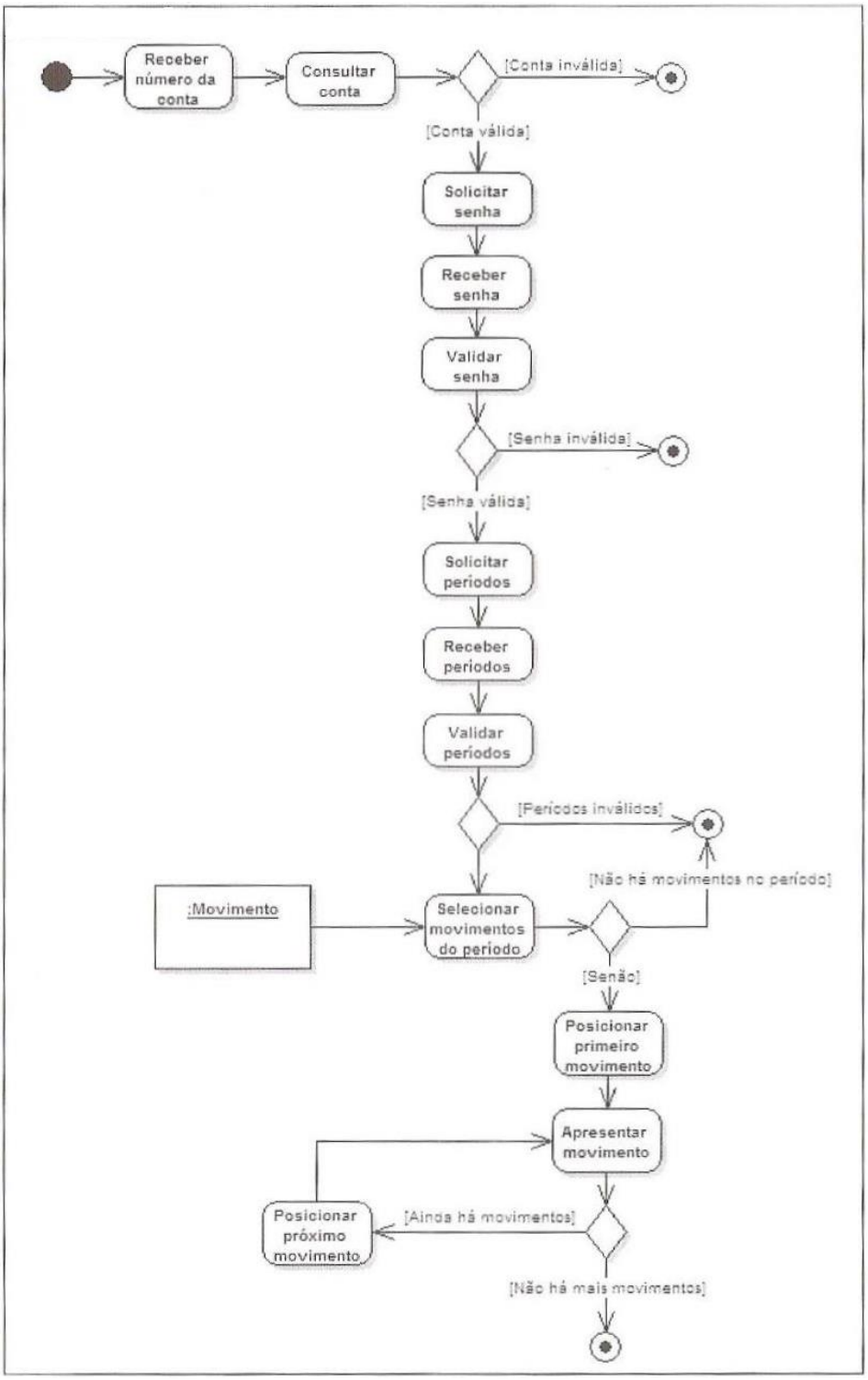


Figura 10: Diagrama de atividades – Processo de Emissão de Extrato (GUEDES, 2009 p. 302).

Um diagrama de atividades comumente descreve os passos a serem percorridos para a conclusão de uma determinada atividade, podendo a mesma ser representada por um método, algoritmo ou até mesmo por um processo complexo. (GUEDES, 2009, p. 38).

A Figura 10 traz um exemplo de diagrama de atividades que tem como objetivo representar o processo de emissão de extrato. Ao cliente informar o número da conta, é realizado uma consulta com a finalidade de verificar se a conta é válida. Se a mesma for, é solicitado a senha ao usuário e novamente é feita uma validação dos dados do cliente, neste caso a senha. Após a validação da senha, é requerido o período de consulta para o usuário. O período informado também é avaliado e caso seja valido retorna o extrato para o cliente (GUEDES, 2009, p. 302).

4.3.3. Diagrama de gráficos de estados

O diagrama de gráficos de estados tem como função mostrar uma máquina de estados, ou seja, demonstrar o comportamento de um elemento por meio de um conjunto finito de transições de estado (GUEDES, 2009, p. 37).

Este tipo de diagrama é empregado para a modelagem dos aspectos dinâmicos de um sistema. Isso envolve a modelagem do comportamento de objetivos reativos[4] que tem um claro tempo de vida cujo comportamento atual é afetado pelo seu passado. Os diagramas de gráficos de estados podem ser usados juntamente com classes, casos de uso ou em sistemas inteiros com o intuito de visualizar, especificar, construir e documentar a dinâmica de um objeto individual (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 341).

É importante frisar que os diagramas de atividades são um caso especial de diagramas de gráficos de estados. Segundo Booch, Jacobson, Rumbaugh (2006, p.341) isso ocorre quando *"[...] todos ou a maioria dos estados são estados de atividades e todas ou a maioria das transições são ativadas pela conclusão de atividades no estado de origem".*

Como exemplo de uso do diagrama de gráfico de estados, considere a Figura 11 que demonstra o processo de realizar depósitos. O cliente ao fornecer o número da conta que receberá o depósito, o método *Consulta_Conta* é disparado. Após a consulta, o evento que representa a informação do valor a ser depositado gera o estado *Depositando Valor,* onde o método *Depositar_Valor* é executado. Quando esse estado é concluído, é gerado uma transição que leva a um estado de

[4] Objetos reativos são aqueles cujo comportamento é mais bem caracterizado por sua resposta a eventos ativados externamente ao seu contexto. (BOOCH, JACOBSON, RUMBAUGH, 2006, p. 341)

submáquina que representa o processo de *Registrar Movimento*. Por fim, quando o registro do movimento é finalizado, o processo de realizar depósitos chega ao fim (GUEDES, 2009, p. 37).

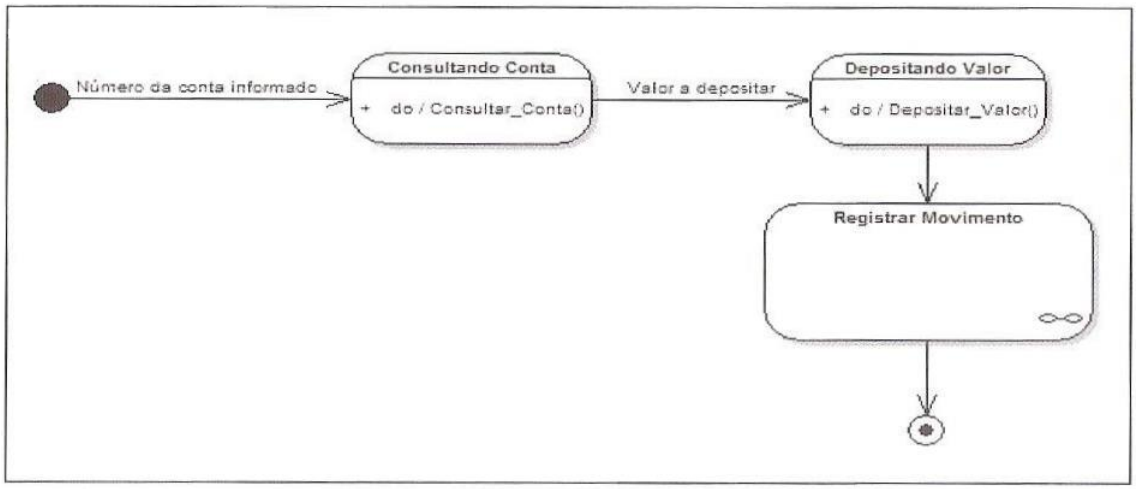


Figura 11: Diagrama de gráficos de estado – Processo de Realizar Depósito (GUEDES, 2009 p. 272).

4.3.4. Diagrama de sequência

O diagrama de sequências tem como objetivo descrever como grupos de objetos cooperam em algum comportamento. *"Normalmente, um diagrama de sequencias captura o comportamento de um único cenário"* (FOWLER, 2004, p. 67).

Segundo Booch, Jacobson, Rumbaugh (2006, p. 253) *"Um diagrama de sequências é um diagrama de interação que dá ênfase â ordenação temporal de mensagens"*. Em outras palavras, um diagrama de sequência procura identificar o evento de geração do processo modelado, como também o ator responsável por esse evento, e define como o processo deve prosseguir e ser concluído por meio da chamada de métodos invocados por mensagens enviadas entre os objetos (GUEDES, 2009, p. 35).

A Figura 12 ilustra um exemplo de uso do diagrama de sequências. Neste cenário, um pedido terá um comando executado para calcular seu preço. Pra realizar esse cálculo é preciso examinar os itens de linha do pedido e determinar seus preços, que são baseados nas regras de composição de preços dos produtos da linha. Depois de feito isso para todos os itens de linha, é preciso calcular um desconto global, que é baseado em regras vinculadas ao cliente que realizou o pedido (FOWLER, 2004, p. 67).

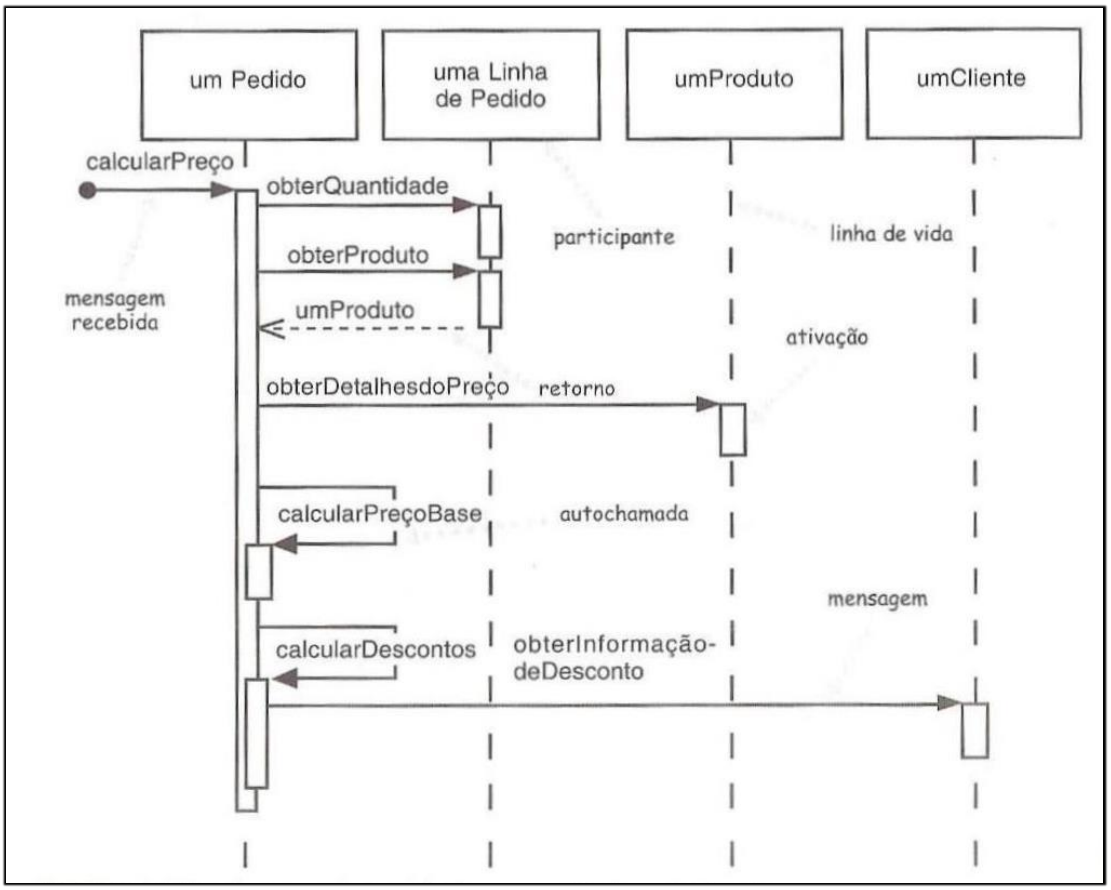


Figura 12: Exemplo de diagrama de sequencia (FOWLER, 2004 p. 272).

4.3.5. Diagrama de visão geral da interação

O diagrama de visão geral da interação é uma variação do diagrama de atividades que tem como objetivo fornecer uma visão geral do controle de fluxo (GUEDES, 2009, p. 321). Este tipo de diagrama traz uma mistura entre o diagrama de atividades e o diagrama de sequências, podendo considerar os diagramas de visão geral da interação *"[...] como diagramas de atividades nos quais as atividades são substituídas por pequenos diagramas de sequência ou como um diagrama de sequência fragmentado, com a notação de diagrama de atividades é usado para mostrar o fluxo de controle"* (FOWLER, 2004, p. 139).

A Figura 13 traz um exemplo simples da utilização do diagrama de visão geral da interação. Neste exemplo, é mostrado a produção e formatação de um relatório de resumo de pedidos. Se o cliente for externo, as informações serão obtidas em XML, e se for interno, serão obtidas em um banco de dados. As duas alternativas são apresentadas por pequenos diagramas de sequência. Quando os dados são obtidos, formatamos o relatório; neste caso, não mostramos os detalhes do

diagrama de sequência, mas somente fazemos referência a ele com um quadro de interação de referência (FOWLER, 2004, p. 139).

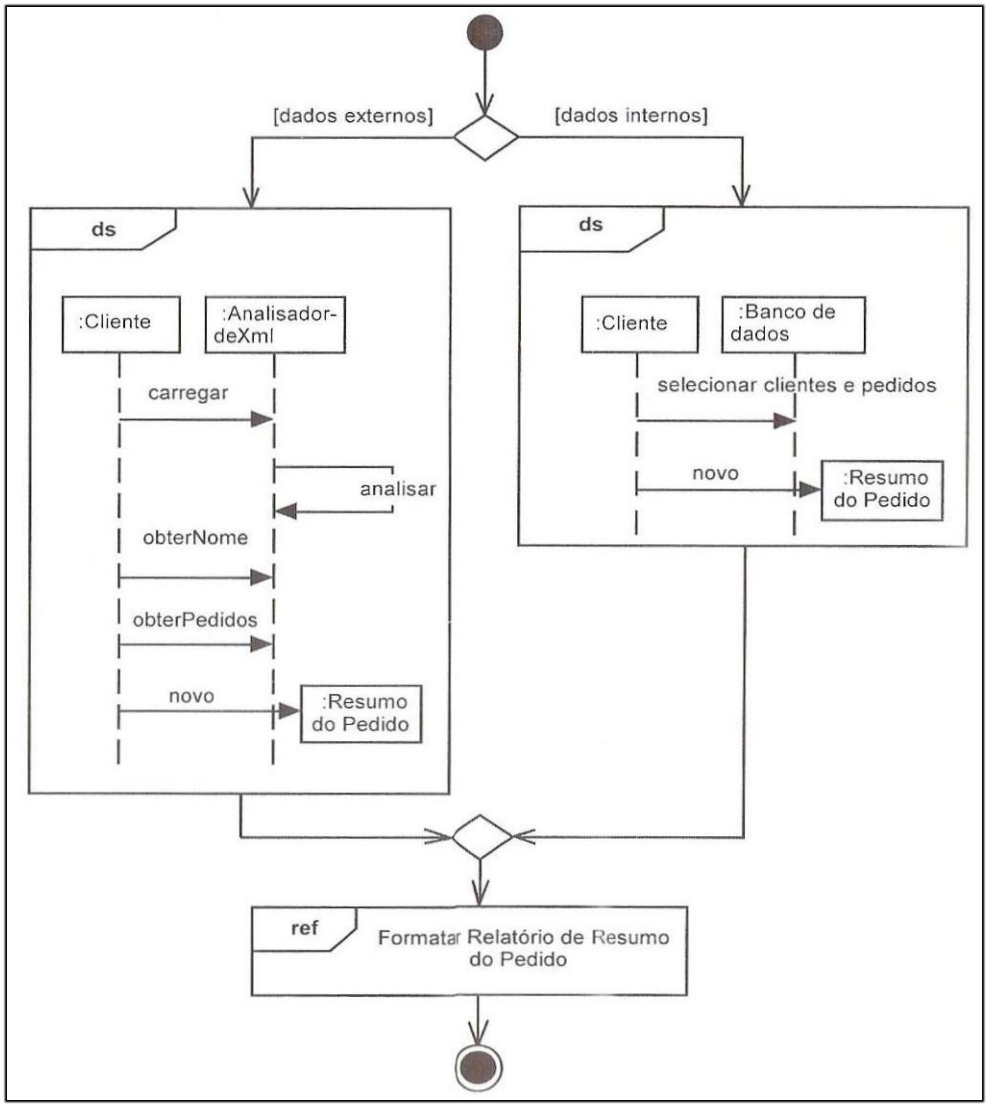


Figura 13: Exemplo de diagrama de visão geral da interação (FOWLER, 2004 p. 140).

4.3.6. Diagrama de comunicação

O diagrama de comunicação é um tipo de diagrama de interação que evidencia os vínculos de dados entre os vários participantes na interação. Ao invés de modelar cada participante como uma linha de vida e mostrar a sequência de mensagens por meio da direção vertical, como pode ser feito utilizando os diagramas de sequência, o diagrama de comunicação permite posicionar livremente os participantes, desenhar vínculos para mostrar como eles se conectam e usam numeração para mostrar a sequência de mensagens (FOWLER, 2004, p. 129).

Como exemplo, considere um processo de emissão de saldo. A Figura 14 mostra a interação entre o ator Cliente e os objetos das classes *Interface_Banco, Controlador_Banco* e *Conta_Comum.* O ator Cliente insere o cartão da conta, então a interface manda uma mensagem para a controladora que por meio do vínculo com o objeto da classe *Conta_Comum* dispara o método *Consultar_Conta*. Em resposta este método retornará se a conta é válida ou não. Caso a mesma seja válida, o ator Cliente informará a senha, a interface repassa essa senha para a controladora, que por sua vez dispara o método *Validar_Senha* no objeto da classe *Conta_Comum*. Se a senha for válida, a controladora solicitará a execução do método *Saldo_Conta* no objeto da classe *Conta_Comum*, que retornará a informação do saldo para a interface finalizando o processo (GUEDES, 2009, p. 243).

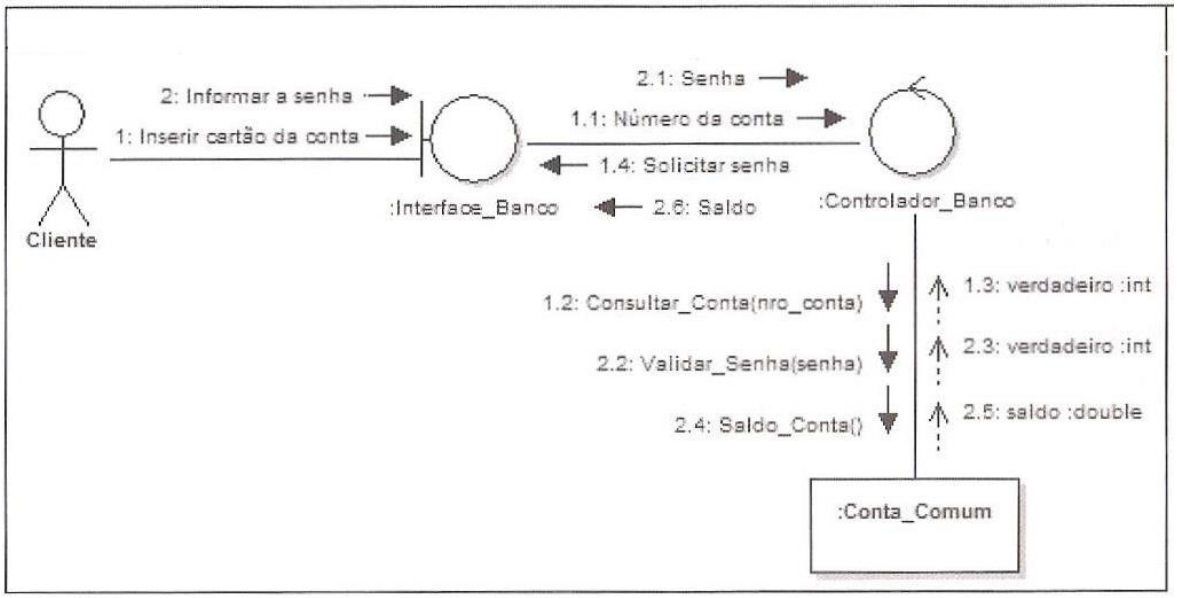


Figura 14: Diagrama de comunicação – Processo de Emissão de Saldo (GUEDES, 2009 p. 243).

4.3.7. Diagrama de tempo ou de temporização

O diagrama de tempo ou temporização tem como objetivo mostrar a mudança no estado ou condição de uma instância de uma classe durante um período. Este diagrama é utilizado normalmente para demonstrar a mudança no estado de um objeto no tempo em função da resposta a eventos externos. (GUEDES, 2009, p.42).

O diagrama de tempo é comumente empregado na modelagem de sistemas de tempo real ou sistemas que utilizam recursos de multimídia, onde o período de

tempo em que um objeto executa alguma rotina é um fator muito importante. (GUEDES, 2009, p.361).

Considere um cenário que tenha uma bomba e uma chapa elétrica de uma cafeteira. Vamos partir do pressuposto que entre o acionamento da bomba e o aquecimento da chapa devem-se passar pelo menos 10 segundos. Quando o reservatório de água se esvazia, a bomba desliga e a chapa não pode permanecer ligada por mais de 15 segundos depois deste evento. (FOWLER, 2004, p. 140).

As Figuras 15 e 16 trazem duas maneiras de modelagem deste cenário. A Figura 15 mostra as mudanças de estado, movendo-se de uma linha horizontal para outra, enquanto a Figura 16 mantem a mesma posição horizontal, porém mostra as mudanças de estado com uma cruz. (FOWLER, 2004, p. 140).

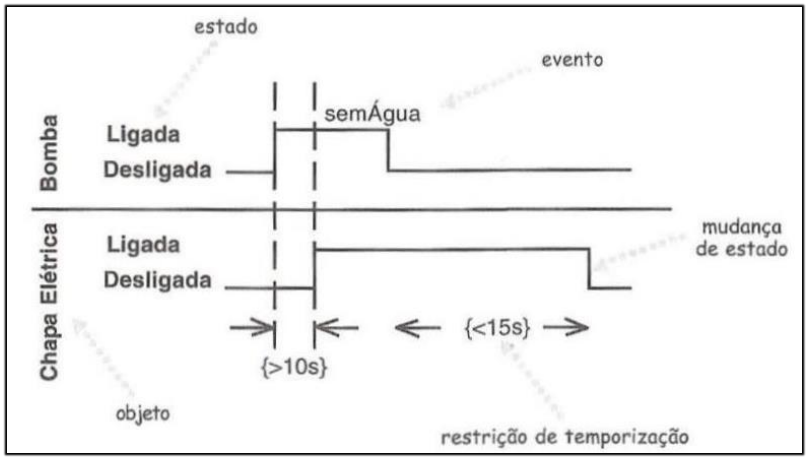


Figura 15: Diagrama de temporização mostrando os estados como linhas (FOWLER, 2004 p. 141).

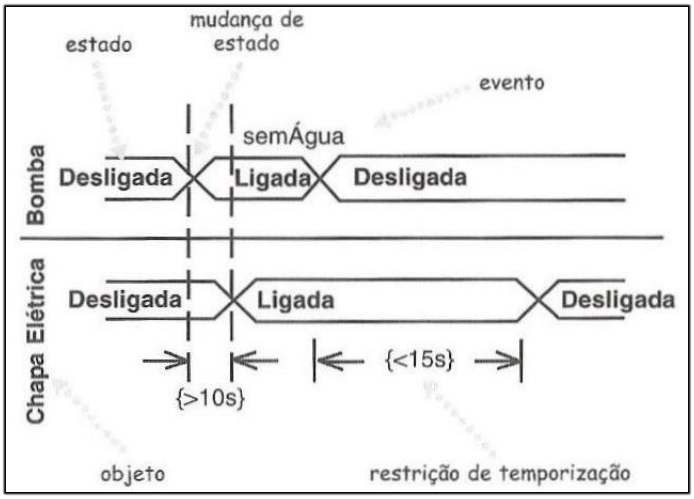


Figura 16: Diagrama de temporização mostrando os estados como áreas (FOWLER, 2004 p. 142).

5. ENGENHARIA REVERSA: UML COMO PROJETO

Conforme explicado no capítulo 4, a UML pode ser empregada de três maneiras distintas: escopo, projeto e linguagem de programação (FOWLER, 2004, p. 25-27). Porém, neste capítulo a UML será abordada como projeto, onde a partir de um sistema de gerenciamento de torneios de xadrez (*SiGTX*), será aplicada engenharia reversa no seu código-fonte com o intuito de mostrar na prática a utilização da UML.

Após prévio levantamento de informações sobre as ferramentas disponíveis para aplicação da engenharia reversa, constatou-se que a maioria delas possuía alto custo financeiro para serem adquiridas. Deste modo, definiu-se que para este estudo de caso seria utilizada ferramenta gratuita. Dentre as ferramentas pesquisadas, foram escolhidas duas, o *ArgoUML* e o *NetBeans*. O *ArgoUML* é uma ferramenta específica de modelagem, enquanto o *NetBeans* é um aplicativo que fornece mecanismos para desenvolvimento de *software*, ou seja, são aplicativos com finalidades diferentes, porém oferecem o mesmo recurso a ser explorado neste capítulo, no caso a engenharia reversa.

5.1. SiGTX - Sistema de gerenciamento de torneios de xadrez

O *SiGTX* surgiu de um projeto proposto pelo professor Luiz Eduardo Galvão Martins durante as aulas de Engenharia de *software* IV na Faculdade de Tecnologia de Americana (FATEC) no ano de 2011, onde o objetivo era desenvolver um sistema que fosse capaz de gerenciar torneios de xadrez tanto em nível amador, como em nível profissional. O sistema foi desenvolvido na linguagem Java. Para mais informações do funcionamento do sistema, vide os requisitos do sistema no “ANEXO A”.

A escolha do *SiGTX* deve-se muito pelo fato de ter sido desenvolvido nas melhores práticas de engenharia de *software* e por ser um sistema complexo, justificando a aplicação da engenharia reversa.

A Figura 17 mostra a estrutura do projeto na ferramenta *NetBeans*. É possível notar que o código do sistema está bem organizado, as classes **(destacado de**

azul) foram agrupadas por pacotes **(destacado de vermelho)**, dividindo de forma consciente cada funcionalidade do sistema e permitindo que as mesmas possam ser utilizadas como grupos (FOWLER, 2004, p. 96).

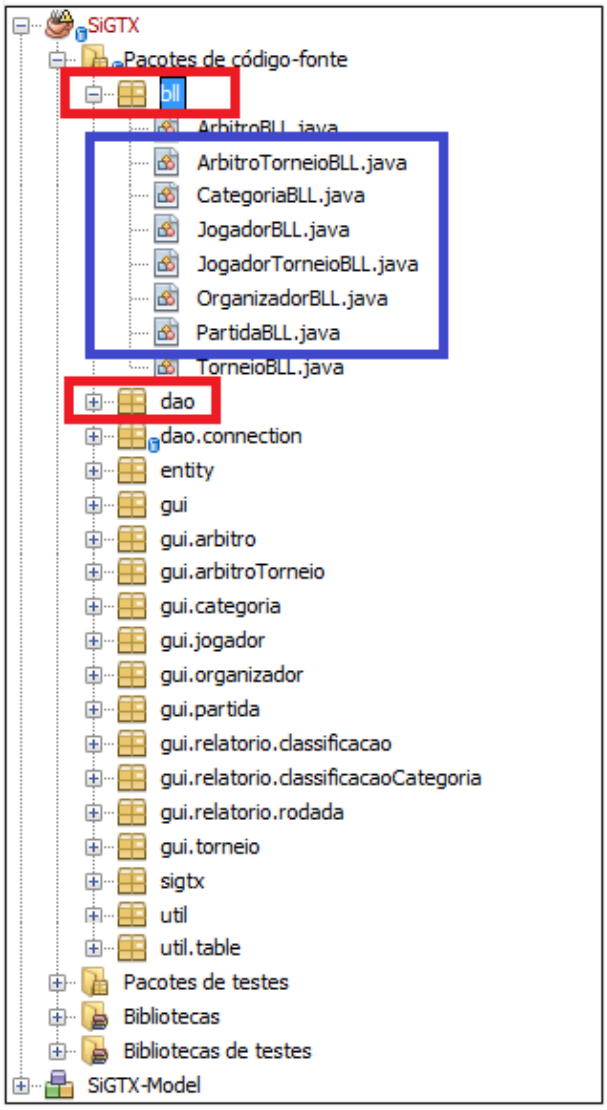


Figura 17: Estrutura do código-fonte do SiGTX (AUTORIA PRÓPRIA, 2013).

5.2. ArgoUML

O *ArgoUML* é uma ferramenta de modelagem desenvolvida por um grupo de desenvolvedores de código livre chamado *Tigris.* Este grupo está vinculado à Universidade da Califórnia em *Berkeley*. O *ArgoUML* é distribuído através da licença BSD (*Berkeley Software Distribution*), possui código aberto e está disponível em vários idiomas, inclusive no português brasileiro (ARGOUML, 2013).

Esta ferramenta foi desenvolvida em *Java*, podendo rodar em boa parte das plataformas. Além disso, ela fornece suporte ao XMI (*XML Metadata Interchange*) [5] e permite salvar os diagramas em formatos de imagem, como JPEG e PNG. A versão que foi utilizada para a aplicação da engenharia reversa foi a 0.34. A mesma fornece suporte para todos os nove diagramas da UML 1.4 (ARGOUML, 2013), conforme listados abaixo:

- Diagrama de classes;
- Diagrama de casos de uso;
- Diagrama de sequências;
- Diagrama de colaborações;
- Diagrama de componentes;
- Diagrama de pacotes;
- Diagrama de atividades;
- Diagrama de gráficos de estados;
- Diagrama de instalação ou implantação.

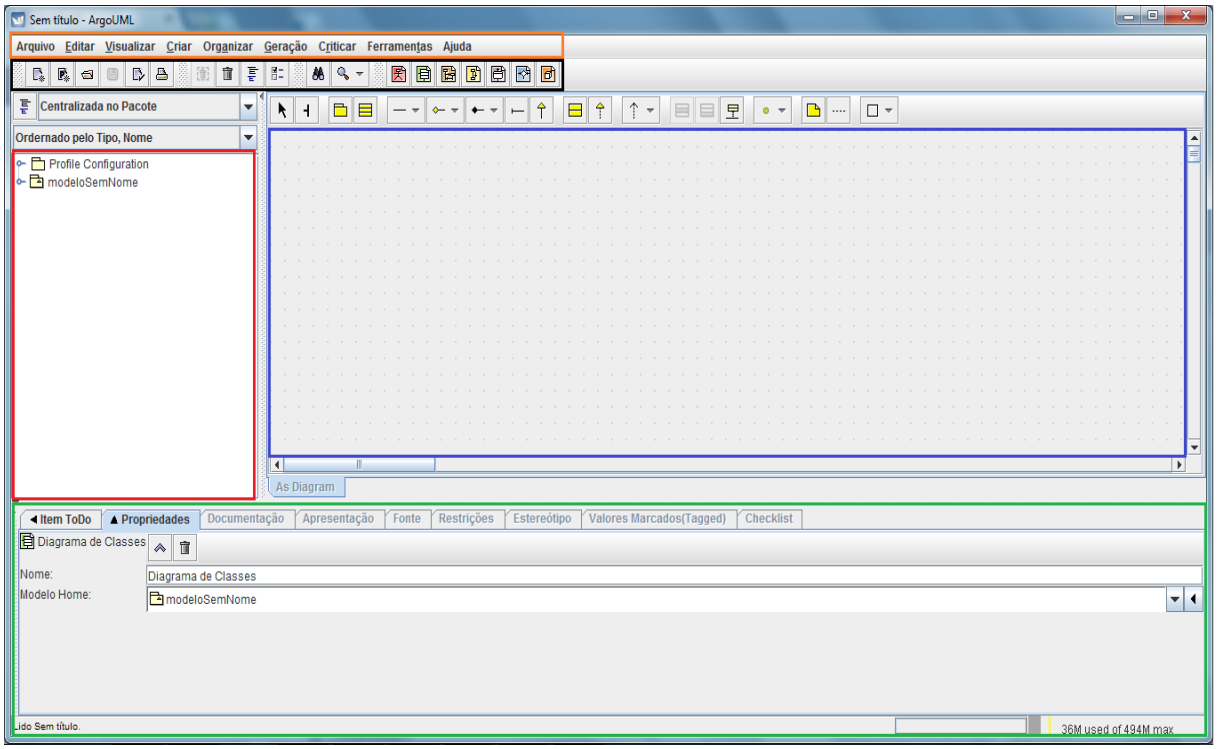


Figura 18: Interface da ferramenta ArgoUML (AUTORIA PRÓPRIA, 2013).

[5] Com a geração do XMI é possível trocar metadados (dados sobre outros dados) entre ferramentas de modelagem UML e entre ferramentas e depósitos de metadados MOF.

A Figura 18 traz a interface do *ArgoUML*. Na parte superior da ferramenta ficam os menus com as opções de manipulação de dados **(destacado de laranja)**. Abaixo deles se encontra a barra de ferramentas **(destacado de preto)** com diversas opções de modelagem. No canto esquerdo da tela **(destacado de vermelho)** ficam os projetos carregados na ferramenta. As notações gráficas ou diagramas são apresentados no centro da tela **(destacado de azul)**, enquanto as especificações de cada diagrama são mostradas na parte inferior do *ArgoUML* **(destacado de verde)**.

O “APÊNDICE A” traz informações de como instalar o *ArgoUML* 0.34.

5.3. Engenharia reversa com a ferramenta ArgoUML

Para aplicar a engenharia reversa nesta ferramenta, é necessário importar o código do sistema *SiGTX* para o *ArgoUML*. A Figura 19 mostra como proceder. Para iniciar a importação, basta ir até o menu ‘Arquivo’ e escolher a opção ‘Importar Fontes’ **(destacado de vermelho)**.

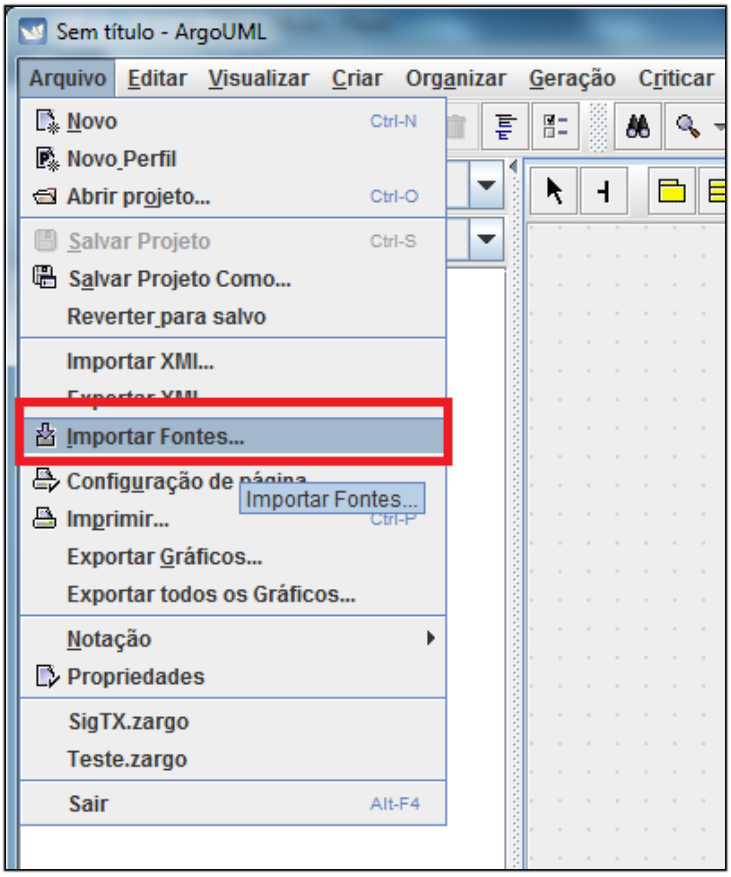


Figura 19: Importação do projeto SiGTX para o ArgoUML (AUTORIA PRÓPRIA, 2013).

Ao clicar nesta opção, o processo de importação do projeto é inicializado. O *ArgoUML* indica a finalização do processo, ao retornar a palavra ‘Feito’ **(destacado de verde)** na caixa de dialogo conforme mostra a Figura 20.

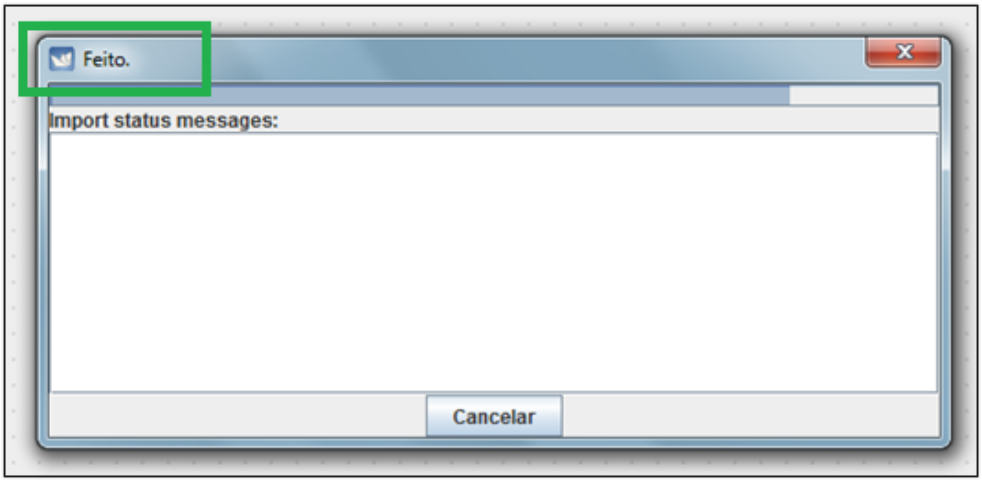


Figura 20: Final do processo de importação do projeto SiGTX (AUTORIA PRÓPRIA, 2013).

Ao fim da importação, a ferramenta *ArgoUML* recuperou boa parte da estrutura de pacotes **(destacado de vermelho)**, e outros elementos da UML como classes **(destacado de azul)**, generalizações, métodos **(destacado de verde)**, atributos **(destacado de laranja)**, interfaces e componentes **(destacado de preto)**.

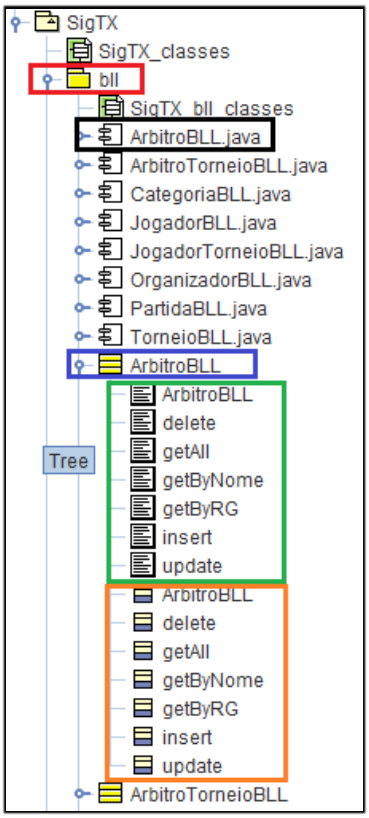


Figura 21: Elementos UML recuperados após a importação do projeto (AUTORIA PRÓPRIA, 2013).

O *ArgoUML* apresentou alguns diferenciais importantes, na visualização das propriedades dos elementos da UML, ele permite que o usuário veja todos os elementos contidos dentro do elemento selecionado, conforme pode ser visto na Figura 21.

Porém, durante o uso deste *software* também foi possível notar algumas limitações. A ferramenta não gerou automaticamente os diagramas de classes com seus atributos e operações, ele somente importou a classe com o seu nome. Essa situação é retratada na Figura 22.

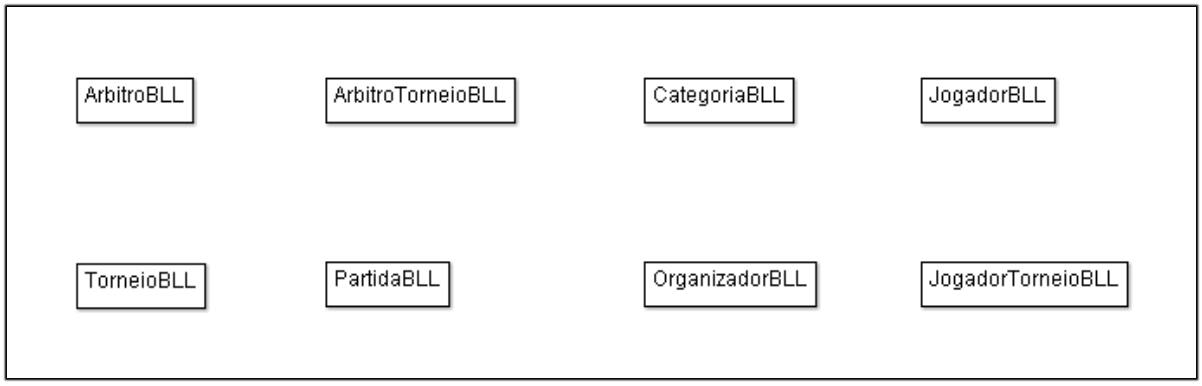


Figura 22: Classes sem os seus atributos e operações (AUTORIA PRÓPRIA, 2013).

Para criar os diagramas com seus devidos atributos e operações, é preciso arrastá-los, conforme mostra a Figura 23.

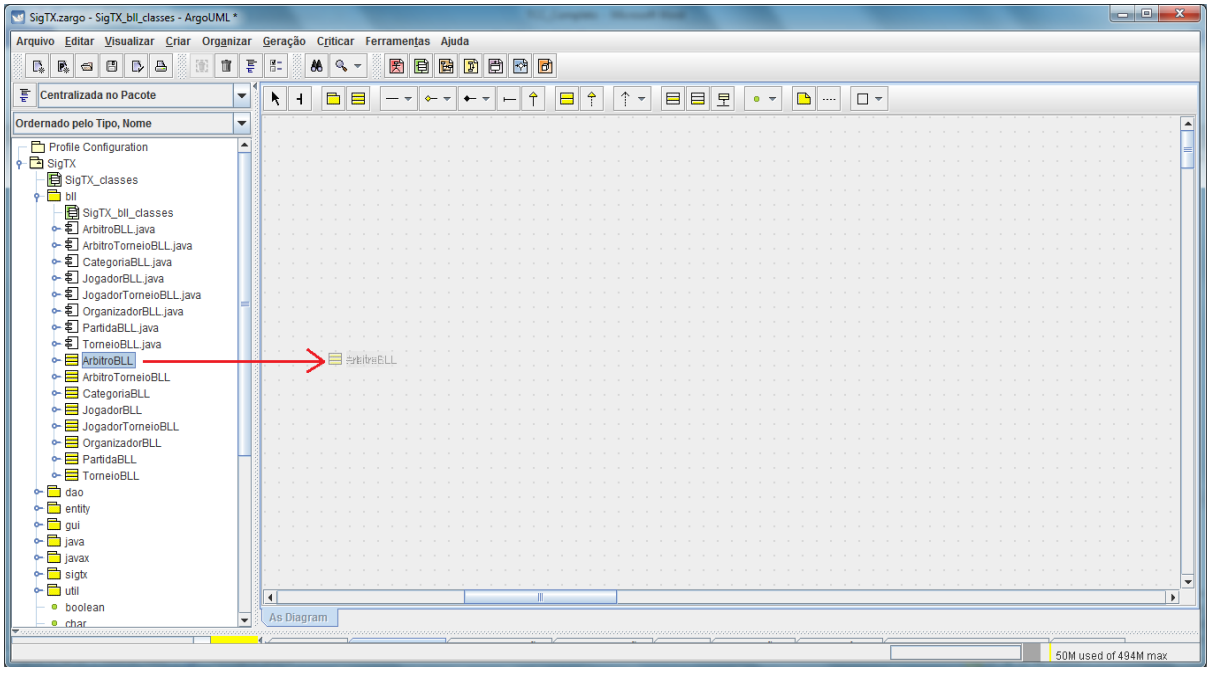


Figura 23: Arrastando elementos UML para o ArgoUML (AUTORIA PRÓPRIA, 2013).

O resultado da ação da Figura 23 é apresentado na Figura 24.

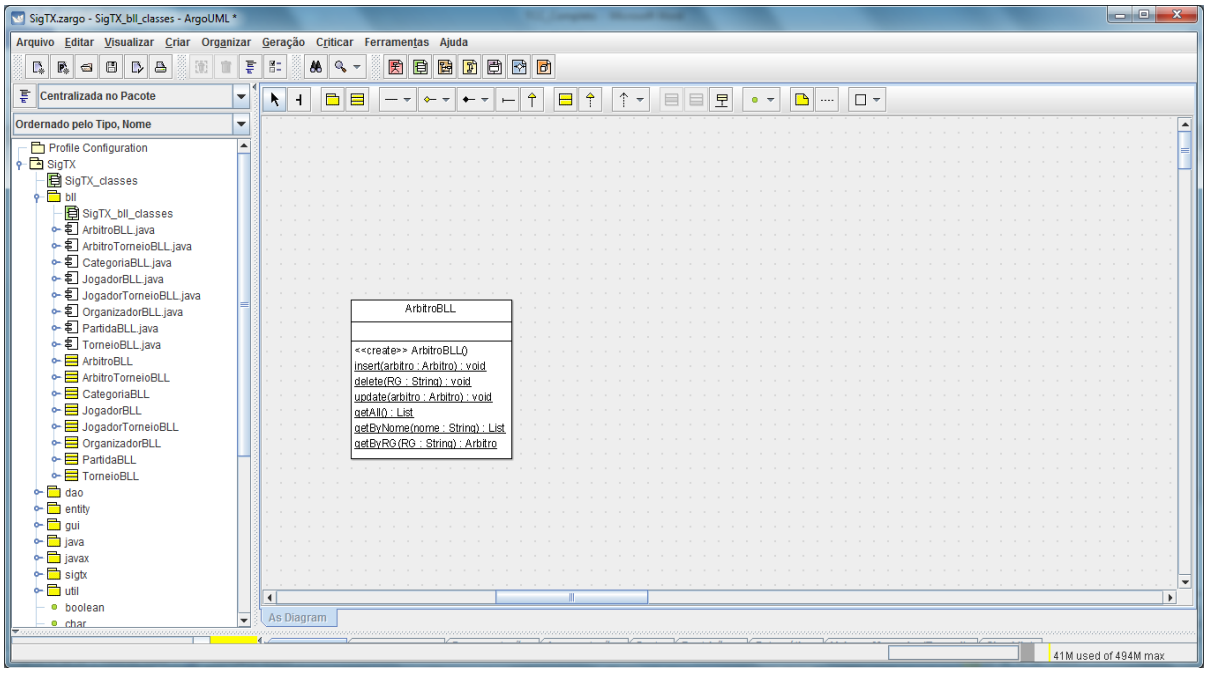


Figura 24: Classe ArbitroBLL com seus atributos e operações (AUTORIA PRÓPRIA, 2013).

Outro problema encontrado foi a ausência do desfazer. A falta deste recurso dificultou muito a modelagem, pois uma vez que se comete algum erro, é preciso apagar o elemento da UML e criá-lo novamente.

No mais a ferramenta se mostrou eficiente, criando os diagramas conforme a estrutura do projeto. A Figura 25 ilustra os diagramas de pacotes gerados a partir da engenharia reversa.

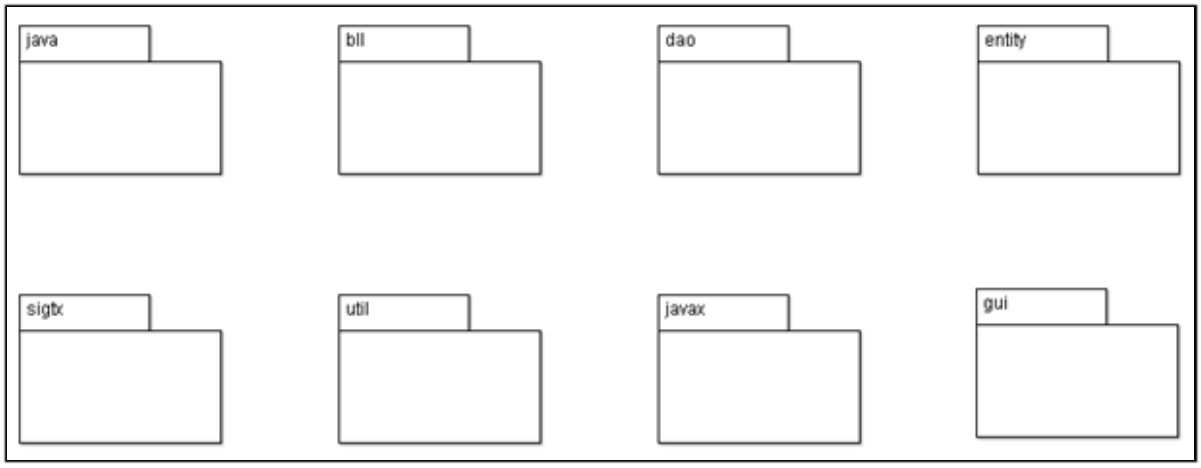


Figura 25: Pacotes do projeto SiGTX representados no ArgoUML após a engenharia reversa (AUTORIA PRÓPRIA, 2013).

O *ArgoUML* possibilita navegar pelos elementos da UML ao dar um duplo clique sobre o mesmo. A Figura 26 traz os diagramas de classes com seus respectivos atributos e métodos e o pacote *Connection*. O conteúdo foi apresentado após duplo clique no pacote *DAO*.

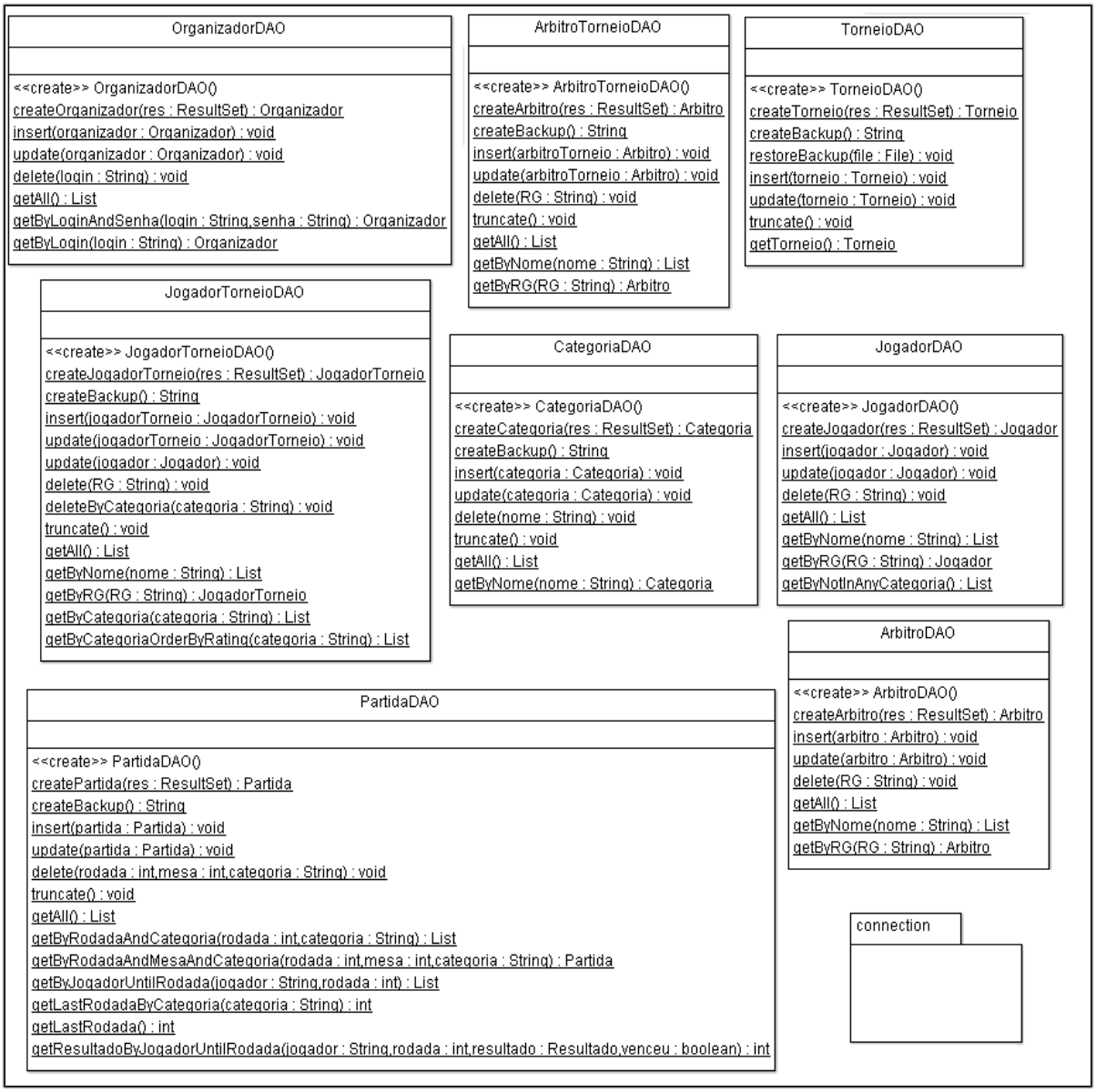


Figura 26: Os diagramas de classes e pacotes que estão dentro do pacote DAO (AUTORIA PRÓPRIA, 2013).

Os outros elementos da UML recuperados pelo *ArgoUML* podem ser vistos na Figura 27. Ela mostra as generalizações, interfaces e componentes gerados em sequência.

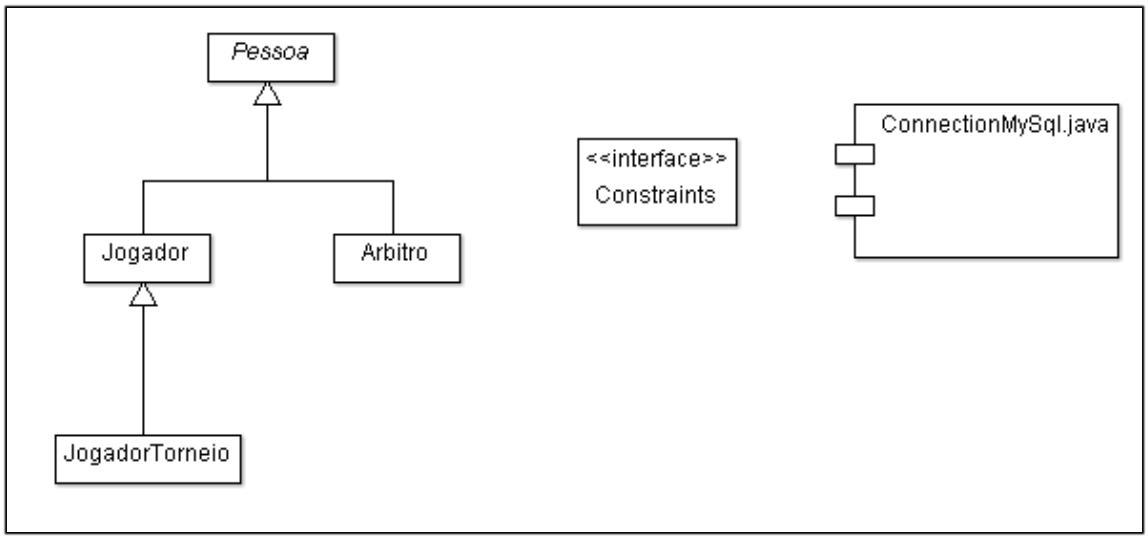


Figura 27: Outros elementos da UML recuperados: generalização, interface e componente. (AUTORIA PRÓPRIA, 2013).

5.4. NetBeans

Diferentemente do *ArgoUML*, o *NetBeans* não é uma ferramenta de modelagem. O NetBeans é uma IDE (*Integrated Development Environment*), ou seja, é um programa que fornece mecanismos para desenvolvimento de *software* em diversas linguagens como, por exemplo, *Java*, PHP, C++ e C. Esta ferramenta foi adquirida pela *Sun Microsystems* em 2000, após a mesma desistir do projeto de sua IDE *Java Workshop* (NETBEANS, 2013).

O *NetBeans* é uma ferramenta gratuita e pode rodar em qualquer sistema operacional que suporte JVM (*Java Virtual Machine*). O projeto *NetBeans* é distribuído sob a licença SPL (*Sun Public License*), que é uma variação da licença MPL (*Mozilla Public License*) (NETBEANS, 2013).

Para modelagem UML, o *NetBeans* oferece o *plugin NetBeans UML Project*. Este recurso permite que o usuário modele seus projetos, gere código fonte através de modelos, e aplique a engenharia reversa nos seus projetos. Este *plugin* fornece suporte para oito tipos de diagramas:

- Diagrama de classes;
- Diagrama de casos de uso;
- Diagrama de sequências;
- Diagrama de atividades;
- Diagrama de pacotes;

- ❖ Diagrama de gráficos de estados;
- ❖ Diagrama de instalação ou implantação;
- ❖ Diagrama de colaborações.

O *plugin NetBeans UML Project* foi integrado a IDE *NetBeans* na versão 6.0 (NETBEANS, 2013). A versão atual do *NetBeans* é a 7.3, porém a utilizada neste trabalho é a 7.0.

A Figura 28 mostra a interface da ferramenta *NetBeans*. Na parte superior da tela ficam os menus **(destacado de laranja)** que disponibilizam todas as opções que o programa oferece. Logo abaixo, estão os ícones da barra de ferramentas **(destacado de preto)** que tem como função facilitar a escolha de determinadas ações por parte do usuário. Do lado esquerdo da tela **(destacado de vermelho)** são mostrados os projetos correntes, enquanto ao centro **(destacado de azul)** é mostrado o conteúdo do elemento selecionado no projeto. Do lado direito da tela são mostradas as propriedades **(destacado de verde)** do elemento corrente e na parte inferior são mostradas os resultados **(destacado de marrom)** de compilação do projeto.

O processo de instalação da ferramenta *NetBeans 7.0* e o *plugin NetBeans UML Project* são detalhados no “APÊNDICE B”.

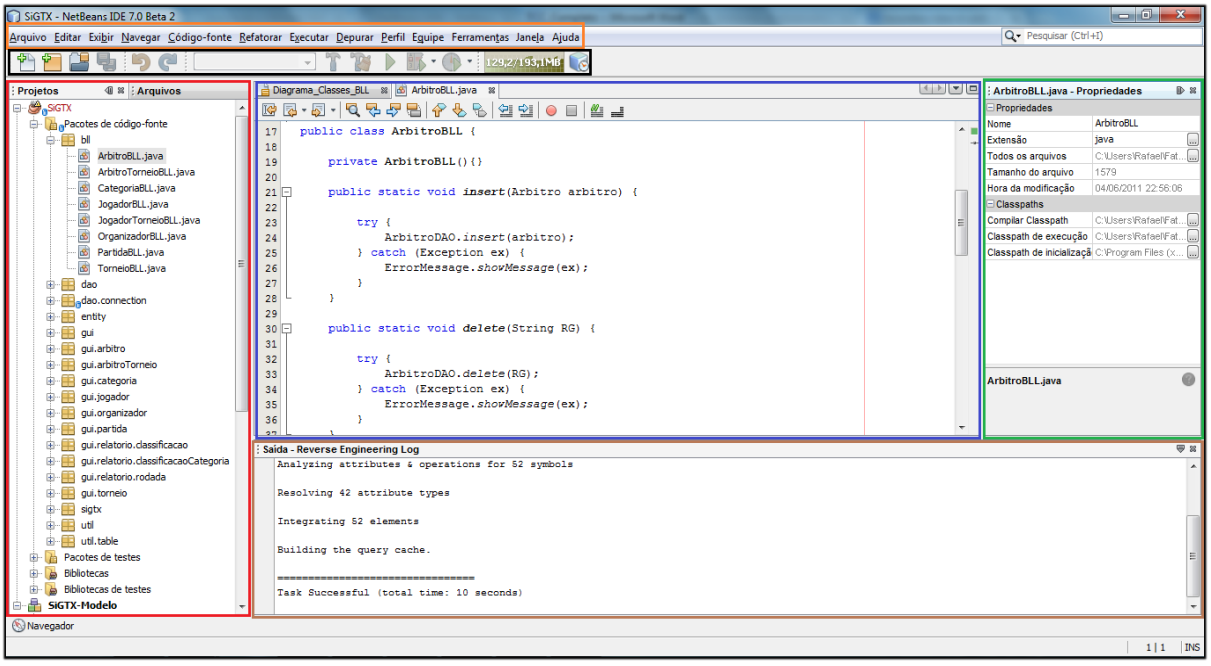


Figura 28: Interface da ferramenta NetBeans (AUTORIA PRÓPRIA, 2013).

5.5. Engenharia reversa com a ferramenta NetBeans

Para aplicar a engenharia reversa na ferramenta *NetBeans*, é preciso primeiramente abrir o projeto. Para tal, basta ir até o menu ‘Arquivo’ e selecionar a opção ‘Abrir Projeto’ **(destacado de vermelho)** conforme é mostrado na Figura 29.

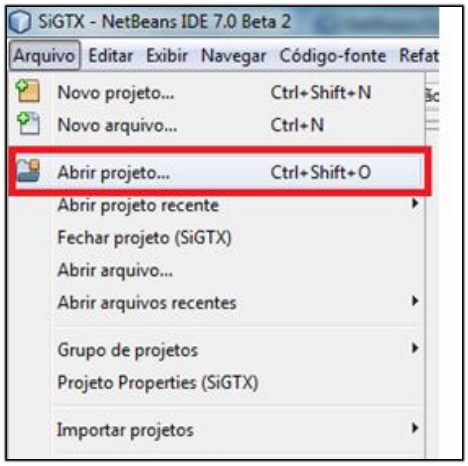


Figura 29: Procedimento para abrir o projeto (AUTORIA PRÓPRIA, 2013).

Ao clicar nesta opção, aparecerá uma tela para selecionar o projeto. Neste caso foi selecionado o projeto *SiGTX* conforme mostra a Figura 30.

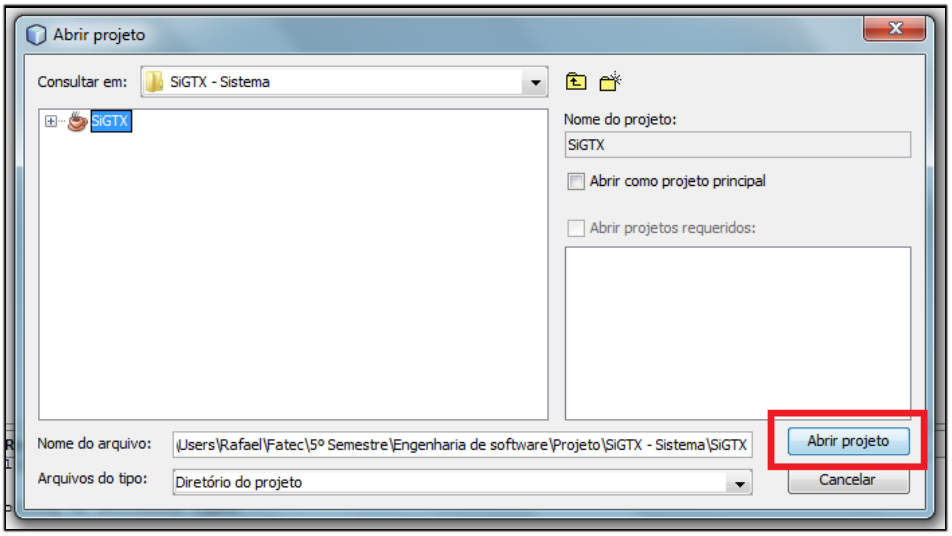


Figura 30: Selecionando o projeto SiGTX (AUTORIA PRÓPRIA, 2013).

Com o projeto selecionado e o *plugin NetBeans UML Project* devidamente instalado, é possível criar o modelo. Para isso, é preciso clicar com botão direito do mouse no projeto e escolher a opção ‘*Reverse Engineer*’ **(destacado de vermelho)**.

A Figura 31 retrata esta ação. Em seguida, basta informar o nome e a localização do modelo conforme é evidenciado na Figura 32.

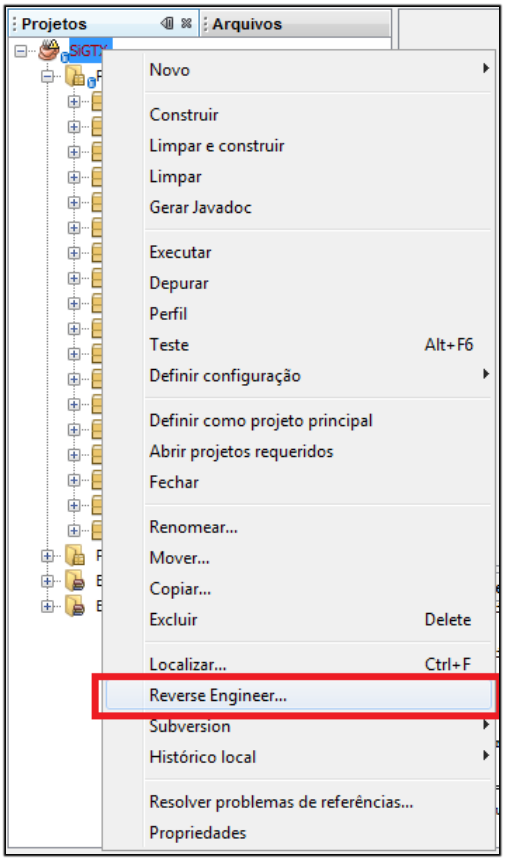


Figura 31: Criando o modelo a partir da engenharia reversa (AUTORIA PRÓPRIA, 2013).

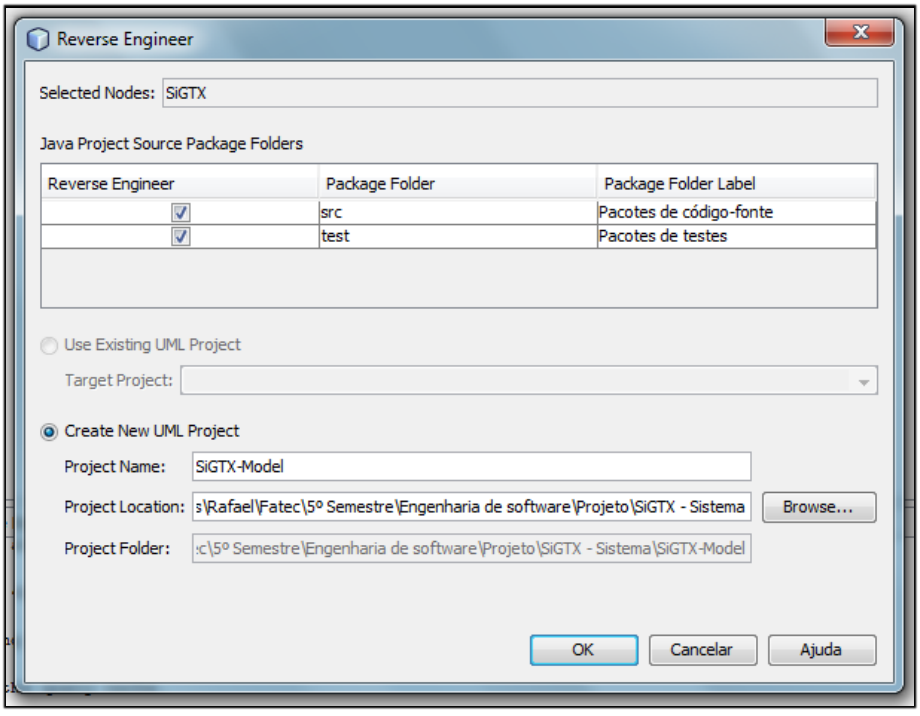


Figura 32: Definindo nome e localização do modelo (AUTORIA PRÓPRIA, 2013).

Após a definição destes detalhes, é só clicar no botão *'OK'* para que o *NetBeans* crie o modelo. Neste caso, o modelo criado foi o que está sendo mostrado na Figura 33. O mesmo é bem parecido com o que o *ArgoUML* criou, porém sem o recurso de rastreabilidade de elementos.

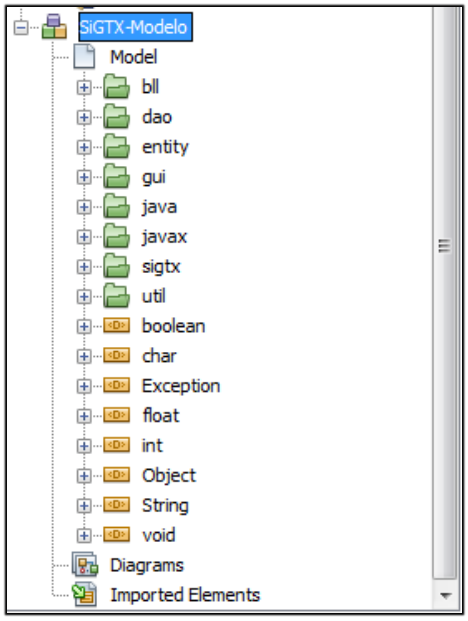


Figura 33: Estrutura do modelo gerado (AUTORIA PRÓPRIA, 2013).

O *NetBeans* não gera automaticamente os diagramas. Para que eles sejam criados, é preciso selecionar o pacote e apertar na opção '*Create Diagram From Select Elements*' **(destacado de vermelho)** conforme ilustrado na Figura 34.

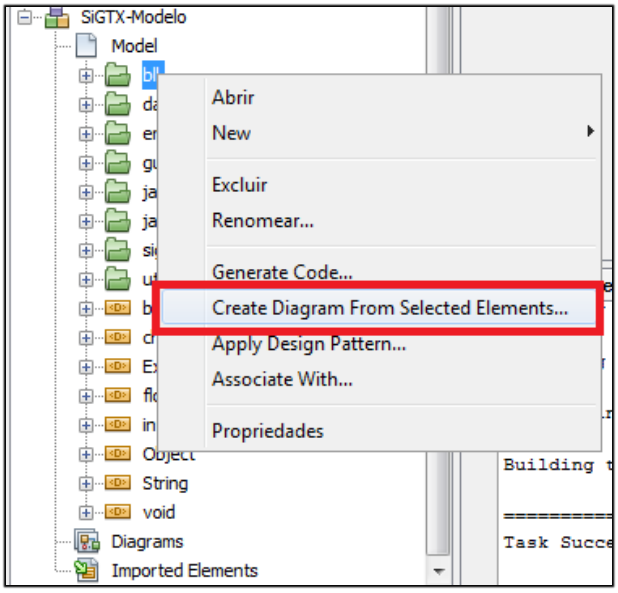


Figura 34: Criando diagramas UML (AUTORIA PRÓPRIA, 2013).

A Figura 35 mostra a tela que a ferramenta abre para selecionar o tipo de diagrama que deseja gerar. Neste caso, foi escolhida a opção diagrama de classes.

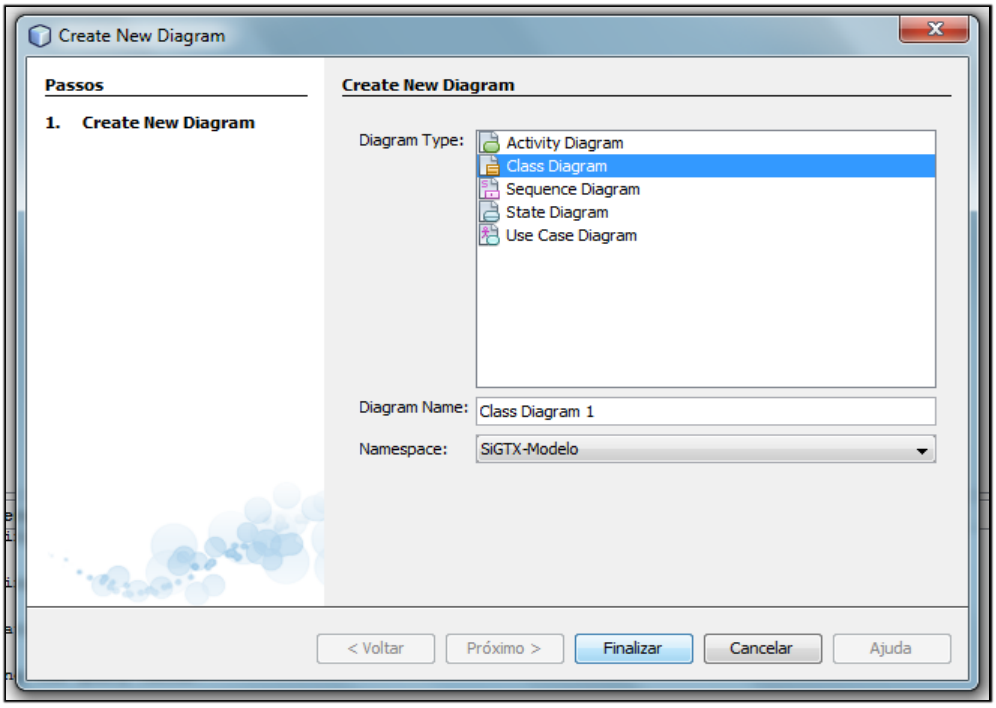


Figura 35: Gerando diagramas de classes (AUTORIA PRÓPRIA, 2013).

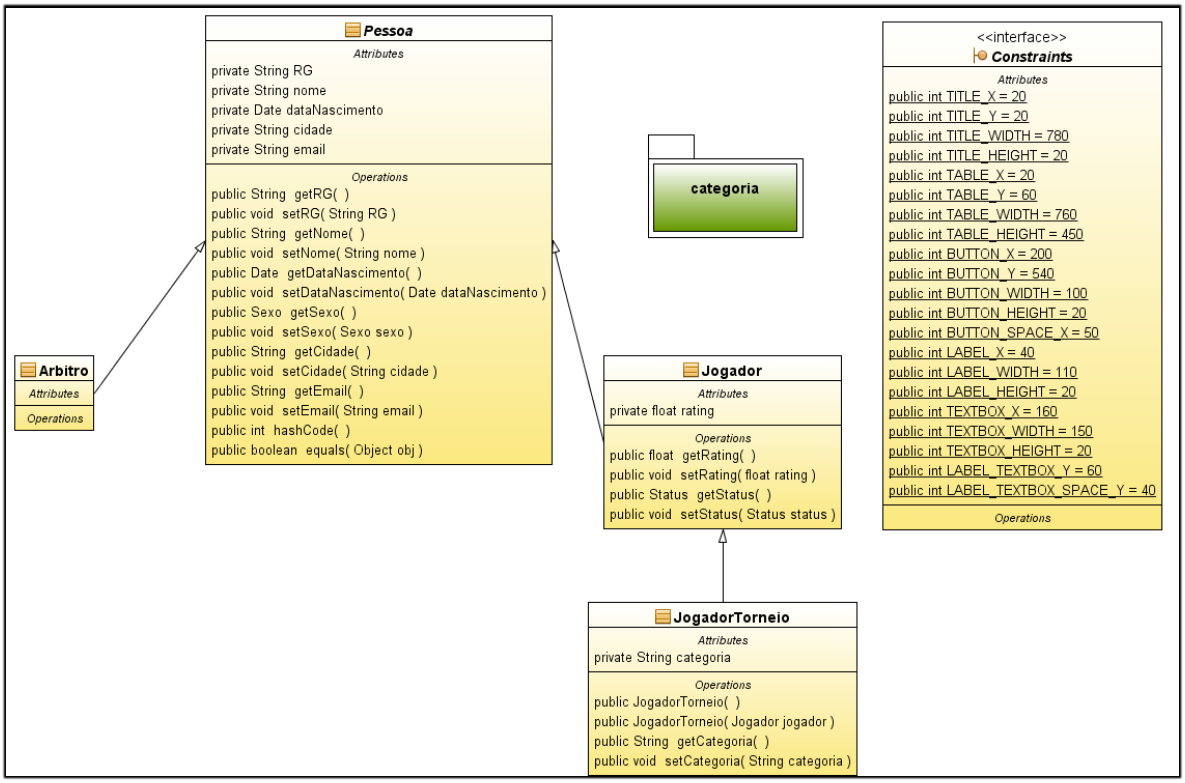


Figura 36: Generalizações, pacotes e interfaces recuperados na engenharia reversa.

(AUTORIA PRÓPRIA, 2013).

O *NetBeans* durante o processo de engenharia reversa recuperou a estrutura de pacotes, implementação de classes, relacionamentos de agregação e generalização, e interfaces.

A Figura 36 mostra os elementos de generalização, pacotes e interfaces recuperados respectivamente, enquanto a Figura 37 mostra a estrutura de diagramas de classes contidos no pacote BLL.

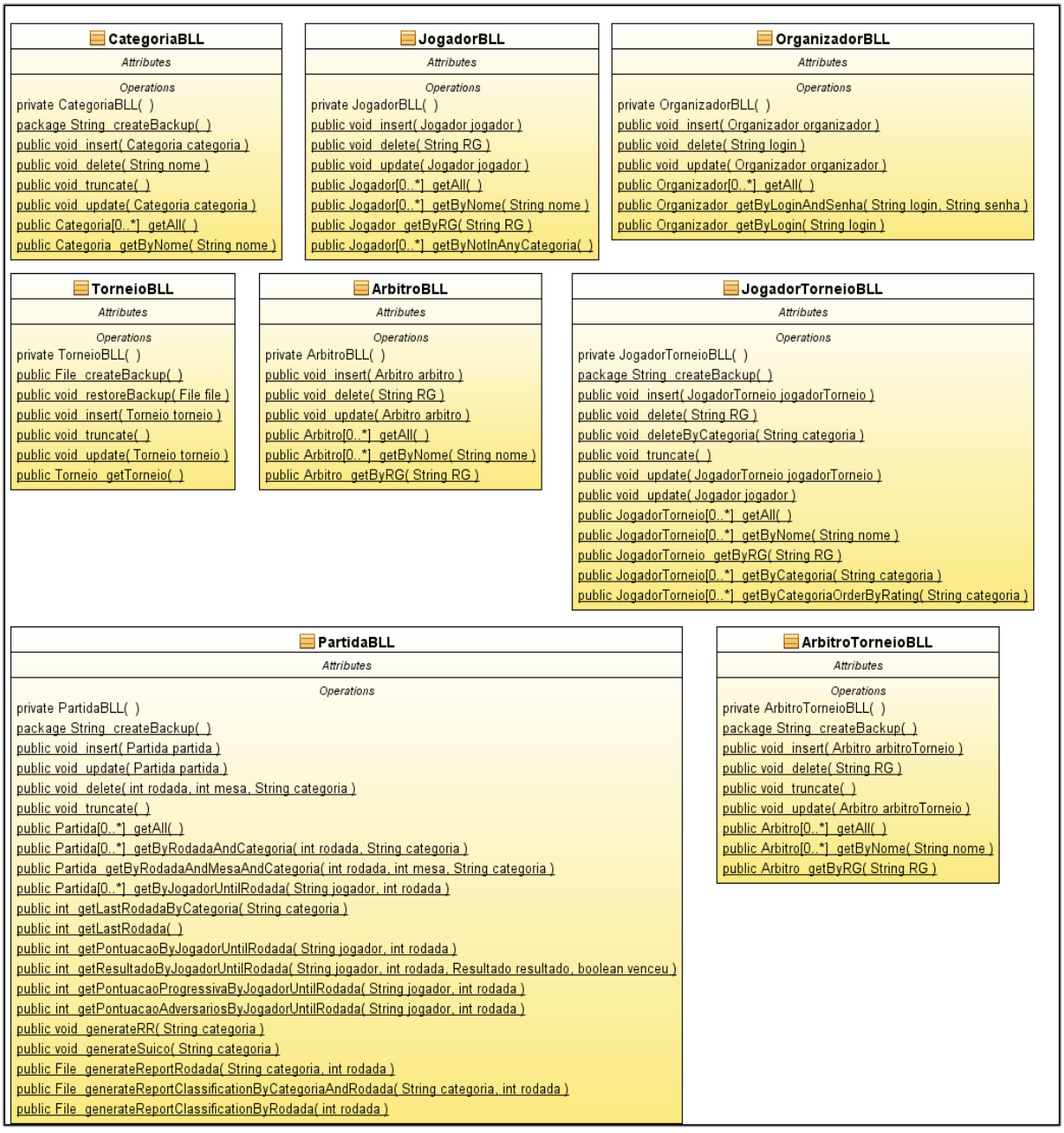


Figura 37: Diagramas de classes do pacote BLL gerados através da engenharia reversa

(AUTORIA PRÓPRIA 2013).

5.6. ArgoUML X NetBeans

Embora o foco deste trabalho não seja comparar as ferramentas escolhidas para a aplicação da engenharia reversa, durante a realização do processo foi possível notar algumas vantagens de uma ferramenta em relação à outra.

O *ArgoUML* é uma ferramenta mais preparada para modelagem, uma vez que ela foi desenvolvida especialmente para este propósito. O resultado disto é uma ferramenta que fornece opções que simplificam o processo de modelagem e disponibiliza recursos que facilitam a navegação entre os elementos da UML.

O *NetBeans* obtém vantagem no que diz respeito à visualização dos elementos modelados. Os artefatos gerados fornecem mais informações sobre o código-fonte do sistema. Além disso, por se tratar de uma IDE, o processo de modelagem pode ser realizado em conjunto com a escrita do código.

As Tabelas 2 e 3 mostram o desempenho das duas ferramentas na aplicação da engenharia reversa. A Tabela 2 apresenta a recuperação dos elementos da UML e a Tabela 3 traz a recuperação dos relacionamentos destes elementos. O *ArgoUML* leva ligeira vantagem sobre o *NetBeans*.

Tabela 2: Elementos recuperados durante a engenharia reversa (AUTORIA PRÓPRIA, 2013).

Ferramenta	Classe	Atributo	Método	Interface	Componente
ArgoUML	X	X	X	X	X
NetBeans	X	X	X	X	

Tabela 3: Relacionamentos do código (AUTORIA PRÓPRIA, 2013).

Ferramenta	Associação simples	Generalização	Composição	Abstração de classe
ArgoUML		X		X
NetBeans	X		X	

5.7. Análise sobre a engenharia reversa

A modelagem normalmente é aplicada antes da implementação do sistema, nas fases de análise de requisitos e projeto. Desta forma a modelagem auxilia os desenvolvedores no processo de desenvolvimento do *software*, servindo de base para a escrita do código.

A engenharia reversa surge como alternativa para modelagem de *softwares* que estão em processo de desenvolvimento ou que já foram concluídos. Através dela é possível gerar artefatos que representam exatamente a estrutura do sistema a partir do código-fonte. Com isso, a equipe de desenvolvimento pode visualizar em um documento comum as funcionalidades do sistema.

Caso a engenharia reversa seja aplicada durante o processo de desenvolvimento, será muito útil para visualizar o que já foi realizado e projetar o que ainda é necessário fazer. Porém, se ela for aplicada após a conclusão do projeto, os modelos representarão exatamente toda a estrutura do sistema, sendo fundamental para possíveis manutenções que possam surgir ao longo do tempo.

O estudo de caso abordando a engenharia reversa utilizando as ferramentas *ArgoUML* e *NetBeans* comprova a eficiência deste recurso. Os diagramas gerados representam exatamente a estrutura e consequentemente o código-fonte do sistema *SiGTX* apresentado na Figura 17. Os resultados da engenharia reversa podem ser vistos nas Figuras 25, 26, 27, 36 e 37.

Portanto, a modelagem é peça importante no desenvolvimento de um *software*, fornecendo uma visão padronizada da estrutura e das funcionalidades do sistema. Embora, o ideal seja a aplicação da modelagem no início do projeto, a engenharia reversa pode ser uma alternativa eficiente para aplicar a modelagem em sistemas que estão em andamento ou que já tiveram todas as etapas do processo de desenvolvimento concluídas.

6. CONCLUSÃO

O presente trabalho abordou como tema principal, a modelagem de sistemas utilizando a linguagem de modelagem UML, tendo em seu início discorrido sobre os fatores que auxiliaram para o seu surgimento e, em qual fase do processo de desenvolvimento ela se aplica. Em seguida, foi abordada a importância de se fazer a modelagem, sobretudo, em sistemas complexos, enfatizando principalmente a modelagem em sistemas que são desenvolvidos no paradigma de orientação a objetos.

Em uma segunda etapa do trabalho, foi dado um maior destaque para a UML, descrevendo como ela pode ser aplicada e apresentando os recursos que a mesma oferece. Além disso, foram explicados os conceitos básicos e as funcionalidades dos diagramas que fazem parte desta linguagem.

Por fim, foi aplicada a engenharia reversa em um sistema de gerenciamento de torneios de xadrez. O objetivo foi mostrar na prática a utilização da UML, bem como propor uma alternativa para a aplicação da modelagem.

6.1. Estudos Futuros

Com a conclusão do presente trabalho, pode-se propor alguns estudos futuros tendo como base o tema aqui apresentado. Por exemplo, um estudo de caso real, que compare o processo de desenvolvimento de um *software* que aplicou a modelagem e outro que não fez uso dela. Com isso, seria possível fazer um levantamento dos recursos utilizados, do tempo despendido e o cumprimento de prazos do projeto, provando com resultados reais os benefícios de se utilizar a modelagem.

Outro ponto bastante interessante para explorar são as ferramentas de modelagem. Poderia ser feito um levantamento das ferramentas que fornecem o recurso de engenharia reversa e aplica-las em um mesmo projeto com o intuito de encontrar a melhor ferramenta para executar esta técnica.

Nos dias de hoje, a UML tem perdido espaço para duas novas tecnologias, o BPMN *(Business Process Modeling Notation)* e o TDD *(Test Driven Development)*.

Embora o foco da primeira seja a modelagem de processo de negócio e a segunda seja uma prática de desenvolvimento, ambas as ferramentas estão sendo empregadas para substituírem a UML no processo de desenvolvimento. Esta mudança de tecnologias poderia ser muito bem argumentada em um trabalho acadêmico.

Por fim, com o crescimento da utilização das metodologias ágeis, seria pertinente elaborar um estudo relacionando as tais com a UML, evidenciando onde a mesma pode ser aplicada, bem como, apresentando as vantagens e desvantagens que tal prática pode trazer para o processo de desenvolvimento de software.

6.2. Considerações Finais

Destacar os motivos de se utilizar a modelagem, bem como, apresentar uma linguagem para este processo, foi uma das maneiras encontradas para demonstrar as organizações o quão é importante a utilização de uma modelagem adequada no processo de desenvolvimento de sistemas computacionais.

Com os resultados obtidos na aplicação da engenharia reversa, foi possível ver em modelos a estrutura e o comportamento do sistema de gerenciamento de xadrez (SiGTX). Os artefatos gerados transmitem uma visão padronizada do projeto, fazendo com que o entendimento da equipe de desenvolvimento seja uniforme. Além disso, a UML mostrou-se eficiente, fornecendo diferentes tipos de diagrama para a especificação do sistema, e consequentemente, aumentando o seu nível de detalhamento.

O ideal da modelagem é ser aplicada antes da codificação do sistema, uma vez que um sistema planejado é um sistema menos propenso a erros. Porém, a engenharia reversa surge como alternativa para transmitir uma visão parcial (durante o processo de desenvolvimento) e final (após o processo de desenvolvimento) do sistema.

Tendo em vista, que o mercado está cada vez mais competitivo, torne-se imprescindível a utilização correta dos recursos que dispõem, otimizando processos, cumprindo os prazos estimados e oferecendo produtos de qualidade. A modelagem é um destes recursos e, portanto, sua utilização também deve ser planejada.

7. REFERÊNCIAS

ANDRADE, Maria Margarida de. **Introdução à metodologia do trabalho científico.** 9ª ed. São Paulo: Atlas, 2009. 160 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Citação**: NBR-10520/ago - 2002. Rio de Janeiro: ABNT, 2002.

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

ARGOUML. Open Source Software Engineering Tools. Disponível em: <http://argouml.tigris.org/>. Acesso em: 13/05/2012.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Editora Campus/Elsevier, 2007 (p. 5-6).

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. **UML guia do usuário**. 2. ed. Rio de Janeiro, 2006. 474 p.

BRAUDE, Eric. **Projeto de software**: da programação à arquitetura. 1. ed. Porto Alegre – RS: ARTMED EDITORA S.A, 2004 (p. 22- 28).

FOWLER, Martin. **UML essencial**: um breve guia para linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre – RS: ARTMED EDITORA S.A, 2004. 160 p.

FURLAN, José Davi. **Modelagem de objetos através da UML**: the unified modeling language. 1. ed. São Paulo: Makron Books, 1998. 329 p.

GUEDES, Gilleanes T. A. **UML 2**: uma abordagem prática. São Paulo: Novatec Editora, 2009. 488 p.

LARMAN, Craig. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre – RS: ARTMED EDITORA S.A, 2005. 696 p.

LOBO, Edson. **Guia prático de engenharia de software**. 1. ed. Porto Alegre – RS: ARTMED EDITORA S.A, 2009. 128 p.

MAGELA, Rogerio. **Engenharia de software aplicada:** fundamentos. 2. ed. Rio de Janeiro – RJ: Alta Books, 2006. 418 p.

MELO, Ana Cristina. **Exercitando modelagem em UML**. 1. ed. Rio de Janeiro – RJ: Brasport, 2006. 117 p.

NETBEANS. NetBeans IDE and Plugins NetBeans UML Project. Disponível em: < http://netbeans.org/>. Acesso em: 18/05/2012.

RAMOS, Ricardo Argenton. **Treinamento prático em UML**. 1. ed. São Paulo – SP: Digerati Books, 2006. 144 p.

SCHACH, Stephen R. **Engenharia de Software:** os paradigmas clássico orientado a objetos. 7. ed. Porto Alegre – RS: AMGH, 2010. 618 p.

SHALLOWAY, Alan; TROTT, James. **Explicando padrões de projeto:** uma nova perspectiva em projeto orientado a objeto. 1. ed. Porto Alegre – RS: ARTMED EDITORA S.A, 2002 (p. 55).

ANEXO A - Documento de Requisitos do SiGTX

Finalidade do Produto

O objetivo do desenvolvimento do *SiGTX* é oferecer à comunidade enxadrística brasileira uma ferramenta de apoio à realização de torneios de Xadrez, facilitando a organização e gerenciamento de tais torneios, tanto em nível amador (para escolas, clubes etc.) como em nível profissional (federações, confederações, torneios internacionais). Esta será a 1ª versão a ser desenvolvida do *SiGTX*, portanto será referenciada como versão 1.0 (*SiGTX v1.0*).

Stakeholders do Sistema

Os *stakeholders* do sistema são os interessados no desenvolvimento do *software*. Neste caso, os *stakeholders* são: organizadores de torneios de Xadrez, árbitros e jogadores que participam do torneio. Os usuários diretos deste *software* são os organizadores de torneios, eles "alimentarão" o sistema e receberão os resultados das funções realizadas pelo *software*.

Visão Geral do Sistema

O *SiGTX* deve ter a capacidade de gerenciar torneios de Xadrez. Para tanto, deve ser considerado que um torneio possui os seguintes atores: organizadores, jogadores e árbitros. Os jogadores participarão das partidas ao longo do torneio, e os árbitros acompanharão as partidas e anotarão os resultados de cada partida. A interação direta com o *software*, será feita pelos organizadores do torneio.

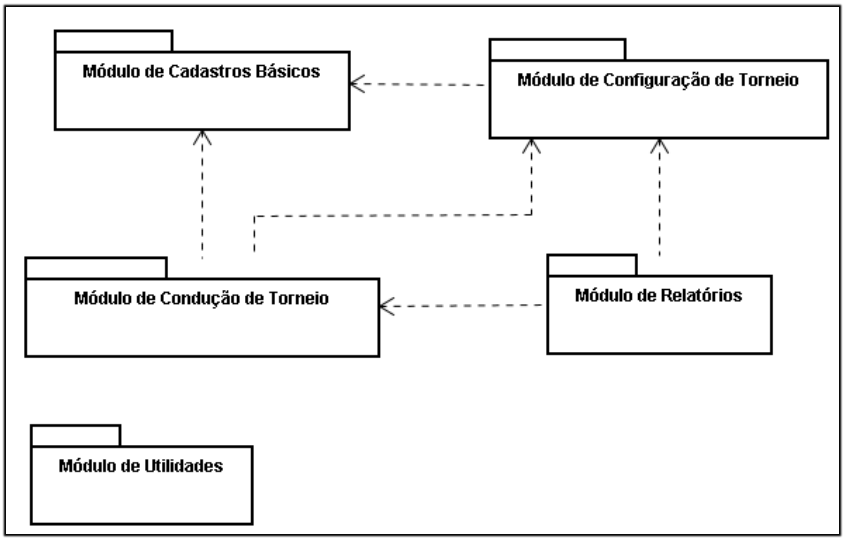
Em um torneio de Xadrez, podem existir muitas rodadas. Cada rodada consiste em um conjunto de partidas. Uma partida ocorre entre dois jogadores, um utilizando as peças brancas e o outro as peças pretas. Toda partida terá um resultado final, que pode ser: vitória das brancas, vitória das pretas ou empate. A vitória deve ser caracterizada como vitória por jogo ou vitória por *WO*.

O árbitro é quem verifica o resultado final da partida e atribui a pontuação correspondente para os jogadores. O *SiGTX* deve permitir a configuração da pontuação das partidas. As pontuações ocorrerão da seguinte forma: pontos para a

vitória, pontos para o empate, pontos para a derrota por jogo e pontos para a derrota por *WO.*

Existem duas funcionalidades fundamentais do *SiGTX* que precisam ser consideradas, são elas: emparceiramento das partidas (por rodada), e critério de desempate. Os sistemas de emparceiramento mais utilizados são: sistema *Schuring* (*Round-Robin)* e sistema Suíço. *SiGTX* deve dar suporte para estas duas modalidades de emparceiramento. O sistema *Schuring* normalmente é adotado para torneios com poucos participantes, neste sistema todos os jogadores jogam contra todos. O sistema Suíço é empregado para torneios com muitos participantes.

Ao término de cada rodada, os resultados das partidas devem ser alimentados no sistema. *SiGTX* deve ser capaz de informar a classificação dos jogadores do torneio, para cada rodada realizada. Após a última rodada, a classificação informada é a classificação final do torneio. Para gerar a classificação final, o sistema deve levar em consideração os critérios de desempate adotado no torneio. Os critérios de desempate podem variar de torneio para torneio, ficando ao cargo dos organizadores do torneio decidirem quais serão os critérios adotados. O diagrama baixo apresenta uma visão dos módulos que deverão compor o *SiGTX.*



Requisitos Não-Funcionais do Sistema

RNF01 – Ambiente Operacional: sistema operacional Windows (XP, Vista ou 7), monitor gráfico, *mouse*, teclado, HD com espaço suficiente para rodar o sistema (instalação do *software* e banco de dados). Sistema monousuário.

RNF02 – Interface com o Usuário: interação baseada em *mouse* e janelas (ambiente gráfico). O sistema deve ser amigável, com telas intuitivas e autoexplicativas. Padronizar local de exibição das mensagens do sistema.

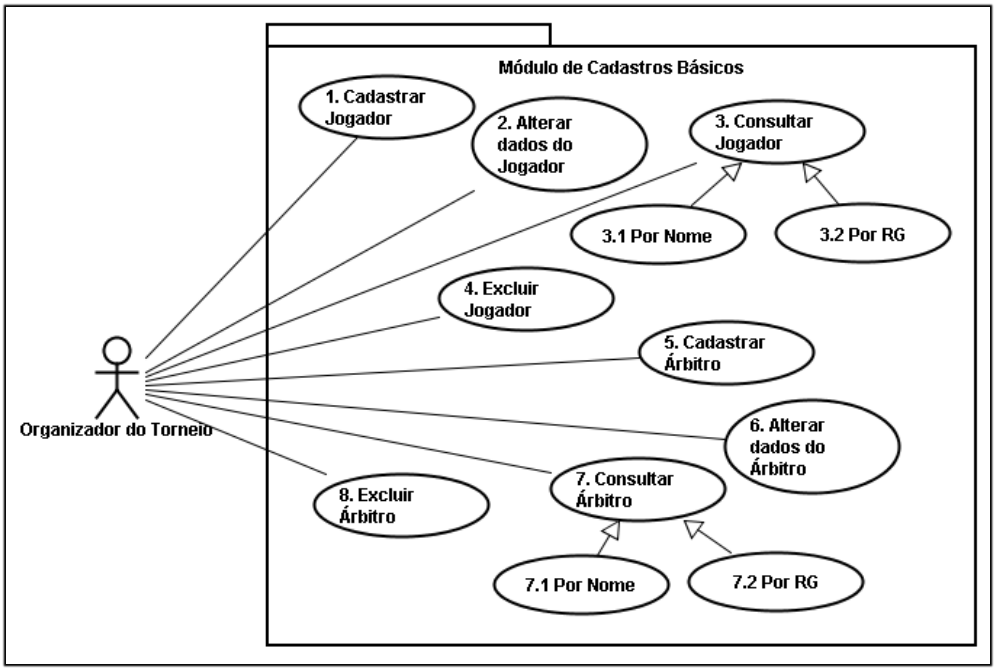
RNF03 – Ambiente de Desenvolvimento: o *software* deverá ser desenvolvido em linguagem Java, no ambiente *NetBeans*.

RNF04 – Documentação do Software: o *software* desenvolvido deverá ser completamente documentado, tanto em nível de *design* (diagramas UML) como de implementação (código comentado).

Requisitos Funcionais do Sistema

Os requisitos funcionais do sistema estão expressos na forma de casos de uso, de acordo com os diagramas de casos de uso apresentados a seguir.

Módulo de Cadastros Básicos



RF01 – Cadastrar Jogador: para o jogador participar do torneio ele precisa estar cadastrado no sistema. Os dados do cadastramento são: RG, nome, data de nascimento, sexo, cidade de origem, e-mail, *rating*, *status* (normal, mestre FIDE, mestre internacional, grande mestre). Apenas e-mail e *rating* são dados opcionais.

RF02 – Alterar Jogador: com exceção do RG, qualquer dado pode ser alterado. Digitar o RG e apresentar em uma janela os dados a serem alterados.

RF03.1 – Consultar Jogador por Nome: digitar o nome, ou parte dele, e trazer em uma janela os nomes correspondentes a *string* digitada. Escolher o jogador da lista de nomes apresentados. Mostrar todos os dados do jogador escolhido.

RF3.2 – Consultar Jogador por RG: digitar o RG e mostrar em uma janela os dados do jogador correspondente.

RF04 – Excluir Jogador: digitar o RG e apresentar em uma janela os dados do jogador, pedir confirmação antes de excluir do cadastro.

RF05 – Cadastrar Árbitro: para o árbitro participar do torneio ele precisa estar cadastrado no sistema. Os dados do cadastramento são: RG, nome, data de nascimento, sexo, cidade de origem e e-mail. Apenas e-mail é opcional.

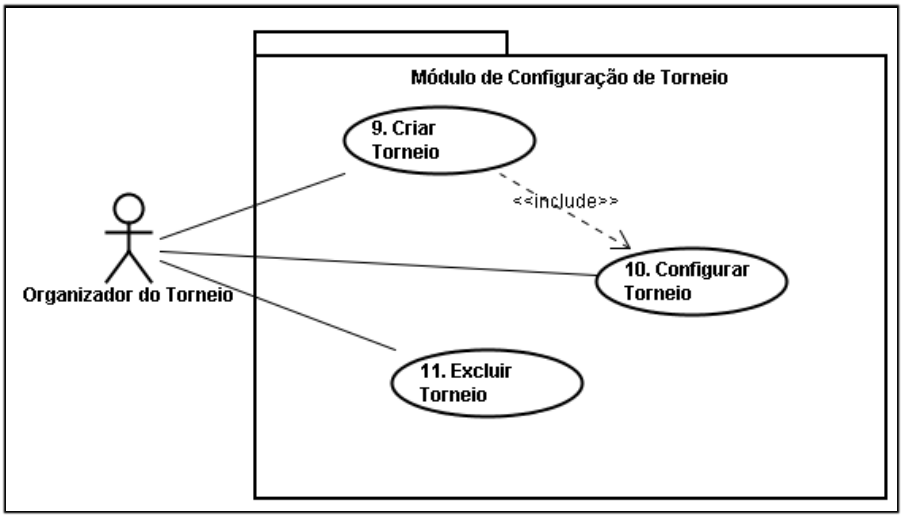
RF06 – Alterar Árbitro: com exceção do RG, qualquer dado pode ser alterado. Digitar o RG e apresentar em uma janela os dados a serem alterados.

RF07.1 – Consultar Árbitro por Nome: digitar o nome, ou parte dele, e trazer em uma janela os nomes correspondentes a *string* digitada. Escolher o árbitro da lista de nomes apresentados. Mostrar todos os dados do jogador escolhido.

RF7.2 – Consultar Árbitro por RG: digitar o RG e mostrar em uma janela os dados do árbitro correspondente.

RF08 – Excluir Árbitro: digitar o RG e apresentar em uma janela os dados do árbitro, pedir confirmação antes de excluir do cadastro.

Módulo de Configuração do Torneio



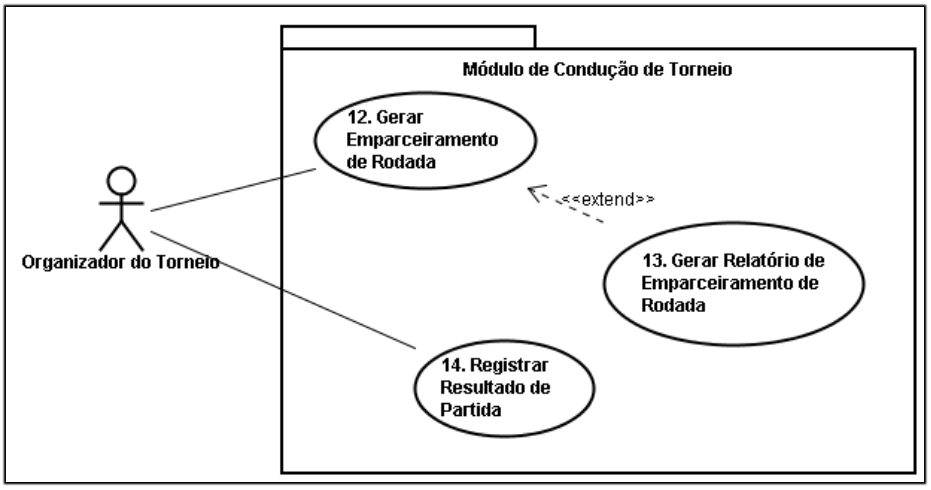
RF09 – Criar Torneio: nesta primeira versão do *software*, o mesmo fará o controle e manterá os dados de apenas um torneio por vez. Dessa forma, criar o torneio significa inicializar todas as tabelas necessárias para a configuração do torneio e atribuir um nome ou título para o torneio (Ex: I Torneio de Xadrez Amador da FATEC Americana). Ao criar o torneio o sistema deve criar um diretório (na raiz do HD do computador) com o nome do torneio.

RF10 – Configurar Torneio: esta funcionalidade do *software* deve permitir as seguintes ações:

- Definir as categorias do torneio (podem ser criadas várias categorias, por exemplo: especial, adulto, sub21, sub14, sub10 etc.)
- Vincular jogadores ao torneio (puxando do cadastro de jogadores), indicando a categoria em que vai participar;
- Vincular árbitros ao torneio (puxando do cadastro de árbitros);
- Definir pontuação por partida (vitória, empate, derrota por jogo, derrota por WO);
- Definir sistema de emparceiramento (*Schuring* ou Suíço);
- Definir número de rodadas (quando o sistema de emparceiramento escolhido for Suíço);
- Definir o tempo de jogo (em minutos);
- Definir critérios de desempate (*Buchholz* total, progressivo, vitórias, vitórias com pretas).

RF11 – Excluir Torneio: a exclusão do torneio é necessária para iniciar um novo torneio. Ao excluir torneio, todas as tabelas necessárias para a configuração do torneio devem ter seus registros apagados fisicamente.

Módulo de Condução do Torneio



RF12 – Gerar Emparceiramento de Rodada: para cada rodada deve ser gerado o emparceiramento das partidas. O emparceiramento é o procedimento que define a dupla de oponentes que jogarão a partida. Ao definir os oponentes, o sistema deve:

- ❖ Seguir as regras de emparceiramento do sistema adotado na configuração do torneio (*Schuring* ou Suíço);
- ❖ Indicar o número da mesa em que a partida ocorrerá;
- ❖ Indicar quem joga de brancas (o sistema deve, se possível, alternar a cor das peças dos jogadores ao longo das rodadas);
- ❖ Não permitir que a dupla de oponentes se repita ao longo das rodadas;
- ❖ Quando houver número ímpar de jogadores no torneio fazer rodízio do *BYE* (*BYE* é a rodada de espera para o jogador que fica sem oponente).

Exemplo: Rodada 2

Mesa 1: João X Camila

Mesa 2: Ana X Carlos

Mesa 3: Rodrigo X Sílvio

Mesa 4: Jair X Danilo

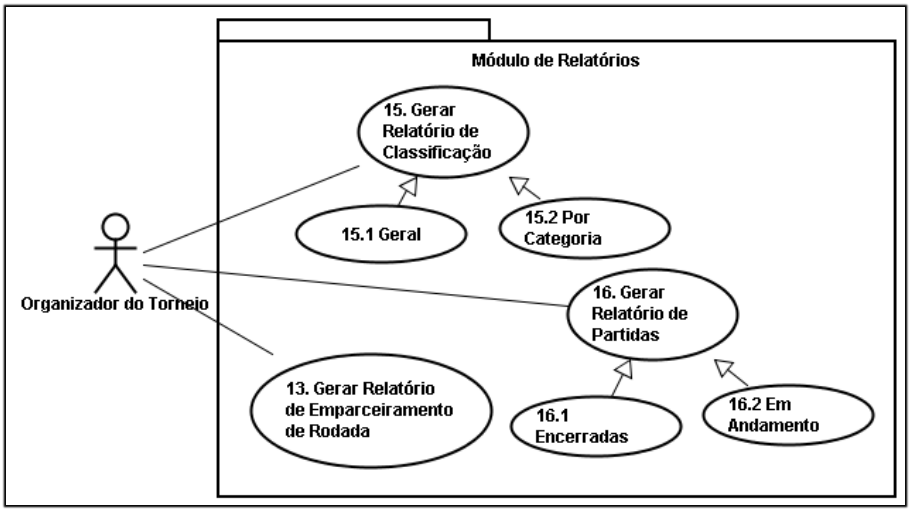
Obs.: o jogador da esquerda joga de brancas.

Os dados apresentados no exemplo devem ser mostrados na tela.

RF13 – Gerar Relatório de Emparceiramento de Rodada: ao executar o RF12, pode-se optar por gerar o relatório de emparceiramento de rodada. Este relatório deve ser gerado no formato .txt, contendo os seguintes dados: Cabeçalho (título do torneio, data e horário, e número da rodada) e partidas que ocorrerão na rodada (no formato apresentado no exemplo do RF12. Adicionar do lado do nome de cada jogador, a pontuação que ele obteve até o momento no torneio). O arquivo .txt gerado deve ter o seguinte nome: RODADA99.TXT, onde 99 indica o número da rodada. Este arquivo deve ficar gravado no diretório com o nome do torneio.

RF14 – Registrar Resultado de Partida: quando uma partida é encerrada, o árbitro preencherá uma ficha de resultado, indicando: o número da rodada, o número da mesa (que equivale ao número da partida naquela rodada) e o vencedor ou empate (no caso de vitória, se houver *WO*). Esta ficha será encaminhada para a organização do torneio, e será com base nela que o resultado da partida será informado ao sistema.

Módulo de Relatórios



RF15.1 – Gerar Relatório de Classificação Geral: este relatório deve ser gerado em formato .txt, indicando a que rodada se refere, e incluindo os jogadores de todas as categorias, contendo: cabeçalho (título do torneio, data e hora, e número da rodada) e corpo do relatório (linhas contendo o nome do jogador, o seu *status,* número de vitórias, empates, derrotas e *WOs*, e pontuação obtida até aquela

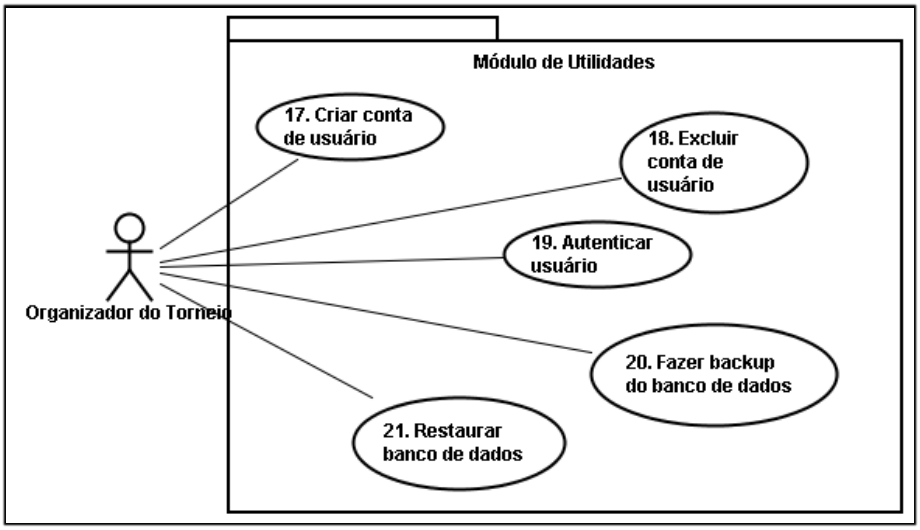
rodada). A lista de jogadores deve ser apresentada em ordem decrescente, de acordo com a pontuação obtida por cada jogador. O arquivo.txt gerado deve ter o seguinte nome: CLASSIFICAÇÃO_GERAL_RODADA99, em que 99 indica até qual rodada se refere aquela classificação. O arquivo deve ser gravado no diretório gerado para o torneio.

RF15.2 – Gerar Relatório de Classificação por Categoria: este relatório deve ser gerado em formato .txt, indicando a que rodada e categoria se refere, contendo: cabeçalho (título do torneio, data e hora, número da rodada e categoria) e corpo do relatório (linhas contendo o nome do jogador, o seu *status,* número de vitórias, empates, derrotas e *WOs*, e pontuação obtida até aquela rodada). A lista de jogadores deve ser apresentada em ordem decrescente, de acordo com a pontuação obtida por cada jogador. O arquivo .txt gerado deve ter o seguinte nome: CLASSIFICAÇÃO_RODADA99_XXXXXX, em que 99 indica até qual rodada se refere aquela classificação e XXXXXX indica a categoria. O relatório de classificação por categoria é importante para torneios que oferecem premiação por categoria. O arquivo deve ser gravado no diretório gerado para o torneio.

RF16.1 – Gerar Relatório de Partidas Encerradas: deve ser informado ao *software* o número da rodada a que se referirá aquele relatório. Após receber o número da rodada, o *software* deve gerar um relatório das partidas que já foram encerradas, e mostrá-las na tela (não será necessário gerar arquivo.txt). Devem ser mostradas as seguintes informações: número da mesa e nome dos jogadores. *SiGTX* considerará uma partida encerrada quando os campos de resultado da partida estiverem preenchidos.

RF16.2 – Gerar Relatório de Partidas em Andamento: deve ser informado ao *software* o número da rodada a que se referirá aquele relatório. Após receber o número da rodada, o *software* deve gerar um relatório das partidas que ainda não foram encerradas, e mostrá-las na tela (não será necessário gerar arquivo .txt). Devem ser mostradas as seguintes informações: número da mesa e nome dos jogadores.

Módulo de Utilidades



RF17 – Criar Conta de Usuário: esta função deve ser oferecida em um arquivo executável separado do restante do *software*. Uma conta deve ser criada com os seguintes dados: RG do usuário (que será seu *login*), nome do usuário e senha (até 10 caracteres: números e letras). Pedir confirmação da senha.

RF18 – Excluir Conta de Usuário: informar o RG, mostrar o nome do usuário correspondente, pedir confirmação de exclusão.

RF19 – Autenticar Usuário: digitar *login* (RG) + senha, e verificar se o usuário está cadastrado. Somente usuários cadastrados terão acesso ao *SiGTX*. Se o usuário for autenticado, mostrar o menu com as opções do software.

RF20 – Fazer *backup* do Banco de Dados: copiar todas as tabelas do banco de dados em uso para uma mídia externa (*pendrive* ou HD externo). Pedir confirmação do *backup*. Este procedimento precisa ser simples para o usuário final, evitando que ocorram erros de operação.

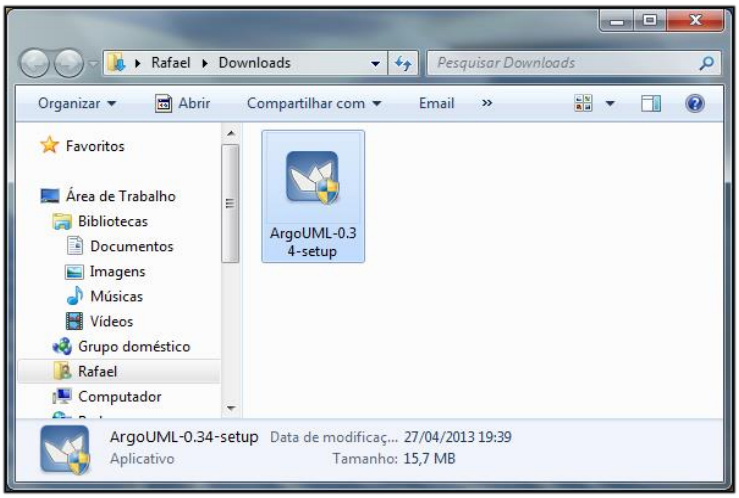
RF21 – Restaurar Banco de Dados: copiar as tabelas da mídia externa (*pendrive* ou HD externo) para o HD principal da máquina em que está rodando o *SiGTX.* Pedir confirmação da restauração do banco de dados. Este procedimento precisa ser simples para o usuário final, evitando que ocorram erros de operação.

APÊNDICE A - Instalação do ArgoUML 0.34

O *ArgoUML* é uma ferramenta totalmente escrita em *Java* e utiliza a fundação de classes desta linguagem de programação, permitindo que ela rode virtualmente em qualquer plataforma com *Java5* ou *Java6*. Para isto, será preciso ter um sistema operacional que suporte *Java*, 10 MB de espaço livre em disco e o JDK *(Java Development Kit)* versão 1.4 ou superior instalado. O JDK 1.4 pode ser obtido através da página da *Sun Microsystems* neste endereço URL: *http://java.sun.com/j2ee/1.4/download-sdk.html.*

Com o JDK 1.4 devidamente instalado, o próximo passo é baixar o *ArgoUML*. Para isso, basta ir até o site *http://argouml.tigris.org/* da organização *Tigris*, grupo que mantêm a ferramenta e apertar no ícone indicado na figura abaixo:

O instalador do programa será baixado. Conforme mostra a figura a seguir:



Após a finalização do *download*, para iniciar o processo de instalação, dê um duplo clique no ícone do instalador. O sistema operacional pedirá permissão para fazer alterações no S*etup*; escolha a opção ‘Sim’.

Antes da inicialização do assistente, é preciso definir o idioma que será apresentado na ferramenta conforme ilustra a imagem abaixo. Escolha um idioma e aperte no botão ‘OK’.

A partir deste momento o processo de instalação é dividido em seis etapas:

1ª etapa: O assistente é inicializado. Para dar continuidade ao processo de instalação aperte no botão ‘Avançar’*;*

2ª etapa: Neste estágio são escolhidos os componentes a serem instalados. O assistente oferece duas opções: JRE *(Java Runtime Environment)* e o *ArgoUML*. Escolha apenas a ferramenta ArgoUML e clique em ‘Avançar’;

3ª etapa: O diretório de instalação do aplicativo é definido nesta fase. O instalador traz a opção *default C:\Program Files(x86)\ArgoUML*. Caso seja necessário, mude-o e selecione a opção ‘Avançar’;

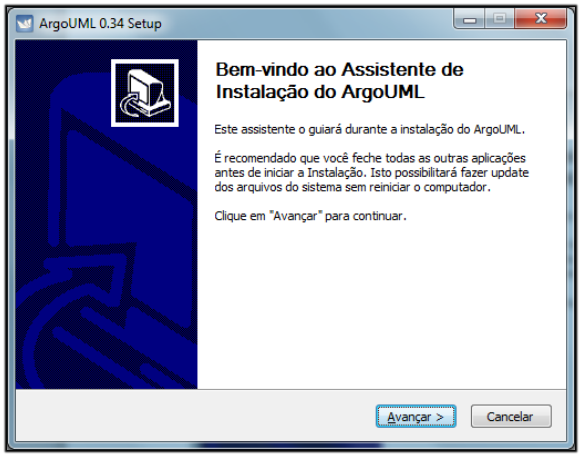
4ª etapa: Após a definição do diretório de instalação. O assistente solicita que seja escolhido uma pasta no Menu Iniciar. Neste caso, escolha a pasta *ArgoUML*. Esta é a última etapa de configuração, inicie o processo de instalação clicando no botão ‘Instalar’;

5ª etapa: Nesta fase são instaladas todas as *DLLs* e aplicativos necessários para que o *ArgoUML* funcione corretamente. A instalação é realizada de forma instantânea;

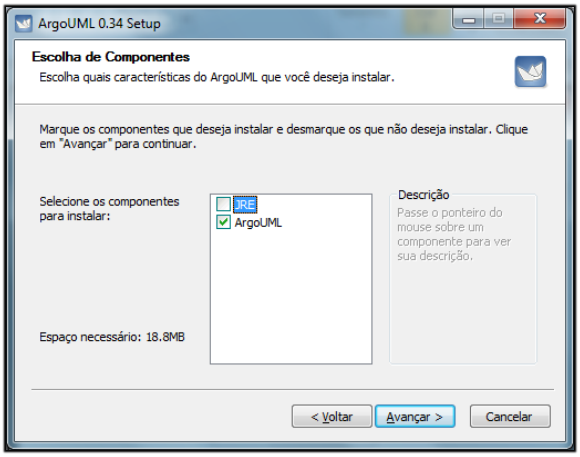
6ª etapa: Por fim, para encerrar a instalação clique no botão ‘Terminar’. Após esta ação a ferramenta *ArgoUML* será executada, estando pronta para uso.

Ilustração das etapas de instalação do ArgoUML 0.34

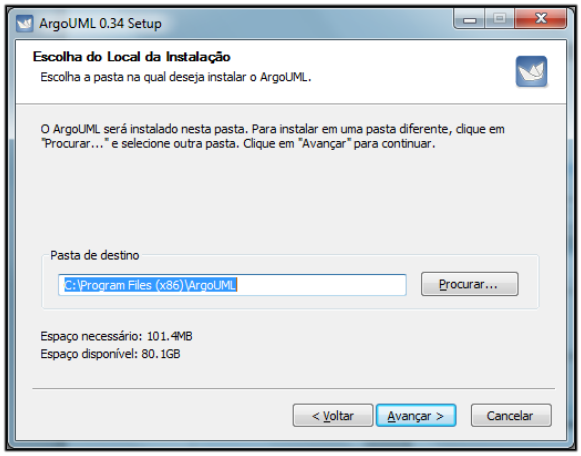
1ª Etapa: Inicialização do assistente



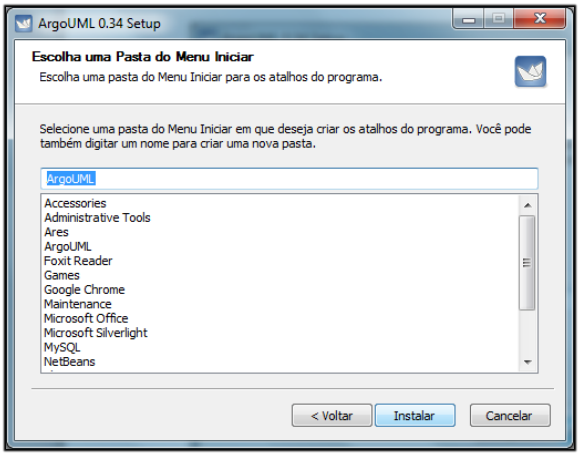
2ª Etapa: Escolha dos componentes



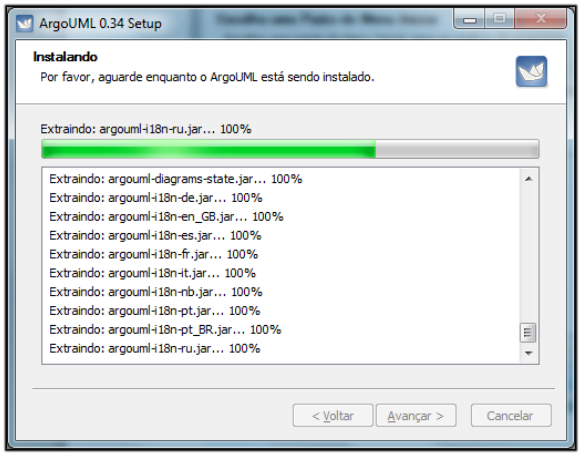
3ª Etapa: Localização da instalação



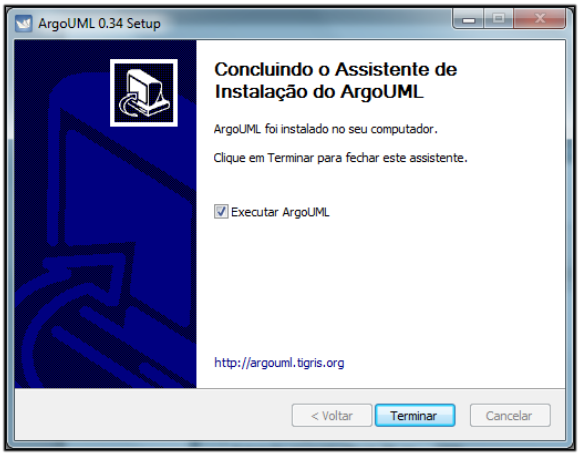
4ª Etapa: Diretório no Menu Iniciar



5ª Etapa: Processo de Instalação



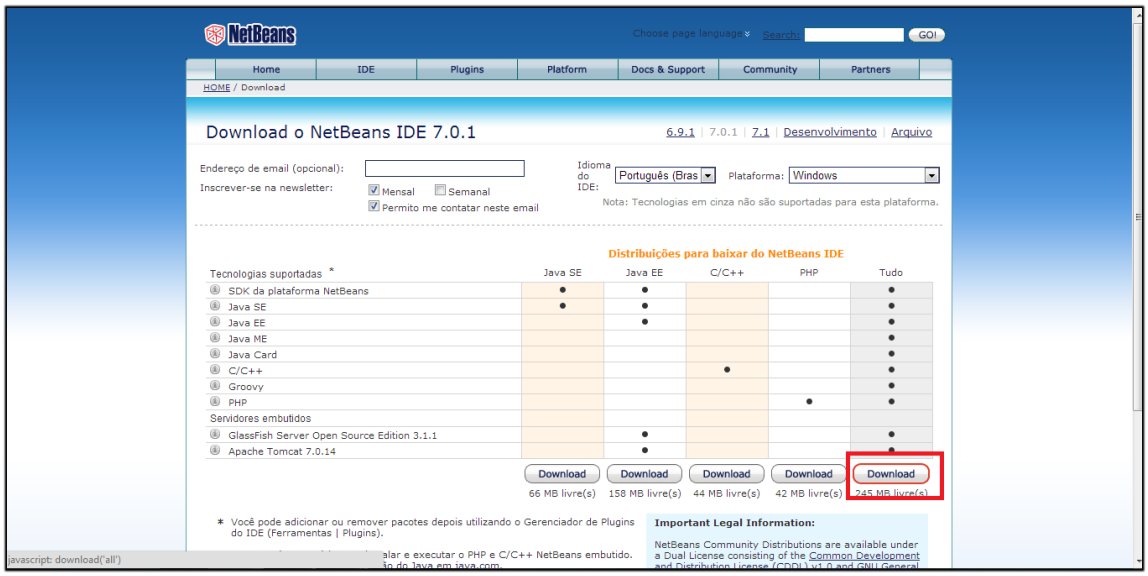
6ª Etapa: Conclusão da instalação



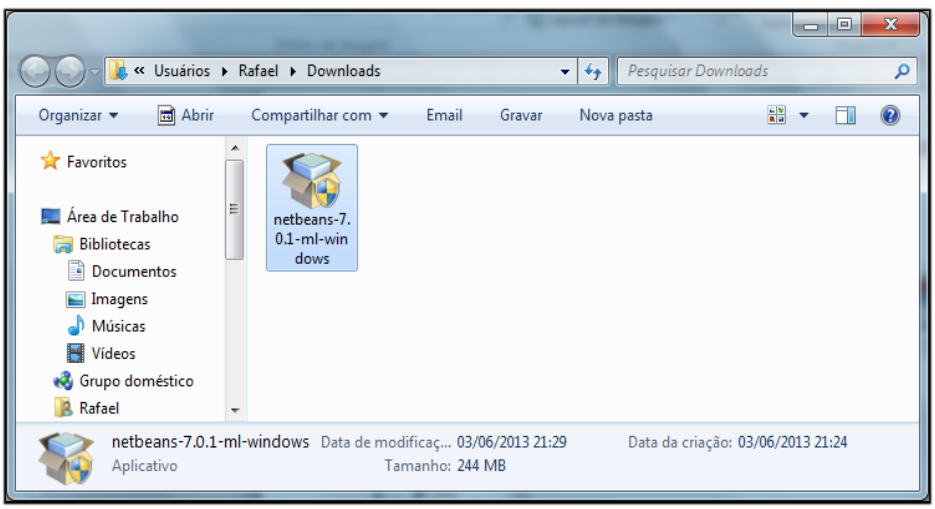
APÊNDICE B - Instalação do NetBeans 7.0 e do plugin NetBeans UML Project

O *NetBeans* fornece um excelente serviço de suporte ao usuário através de seu site. Para consultar quais configurações de *hardware* e *software* utilizar e em quais sistemas operacionais a ferramenta roda, basta ir até este endereço URL: *https://netbeans.org/community/releases/70/relnotes.html#system_requirements.*

Para baixar o aplicativo, vá até este link *https://netbeans.org/downloads/7.0.1/* e selecione a opção destacada na imagem abaixo:



Após a conclusão do *download*. Dê um duplo clique no instalador da ferramenta para iniciar o processo de instalação.



O sistema operacional pedirá permissão para fazer as alterações no *Setup*, é só confirmar a operação clicando no botão ‘Sim’.

O processo de instalação é dividido em oito etapas:

1ª etapa: O assistente é inicializado. Para dar seguimento ao processo, aperte o botão ‘Próximo’;

2ª etapa: Em seguida é apresentado o contrato de licença. Caso concorde com os seus termos selecione o *checkbox* logo abaixo do contrato e aperte o botão ‘Próximo’;

3ª etapa: A terceira fase também apresenta um contrato de licença, porém é do *JUnit*. O procedimento aplicado nesta fase é o mesmo que a anterior;

4ª etapa: Após aceitar os contratos dos aplicativos, é preciso definir a localização de onde a IDE *NetBeans* e o JDK serão instalados. O assistente sugere um diretório *default*, porém ele pode ser mudado. Defina o local e aperte o botão ‘Próximo’;

5ª etapa: O *GlassFish* também é instalado nesta versão. Esta ferramenta é um servidor de aplicação *open source* que suporta todas as APIs do *Java* (Para mais informações, acesse o site do aplicativo *https://glassfish.java.net/*). Assim como na etapa anterior, selecione o diretório e aperte no botão ‘Próximo’;

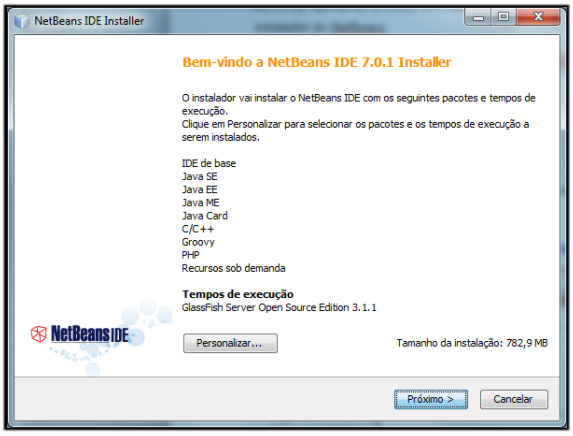
6ª etapa: Neste estágio, inicia-se a instalação propriamente dita. O assistente mostra o resumo das informações definidas nas etapas anteriormente. Para dar continuidade no processo clique no botão ‘Instalar’;

7ª etapa: Após a confirmação, os aplicativos *NetBeans*, *JUnit* e o *GlassFish* são instalados. O processo de instalação tende a demorar alguns minutos.

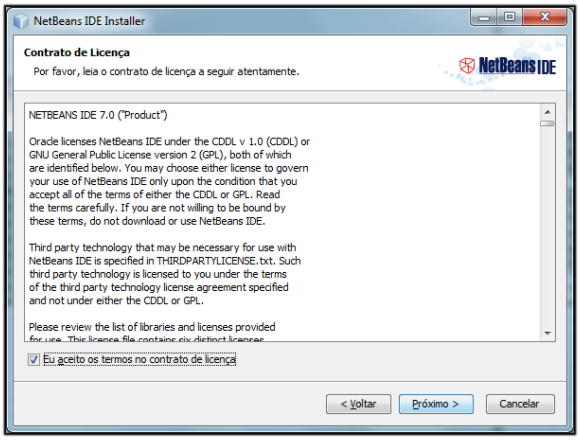
8ª etapa: Por fim, o assistente retorna algumas dicas da ferramenta e pede ao usuário que ajude o *NetBeans* a melhorar fornecendo algumas informações. Caso não queira contribuir, desmarque o *checkbox* da tela e aperte no botão ‘Finalizar’ para utilizar a ferramenta.

Ilustração das etapas de instalação do NetBeans 7.0

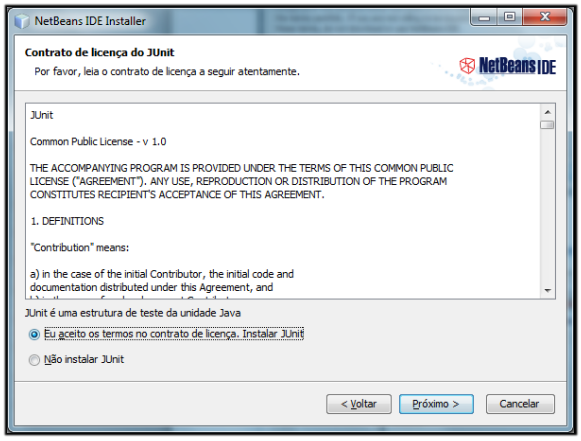
1ª etapa: Inicialização do assistente



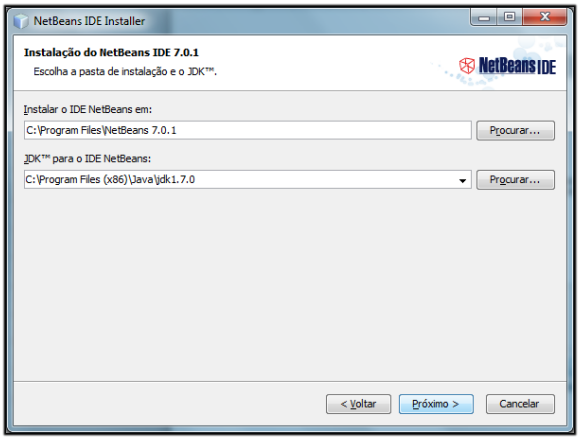
2ª etapa: Contrato de licença NetBeans



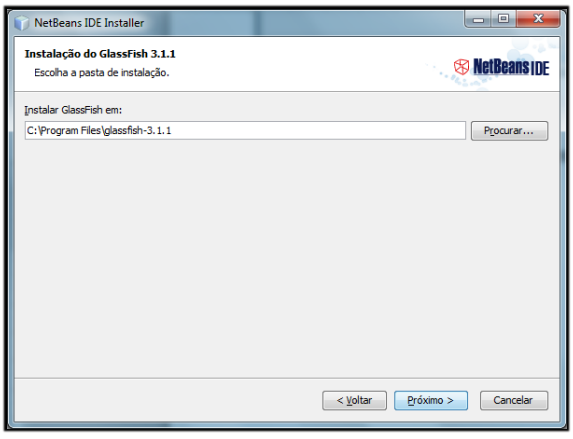
3ª etapa: Contrato de licença JUnit



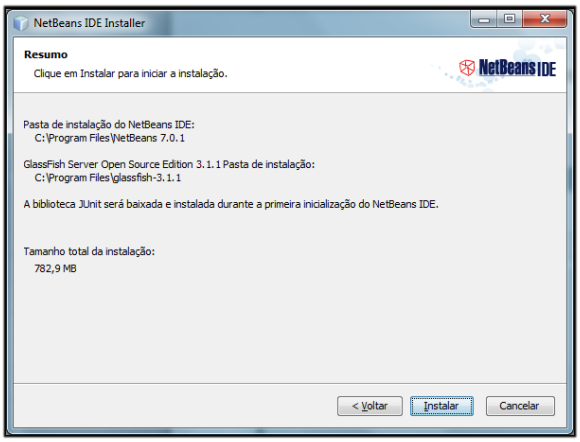
4ª etapa: Diretório do NetBeans e JUnit



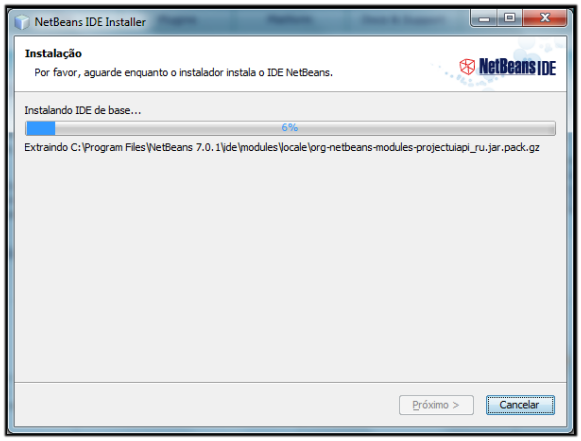
5ª etapa: Diretório do GlassFish



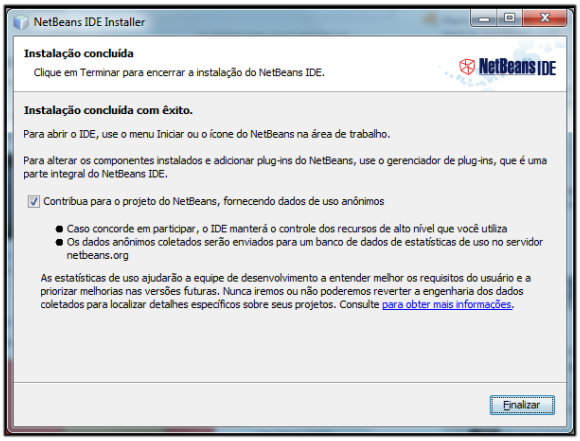
6ª etapa: Resumo da futura instalação



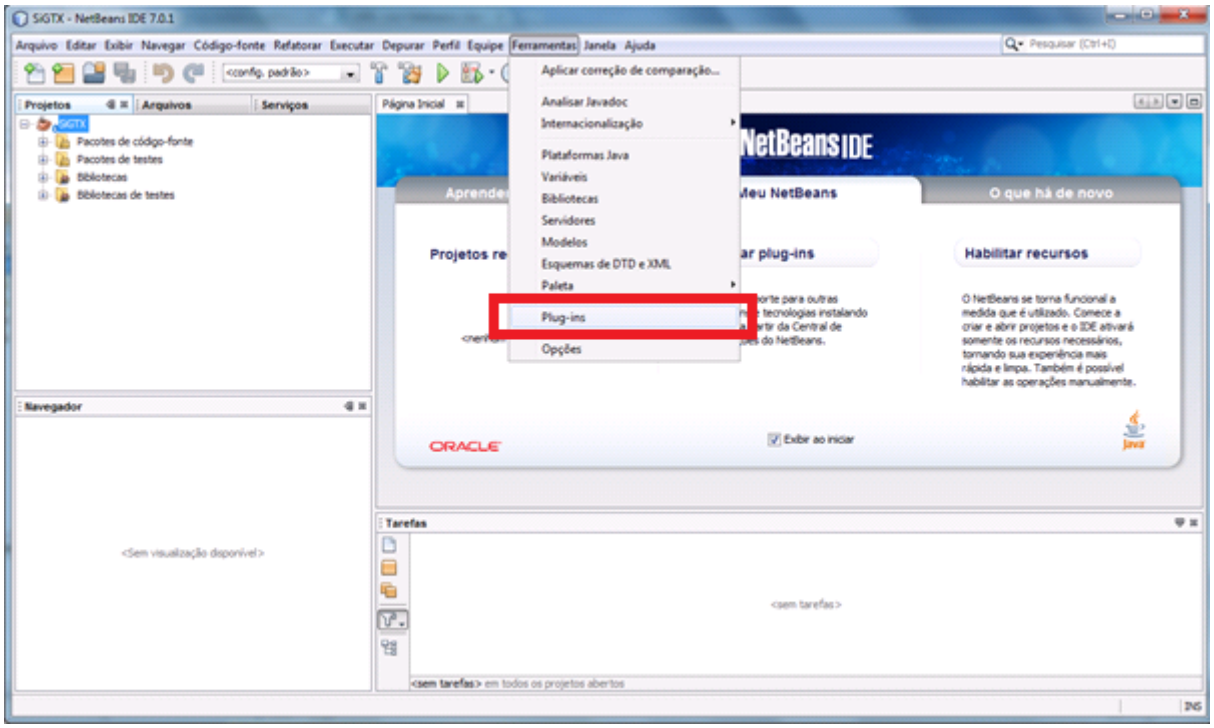
7ª etapa: Processo de instalação



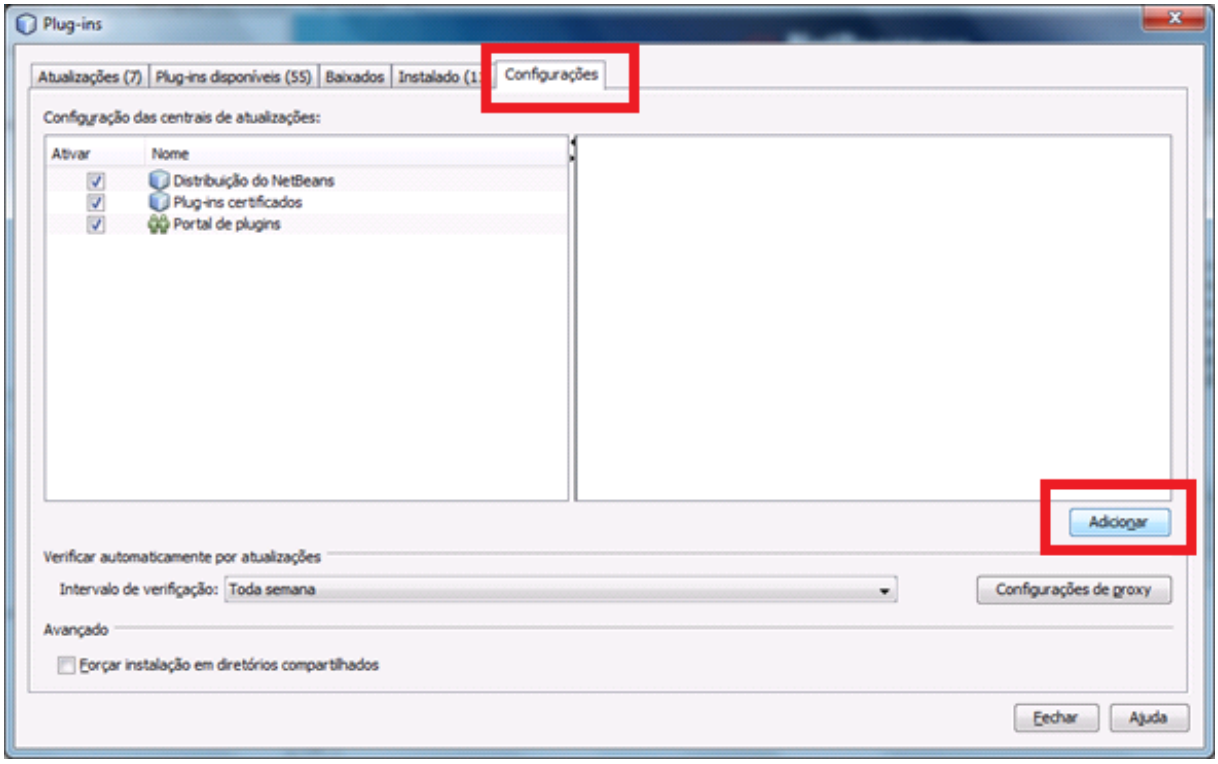
8ª etapa: Conclusão da instalação



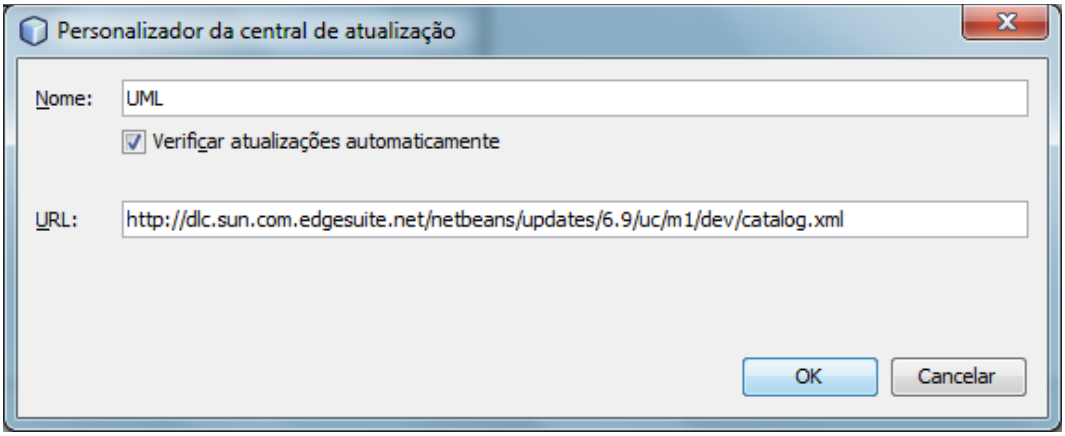
Para instalar o *plugin NetBeans UML Project*, é preciso abrir o *Netbeans*, ir até o menu ‘Ferramentas’ e selecionar a opção ‘*Plug-ins*’ conforme mostra a figura abaixo:



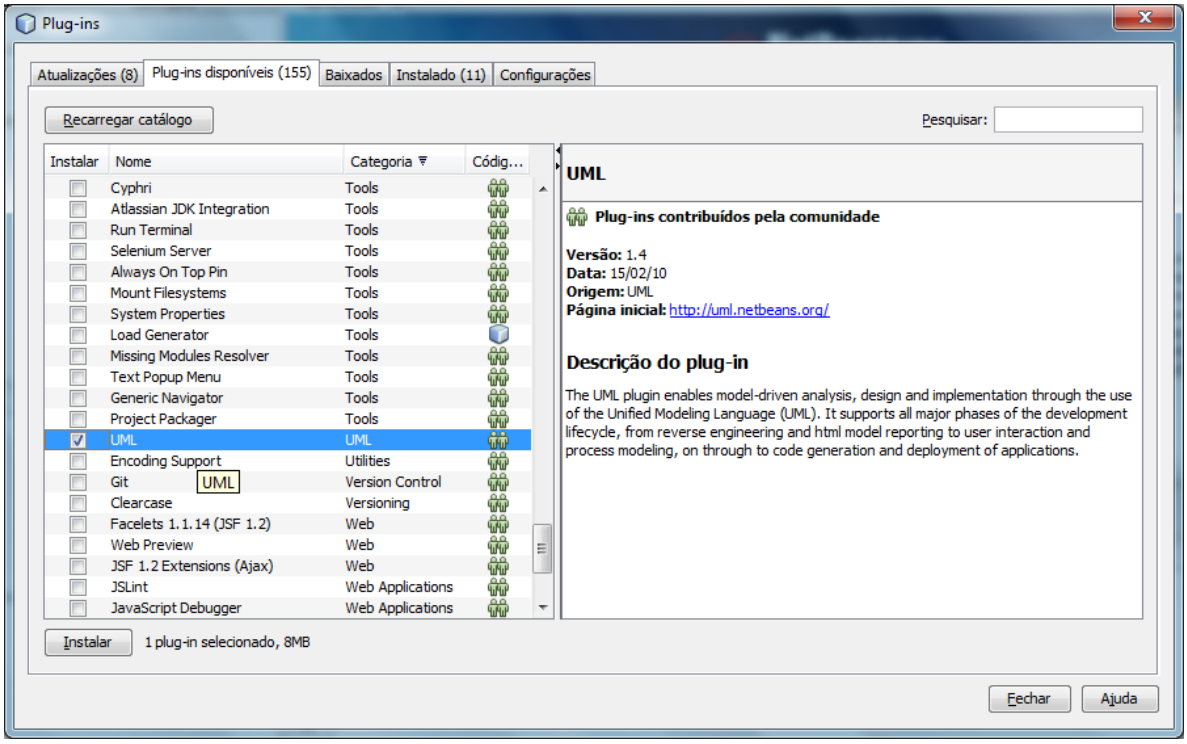
A ferramenta abrirá uma tela de gerenciamentos de *plugins*. Vá até a aba ‘Configurações’ e clique no botão ‘Adicionar’. A imagem abaixo ilustra esta operação:



Uma nova caixa vai aparecer com a seguinte descrição ‘Personalizador da Central de Atualização’. Na janela que abriu, adicione a URL: *http://dlc.sun.com.edgesuite.net/netbeans/updates/6.9/uc/m1/dev/catalog.xml* e dê um nome de sua preferência. No caso abaixo, o nome informado foi UML.



Agora é só clicar na aba ‘*Plug-ins* disponíveis’ e procurar a opção ‘UML’, se não aparecer clique no botão ‘Recarregar Catálogo’.



Procure o *plugin* com o nome ‘UML’, selecione-o e clique em ‘Instalar’. Será aberto um assistente, basta avançar as fases e finalizar o processo de instalação.

Pronto, as ferramentas já estão disponíveis para serem utilizadas.