UNIJUI – UNIVERSIDADE REGIONAL DO NOROESTE DO ESTADO DO RIO GRANDE DO SUL

DETEC – DEPARTAMENTO DE TECNOLOGIA

CURSO DE INFORMÁTICA - SISTEMAS DE INFORMAÇÃO

GABRIELUS: PROTÓTIPO DE UM EDITOR DE DIAGRAMAS E INTERFACES DE EXECUÇÃO PARA UMA FERRAMENTA DE MODELAGEM ARQUITETURAL DE SISTEMAS COMPUTACIONAIS

ELEONOR VINÍCIUS DUDEL MAYER

Santa Rosa, Julho de 2006

UNIJUI – UNIVERSIDADE REGIONAL DO NOROESTE DO ESTADO DO RIO GRANDE DO SUL

DETEC – DEPARTAMENTO DE TECNOLOGIA

CURSO DE INFORMÁTICA - SISTEMAS DE INFORMAÇÃO

GABRIELUS: PROTÓTIPO DE UM EDITOR DE DIAGRAMAS E INTERFACES DE EXECUÇÃO PARA UMA FERRAMENTA DE MODELAGEM ARQUITETURAL DE SISTEMAS COMPUTACIONAIS

ELEONOR VINÍCIUS DUDEL MAYER

Orientador: Prof. André Souza Lemos

Co-Orientador: Prof. Éder Mathias

Trabalho apresentado como requisito à aprovação na disciplina de Laboratório de Sistemas II

Santa Rosa, Julho de 2005

Sumário

1 Introdução

Apresentamos neste relatório, de forma bastante objetiva, os trabalhos realizados na construção do Gabrielus*,* um protótipo de editor de diagramas e interfaces de execução para uma ferramenta de modelagem arquitetural de sistemas computacionais.

Capitulo 2, Projeto Científico, estaremos reprisando sucintamente o que foi tratado no trabalho apresentado como relatório à disciplina de Laboratório de Sistemas I, [Mayer, 2005]. Em suma o relatório apresentado à disciplina de Laboratório de Sistemas I é a fundamentação teórica do presente trabalho e também, grande parte da documentação de implementação do mesmo. Neste mesmo capitulo encontram-se então os objetivos deste trabalho assim como as suas justificativas, sempre lembrando todo o trabalho realizado na disciplina de Laboratório de Sistemas I.

No Capitulo 3, Ambiente Desktop, estaremos apresentando a arquitetura *desktop* montada para o Gabrielus. São comentados os *frameworks* que foram utilizados para montas esta dita arquitetura do software e, qual o propósito da adoção da cada um deles. Também é mostrada e comentada a arquitetura do próprio Gabrielus, a estruturas dos pacotes Java que o compõe, qual o propósito de cada um dos pacotes assim como a dependência que existe entre os pacotes, tanto os definidos para a arquitetura do Gabrileus quando a dos pacotes do Gabrielus com os pacotes dos *frameworks* utilizados.

Quando ao Ambiente *Web*, tratado no Capitulo 4, diz respeito a arquitetura que envolve a disponibilização do Gabrielus no *site* deste projeto e também a arquitetura do *site* em si. Quais as tecnologias que estão envolvidas no desenvolvimento do *site*, quais os *frameworks* adotados para o

desenvolvimento do mesmo, quais as funcionalidades desejadas, entre outras curiosidades. Já, quando a disponibilização do Gabrielus no site, trataremos especificamente de *Java Web Start* e, mostramos passo a passo como é que se deu o trabalho de disponibilizar o Gabrielus no *site* do projeto.

E por fim, na conclusão, Capitulo 5, apresentamos uma breve análise do trabalho que realizamos e, sugestões de trabalhos que ainda poderão/virão a ser desenvolvidos, assim como os trabalhos que desejariamos ter desenvolvidos e que por falta de tempo habil não foram realizados. Quanto a estes trabalhos futuros e/ou tarefas pendentes, como forma de orientação apresentamos uma série de alternativas e sugerimos alternativas ideais. E assim, apresentamos o possível do trabalho realizado no projeto Gabrielus.

2 Projeto Científico

Para além do espectro de aplicações mais sofisticadas, a proposta das linguagens de programação visual *DataFlow* pode ser uma alternativa para a introdução de noções básicas de ciência da computação, tanto para o público em geral – inclusive no espaço da escola – como para o público universitário, e até mesmo em cursos de informática ou computação.

Sem ter a intenção de pôr diretamente em questão a realidade conjuntural, o presente trabalho pretende buscar um espaço novo para a aplicação da concepção de sistemas *DataFlow*, que é o espaço da educação e do desenvolvimento de uma cultura informática mais diversificada. Lembramos que, de certo modo, uma máquina de von Neumann pode ser pensada como um caso particular de arquitetura de fluxo de dados, de modo que nosso trabalho não vai na direção de propor necessariamente uma alternativa à tradição do ensino de arquiteturas de computação, mas de propor-lhe um outro ponto de partida. Idealmente, este novo ponto de partida pode significar uma ampliação do público alvo, caso se perceba que, sob esta nova forma, a própria noção de arquitetura de computadores torna-se mais acessível ao público leigo interessado.

2.1 Justificativas

A hipótese que orienta este trabalho, é a de que um conceito de computação é também um conceito de utilização de computadores [Nardi, 1993]. Se um dia for outra a idéia de programa, será também

outra a idéia de programador, e também a de usuário. No que diz respeito à adoção da computação como parte do currículo escolar, por exemplo, uma série de escolhas estão ainda por serem feitas. Elas dependem do modo como se disponha da computação – teoria e prática – na cultura humana como um todo, não decorrem apenas do que se pensa nos setores de aplicação mais tradicionais da computação, voltados para o processamento de dados (tecnologia da informação), a automação e a computação científica.

Apesar do que sugerem os manuais de técnicas de programação, um programador de computadores faz uso constante da sua experiência. Ao projetar um sistema de informações complexo, o engenheiro de software se assemelha mais a um médico-cirurgião, e menos a um engenheiro eletricista: práticas adequadas são tão importantes quanto princípios adequados. Talvez seja por isso que o grande avanço na programação orientada a objetos sobreveio, não quando os seus princípios teóricos foram concebidos, mas bem depois, quando padrões de projeto, de codificação e de implementação foram impostos como um conjunto coeso e consistente de práticas a serem seguidas.

Sendo assim, justifica-se uma busca de padrões de programação que sejam compatíveis com o universo do usuário leigo. Não seriam estes padrões uma preparação para o exercício profissional da programação, mas também não pertenceriam a um mundo separado. Sabe-se hoje que o conhecimento médico, por mais sofisticado que seja, deve estar em certa medida acessível ao leigo, sem o que não há possibilidade de que decisões importantes, que cabem somente aos pacientes e seus familiares, sejam tomadas à luz da razão. Da mesma maneira, é importante que o conhecimento da ciência da computação e da engenharia de software não seja um mistério para o usuário de computadores. Infelizmente, trata-se de uma tecnologia que se diferencia justamente por produzir mistérios com muita facilidade.

2.2 Objetivos

Temos como objetivos, portanto, a definição da arquitetura de desenvolvimento do protótipo, implementar o protótipo propriamente dito e, dentro do possível, implementar também alguns, um ou mais, objetos (os tokens, o repositório, a interface, dentre os demais os mais simples) para que possamos comprovar a robustes e simplicidade da arquitetura definida, assim como também validar

a implementação do editor em si.

Em seguinda, pretendemos integrar o trabalho desenvolvido em trabalho apresentado a disciplina de Laboratório de Sistemas I (núcleo de ambiente de programação visual *DataFlow* projetado) ao editor desenvolvido e, também, já começar com os trabalhos que dizem respeito as animações dos diagramas, ou seja, a execução dos possíveis fluxos de dados entre os objetos componentes de um diagrama proposto.

Entendemos como núcleo de um ambiente de programação visual *DataFlow* a parte de um sistema responsável pelo controle dos possíveis fluxos de dados entre os objetos componentes do próprio sistema e, necessariamente, o devido protocolo de comunicação.

O modelo de programação adotado busca a máxima simplicidade. São utilizadas construções cujo sentido é óbvio pelo contexto, não requerendo, por parte do usuário, um conhecimento prévio específico de sistemas computacionais, seja do ponto de vista da organização, seja do ponto de vista da arquitetura. Neste sentido, estão excluídos do modelo, por exemplo, conceitos sofisticados de sincronização e controle de fluxo, que tornariam o modelo aplicável ao desenvolvimento tecnológico em sentido estrito.

Em fim, este trabalho apresenta o protótipo de um ambiente de programação que, partindo de uma notação diagramática dada e apresentada, permitira a representação de sistemas computacionais de forma puramente arquitetural.

2.3 Projeto do Núcleo

O projeto de um núcleo de ambiente de programação visual *dataflow* levou em consideração a existência de um editor diagramático suficientemente flexível que pudesse aliar visibilidade, acessibilidade e eficiência ao projeto arquitetural. Fez-se necessário contar com uma representação interna, que faz o papel de interface entre a representação visual dos diagramas e o componente editor que produz o funcionamento propriamente dito dos sistemas e, este é o mecanismo que estamos chamando de núcleo de ambiente de programação visual *dataflow* [Mayer, 2005].

Dividimos o projeto do núcleo em módulos: a) o módulo do Protocolo de Comunicação entre Objetos, responsável pela dinâmica de conversação entre os objetos; b) o módulo do Esquema *Multithreadig* do Sistema: responsável por todo o processamento do módulo do protocolo; c) o módulo de Modelo de Objetos: onde organizamos toda a hierarquia de classes do sistema e; d) o módulo dos Objetos: onde, finalmente, apresentamos os objetos como componentes do núcleo deste ambiente de programação visual.

2.3.1 Protocolo de Comunicação entre Objetos

O módulo do protocolo de comunicação entre objetos define, portanto, de que forma se realizará a comunicação entre os objetos ativos do sistema. A comunicação que desejamos que aconteça entre estes objetos ativos nada mais é do que a transferência do conteúdo de um objeto para outro. Os componentes que estão sendo considerados ativos, portanto, são o Repositório e a Porta de entrada; o repositório porque pode tanto querer transferir um objeto *Token* como também receber objetos e; a Porta de Entrada, pois, embora não seja capaz de receber objetos, estará gerando objetos e, consequentemente, querendo transferi-los. Então, estamos entendendo por objeto ativo todo aquele componente do sistema que é capaz de iniciar uma nova fase de transferência de *tokens* entre os objetos do sistema, ou seja, queira fazer com que este conteúdo, o *Token*, flua, ou continue fluindo por entre os objetos constituintes do sistema.

Baseamos o nosso protocolo de comunicação em um protocolo muito simples, o modelo de solicitação/resposta (*request/response*). Este protocolo é a idéia que há por trás das estruturas mais comuns de sistemas distribuídos do tipo cliente-servidor. O cliente envia uma mensagem ao servidor solicitando algum tipo de serviço, como, no nosso caso, uma requisição de transferência; o servidor faz o trabalho e envia para o cliente os dados solicitados. Podemos então fazer uma analogia desta dinâmica com os nossos componentes onde, o cliente seria um objeto, um Repositório, por exemplo, e o servidor poderia ser qualquer outro objeto, inclusive um Repositório. E isso é característico desse tipo de estrutura de serviço, pois tanto o cliente quanto o servidor mantêm o mesmo tipo de implementação e, portanto, tanto os clientes quanto os servidores podem executar processos como clientes.

2.3.2 Esquema Multithreading do Núcleo

O esquema *multithreading* do núcleo do protótipo de ambiente de programação visual foi modelado da seguinte maneira: a) uma classe abstrata `TThread`, que define a estrutura responsável por manter todos os dados pertinentes ao processamento de transferências; b) uma classe `TDataFlow`, que será a responsável por disparar e manter todos os processamentos gerados pelos objetos ditos ativos (Repositórios e Portas de Entrada); c) `TPreRequest`: responsável pela manutenção dos dados relacionados ao processamento de um pedido de pré-requisição; d) `TRequest`: responsável pela manutenção dos dados relacionados ao processamento de um pedido de requisição e; e) `TTransfer`: a classe que será responsável pelo gerenciamento de transferência de *Tokens* entre objetos.

2.3.3 Modelo de Objetos e Objetos

O módulo de modelo de objeto é aquele módulo que faz a definição de todas as classes que se encontram um nível acima daquelas classes que estamos nos referindo como "folhas", ou seja, aqueles que representam de fato a implementação de um componente do núcleo, como por exemplo, um repositório, ou ainda um multiplicador.

Já o módulo dos objetos estará apresentando as especificações das classes que definem aqueles objetos que serão, na real, os componentes do núcleo, ou seja, aquelas classes que estarão implementando as funcionalidades esperadas por cada uma das formas apresentadas pela notação diagramática.

2.3.4 Considerações

Partindo da análise dos objetivos que nos propomos a alcançar no trabalho desenvolvido na disciplina de Laboratório de Sistemas I, com o que foi então desenvolvido e estamos agora fazendo uso, devemos dizer que o trabalho desenvolvido na elaboração do projeto do núcleo do ambiente de programação visual *dataflow*, e a sua implementação, foram de fato bastante importantes.

Tanto que, de todo o projeto que foi desenvolvido e, da parte que diz respeito a

implementação dos componentes do núcleo, módulos como o do Esquema de Multithreading, assim como o módulo do Protocolo de Comunicação foram cem por cento aproveitados. Apenas os módulos de Modelo de Objetos e o módulo de Objetos é que, devida a arquitetura definida para o protótipo, que não conseguiram ser aproveitadas.

3 Ambiente Desktop

Nas primeiras versões do Java a única forma de fazer programas gráficos era através da AWT[1], uma biblioteca de baixo-nível que dependia de código nativo da plataforma onde rodava. Ela traz alguns problemas de compatibilidade entre as plataformas, fazendo que nem sempre o programa tinha aparência desejada em todos os sistemas operacionais, sendo também mais difícil de ser usada. Para suprir as necessidades cada vez mais freqüentes de uma API[2] mais estável e fácil de usar, o Swing[3] foi criado como uma extensão do Java a partir da versão 1.2.. Swing fornece componentes de mais alto nível, possibilitando assim uma melhor compatibilidade entre os vários sistemas onde Java roda. Ao contrário da AWT, Swing não contém uma única linha de código nativo, e permite que as aplicações tenham diferentes tipos de visuais (em inglês, *skins*), os chamados *"Look and Feel"*. Já com AWT isso não é possível, tendo todos os programas a aparência da plataforma onde estão rodando [Steil, 2006].

As vantagens do Swing não param por aí. Há uma enorme gama de controles extras disponíveis, tais como áreas de texto que nativamente podem mostrar conteúdo RTF[4] ou HTML[5],

1 Site do AWT: http://java.sun.com/products/jdk/awt/

2 API, de Application Programming Interface (ou Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por programas aplicativos -- isto é: programas que não querem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços. De modo geral, a API é composta por uma série de funções acessíveis somente por programação, e que permitem utilizar características do software menos evidentes ao usuário tradicional.

3 Site do Swing: http://java.sun.com/docs/books/tutorial/uiswing/

4 O Rich Text Format (RTF) é um formato de arquivo proprietário desenvolvido e mantido pela Microsoft deste 1987 para o intercâmbio de documentos em multiplataformas. Muitos dos processadores de texto são capazes de ler e escrever documentos do tipo RTF.

5 A sigla HTML deriva da expressão inglesa HyperText Markup Language (Linguagem de Formatação de Hipertexto). Trata-se de uma linguagem de marcação utilizada para produzir páginas na Internet. De modo geral são documentos de texto escritos em códigos que podem ser interpretados pelos browsers para exibir as páginas da

botões com suporte a imagens, *sliders*, selecionadores de cores, entre outros. É também possível alterar o tipo de borda para a maior parte dos componentes, todos podem ter imagens associadas e é possível até ter controle de como são desenhados os mínimos detalhes de apresentação.

Pretendíamos então integrar ao núcleo do ambiente de programação visual *dataflow* projetado e quase que em sua totalidade implementado em trabalho apresentado à disciplina de Laboratório I [Mayer, 2005], àquela parte do sistema responsável pelo representação visual e também de execução dos possíveis fluxos de dados entre os objetos componentes de um diagrama proposto. Para tanto que, usando a tecnologia Swing que nos é dispota pelo Java que desenvolvermos o Gabrielus.

Abaixo segue Diagrama de Componentes representando o conjunto de componentes e respectivas dependências para o Gabrielus.

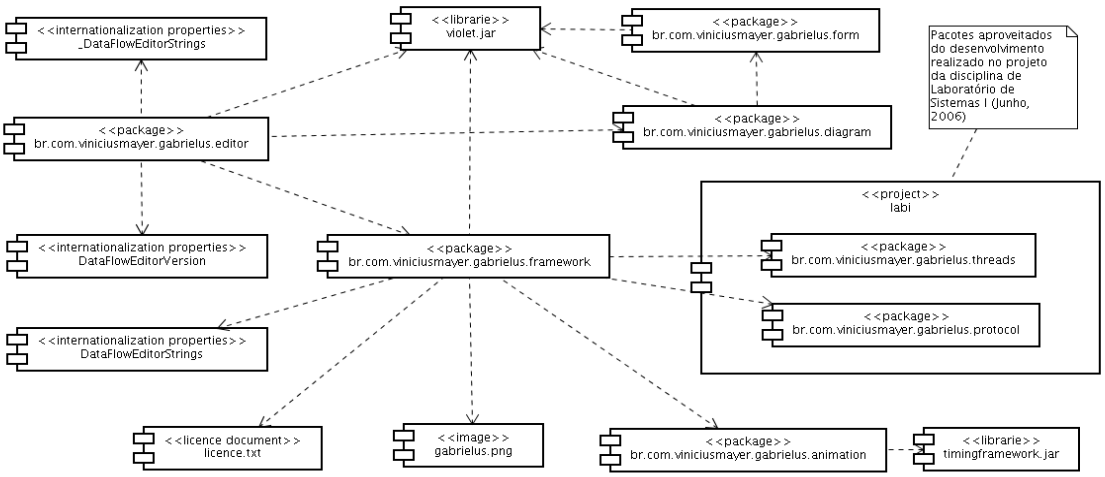

Diagrama 01. Diagrama de Componentes representando o conjunto de componentes e respectivas dependências para o Gabrielus.

No diagrama apresentado acima, estão sendo exibidas as dependências que existem entre os componentes do Gabrielus e, claro, quais as partes/componentes que compõe o Gabrielus. Poderíamos iniciar uma análise deste diagrama a partir do pacote (em inglês: *package)* `br.com.viniciusmayer.gabrielus.editor`, responsável pela construção do aplicativo em si. Este pacote tem dependência de dois arquivos de propriedades, de dois outros pacotes,

World Wide Web.

`br.com.viniciusmayer.gabrielus.framework` e `br.com.viniciusmayer.gabrielus.diagram` e do *framework* `violet.jar`.

Estaremos, no entanto, primeiramente apresentando os *frameworks*, que estão presentes no nosso aplicativo e, em seguida, uma análise dos pacotes que compõe o Gabrielus. E dessa forma, ao passo que apresentamos os pacotes que compõe o Gabrielus, nos propomos a fazer a explanação do diagrama de componentes apresentado acima.

3.1 Frameworks

No desenvolvimento de software, um *framework* é uma estrutura de suporte definida em que um outro projeto de *software* pode ser organizado e desenvolvido. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de *script* e outros softwares para ajudar a desenvolver e juntar diferentes componentes do seu projeto.

Especificamente em orientação a objeto, *framework* é um conjunto de classes com objetivo de reutilização de um *design*, provendo um guia para uma solução de arquitetura em um domínio específico de software. *Framework* se diferencia de uma simples biblioteca (em inglês, *toolkit*), pois esta se concentra apenas em oferecer implementação de funcionalidades, sem definir a reutilização de uma solução de arquitetura (em inglês, *design*).

Utiliza-se o conceito de *Framework* - tecnologia que oferece aos desenvolvedores um veículo para reuso, além de uma forma de capturar a essência de padrões - com o objetivo de aumentar a qualidade e a produtividade no desenvolvimento.

O Gabrielus faz uso de dois *frameworks*, o Viotel e o TimingFramework. O primeiro, Violet, é o framework que utilizamos para montar o editor propriamente dito e está relacionada com processamentos básicos de qualquer aplicativo, como por exemplo o de salvar, abrir, imprimir, entre outros. Já o segundo, TimingFramework, é o *framework* que adotamos para a implementação das animações dos diagramas. Ambos os *frameworks* são apresentados a seguir e, mais tarde, mostramos onde cada um deles aparece na implementação do Gabrielus.

3.1.1 Violet

O Violet é um editor UML[6] desenvolvido por Cay Horstmann[7], e que nos trás as seguintes características:

- É um editor de UML simples de aprender e usar;
- Desenha *nice-looking* diagramas de classe, sequencia, estado e use-case;
- É completamente livre, incluindo os fontes que são distribuídos sob a licença GNU General Public License[8] e;
- É multiplataforma.

O editor, em si, foi desenvolvido pensando nos alunos, professores e/ou autores que precisam desenvolver um simples digrama UML rapidamente e que não necessariamente pretende, como ferramenta, se desenvolver ao ponto de ser um produto comercializável, que venha a concorrer com as ferramentas que hoje existem no mercado e que trazem inúmeras funcionalidades além das que o Violet disponibiliza. Exemplo de ferramentas neste sentido seriam, por exemplo, o Rational Rose[9], Together[10] e o ArgoUML[11], que é comercializado com o nome de Poseidon[12]. Estes programas, embora considerados importantes ferramentas para a serem usadas com editores UML, não caem na graça de estudantes e usuários casuais porque são muito lentos, caros e difíceis de aprender e usar. Estas aplicações trazem funcionalidades do tipo:

- Geração de código. O Violet não gera código algum a partir dos diagramas UML criados;
- Engenharia reversa. O Violet não gera diagramas UML a partir de códigos fonte;
- Checagem da semântica dos modelos. Você poderá desenhar diagramas UML contraditórios;
- Importação e exportação de/em arquivos do tipo XMI[13]. O Violet não gera arquivos do tipo XMI que podem ser importandos para outras ferramenteas UML, nem poder

6 A Unified Modeling Language (UML) é uma linguagem de modelagem não proprietária de terceira geração. A UML não é um método de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como desenhar seu sistema, mas ele lhe auxilia a visualizar seu desenho e a comunicação entre objetos.
7 Site do Cay Horstmann: http://www.horstmann.com/; Cay Horstmann também é autor dos livros Core Java 2, Volume I – Fundamentals, Core Java 2, Volume 2 – Advanced Features; Core JavaServer Faces, entre outros;
8 Site da Licença GNU General Public Licence: http://www.gnu.org/copyleft/gpl.html
9 Site do RationalRose: http://www.rational.com/products/rose/index.jsp
10 Site do Together: http://www.borland.com/together/
11 Site do ArgoUML: http://argouml.tigris.org/
12 Site do Poseidon: http://www.gentleware.com/
13 XMI (ou XML Metadata Interchange) é um grupo de gerenciamento de objetos do XML. O XMI permite a troca facilitada de metadata entre as ferramentas de modelação (baseadas em OMG-UML) e os repositórios (OMG-MOF).

ler modelos desenhados em outras ferramentas UML.

O livro "*Object-Oriented Design & Patterns*" [Horstmann, 2003], traz uma discussão com relação a padrões de projeto que veio a delinear o *framework* do editor gráfico do Violet. A razão para tamanho cuidado com relação aos padrões de projeto se devem ao fato de querer manter o aplicação escrita da forma mais simples possível e com um número mínimo de arquivos de códigos fonte.

3.1.2 TimmingFramework

O Timing Framework, é um framework desenvolvido para tornar mais fácil a implementação de animações em Java com controles baseados em tempo. O framework é dividida em três partes lógicas, que constituem os seguintes pacotes:

- *Fundamentals*: `org.jdesktop.animation.timing`;
- *Interpolation*: `org.jdesktop.animation.timing.interpolation`;
- *Triggers*: `org.jdesktop.animation.timing.triggers`;

O pacote *Fundamentals* é descrito detalhadamente no artigo "*Timing is Everything*", que você poderá encontrar próprio *site* do projeto TimingFramework[14]. As classes deste pacote se comportam como a base do objeto `TimingController`, objeto este que atualmente roda, literalmente, a animação. `Cycle`/`Envelope` são as classes que mantem as especificações da animação, as *Interfaces* `TimingTarget` e `TimingListener` são então usadas para receber os eventos de tempo, e a classe `TimingEvent` que especifica os eventos que o `TimingController` produz. Já o pacote *Interpolation*, cujos conceitos principais são transcritos em seguida, é explicado detalhadamente no artigo "*Time Again",* que você pode acessar também no site do projeto TimingFramework[15]. Eis os principais conceitos relacionados ao pacote *Interpolation* do framework TimingFramework:

- *Property Setters*: Fácil delegação das propriedades da animação do *framework* (com por exemplo trocar o tamanho do componente relacionado ao tempo). Esta capacidade é dada a partir da combinação das classes `PropertyRange` e da classe `ObjectModifier;`

14 Site do artigo "Timing is Everything": http://today.java.net/pub/a/today/2005/02/15/timing.html
15 Site do artigo "Time Again": http://today.java.net/pub/a/today/2006/03/16/time-again.html

- *KeyFrames*: Permite a construção de animações mais poderosas e complexas a partir de múltiplos passos seqüências por meio de pares de tempo e valor;
- `KeyFrames` combina objetos `KeyTimes`, `KeyValues`, `KeySplines`, e `KeyFrames.InterpolationType`;
- *Non-Linear Interpolation*: Animações mais interessantes e úteis a partir da especificações de pares de valores de *interpolation.* Esta capacidade é controlada através do uso da classe `KeySplines`, que contem uma ou mais estruturas `Spline;`
- *Acceleration/deceleration:* Fácil especificação de simples comportamentos não lineares a partir da declaração de períodos `TimingController` de aceleração e desaceleração durante a fração de tempo normal.

O pacote *Triggers* contem um fácil suporte ao iniciar/parar das animações, tudo baseado em eventos. No pacote são basicamente encapsulados *EventListener* que simplificam substancialmente as coisas. Ao invés de escrever vários *listeners* para monitornar os eventos e disparar as animações apropriadamente no seu código, você poderá setar *Triggers* as tarefas dos *listeners* e iniciar/parar as animações para você. A intenção é tornar isso fácil para termos canonical effects nos componentes da interface gráfica, como por exemplo botões pulsantes, componentes que são animados num instante específico, entre outros.

O projeto do TimingFramework é plugável ao NetBeans[16], uma vez que o próprio desenvolvimento do TimingFramework é feito nesta ferramenta. Para que os arquivos do projeto sejam interpretados corretamente, sugere-se que se use o NetBeans versão 5.0, o que não impede, na verdade, que se use uma versão maior da ferramenta. É importante salientar que o TimingFramework exige a versão 5.0 ou superior do JDK, e isso se deve ao fato de que em seu código fonte são usados recursos da tecnologia Generics[17] do Java que, de uma forma bastante interessante, nos dispõe uma outra forma, mais enjuta, de resgatarmos diferentes `PropertyRange` and `KeyValues` associados a uma animação qualquer que esteja fazendo uso deste recursos.

16 Site do NetBeans: http://www.netbeans.org/index_pt.html
17 Site do Generics: http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html

3.2 Packages

No desenvolvimento de pequenas atividades ou aplicações, é viável manter o código e suas classes no diretório corrente. No entanto, para grandes aplicações é preciso organizar as classes de maneira a evitar problemas com nomes duplicados de classes, e localizar o código da classe de forma eficiente.

Em Java, a solução para esse problema está na organização de classes e interfaces em pacotes. Essencialmente, uma classe `Xyz` que pertence a um pacote `nome.do.pacote` tem o nome completo `nome.do.pacote.Xyz` e o compilador Java espera encontrar o arquivo `Xyz.class` em um subdiretório `nome/do/pacote`. Este, por sua vez, deve estar localizado sob um dos diretórios especificados na variável de ambiente `CLASSPATH`.

Tipicamente um pacote em Java contém as declarações de várias classes relacionadas. Essas classes são armazenadas em várias unidades de compilação (arquivos do sistema operacional). Cada uma destas unidades de compilação deve ter no início a seguinte declaração:

```
package pacote;
```

Onde `pacote` é o nome do pacote ao qual a unidade pertence. Além disso, todas esta unidades devem estar localizadas no mesmo diretório do sistema operacional. Note então a forte ligação entre o conceito de diretório e o conceito de pacote em Java; em resumo, cada pacote é representado por um diretório contendo vários arquivos com o mesmo cabeçário, indicando o nome do pacote, onde cada arquivo define uma ou mais classes que fazem parte do pacote.

Além disso, na implementação atual de Java, o nome de um pacote deve ser parte do nome do seu diretório associado. Usando o sistema UNIX[18] como exemplo, podemos definir um pacote chamado `nome.do.pacote` contendo dois arquivos que estão no diretório:

```
/home/viniciusmayer/projects/example/nome/do/pacote
```

18 UNIX é um sistema operativo (ou sistema operacional) portátil, multitarefa e multiutilisador originalmente criado por um grupo de programadores da AT&T e dos Bell Labs. Atualmente, UNIX (ou *NIX) é o nome dado a uma grande família de Sistemas Operativos que partilham muitos dos conceitos dos Sistemas UNIX originais, sendo todos eles desenvolvidos em torno de *standards* como o POSIX (Portable Operating System Interface) e outros.

Assumindo que o compilador Java foi informado para procurar pacotes embaixo do diretório:

```
/home/viniciusmayer/projects/example/
```

Apesar de um pacote corresponder a um diretório, o conceito de subdiretório não implica no conceito de subpacote; isto é, se um diretório contém subdiretórios, estes não correspondem a subpacotes. De fato, o conceito de subpacote não existe em Java. Por exemplo, um pacote chamado `nome.do.pacote.subpacote` não tem necessariamente nenhuma relação direta com o pacote `nome.do.pacote`, apesar dos arquivos do primeiro estarem num subdiretório do diretório onde estão os arquivos do segundo. O fato de que o primeiro pacote é referenciado no segundo é apenas um acaso do exemplo.

O Gabrielus foi todo organizado em módulos, ou como estaremos tratando, em pacotes, onde separamos as classes e/ou arquivos que mantem responsabilidades semelhantes para que pudessemos tem, além de uma codificação mais simples, uma visão mais clara de como está arquiteturado o Gabrielus quando de uma visão mais macro do mesmo. Segue nos próximos itens deste seção do trabalho uma explanação com relação aos pacotes que até então compunham o Gabrielus:

- `br.com.viniciusmayer.gabrielus.editor;`
- `br.com.viniciusmayer.gabrielus.framework;`
- `br.com.viniciusmayer.gabrielus.diagram;`
- `br.com.viniciusmayer.gabrielus.form;`
- `br.com.viniciusmayer.gabrielus.animation;`
- `br.com.viniciusmayer.gabrielus.protocol e;`
- `br.com.viniciusmayer.gabrielus.thread.`

3.2.1 Editor

O pacote `br.com.viniciusmayer.gabrielus.editor`, responsável pela construção do aplicativo em si, tem dependência de dois arquivos de propriedades, de dois outros pacotes, `br.com.viniciusmayer.gabrielus.framework` e `br.com.viniciusmayer.gabrielus.diagram` e do *framework*, `violet.jar`.

Os arquivos de propriedades são, em suma, os arquivos de internacionalização. Com as necessidades de criação de aplicações globais, sejam essas em ambiente *Web* ou não, fica cada vez mais comum o uso de aplicações "internacionalizadas", ou seja, aplicações que mudam o idioma de acordo com a língua (em inglês, *language)* e país (em inglês, *country)* do usuário. A palavra internacionalização um tanto quanto grande, mas podemos também usar I18N, que quer dizer que entre o primeiro "I" e o último "N" da palavra "InternationalizatioN" existem 18 letras. No arquivo `DataFlowEditorString` encontramos as internacionalizações referentes ao nome do aplicativo por exemplo, nome do arquivo da logo do aplicativo, nomes e teclas de acesso rápido para os diagramas que estarão disponíveis, assim como o nome que será dado aos arquivos gerados pelo Gabrielus e as suas extensões. Já no arquivo `DataFlowEditorString,` mantem-se o número da versão do Gabrielus e a data de compilação da versão em questão.

Os outros dois pacotes cujo pacote `br.com.viniciusmayer.gabrielus.editor` depende, são pacotes do próprio Gabrielus e que mantém responsabilidades sobre outras funcionalidades do aplicativo. O pacote `br.com.viniciusmayer.gabrielus.diagram`, mantém responsabilidades exclusivas sobre a construção do ambiente de diagramação DataFlow; já o pacote `br.com.viniciusmayer.gabrielus.framework` é responsável pela disponibilização de componentes básico do aplicativo e que, na maior parte das vezes, são usados por todos os pacotes do Gabrielus. E, por último, o framework `violet.jar` que é a responsável pelo processamento de várias das funcionalidades básicas do aplicativo, como por exemplo a do salvamento de um diagrama *dataflow* desenhado no disco rígido da máquina do usuário.

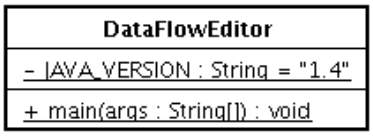

Diagrama 02. Diagrama de Classes do pacote `br.com.viniciusmayer.gabrielus.editor`.

Este é portanto o diagrama de classes do pacote `br.com.viniciusmayer.gabrielus.editor` e, nele apenas a classe `DataFlowEditor`. Nesta classe, apenas um atributo que nos diz qual a versão mínima do JDK que o Gabrielus precisa para garantir o seu pleno funcionamento e, o método `main`. O código do método `main` segue abaixo para que possamos dar uma passada de olhos nele.

```
public static void main(String[] args){
      VersionChecker checker = new VersionChecker();
      checker.check(JAVA_VERSION);
      try{
         System.setProperty("apple.laf.useScreenMenuBar
", "true");
      } catch (SecurityException ex){
         // bom, nós tentamos...
      }

      DataFlowEditorFrame frame = new
DataFlowEditorFrame(DataFlowEditor.class);

      frame.addGraphType("dataflow_diagram",
DataFlowDiagramGraph.class);

      frame.addGraphType("class_diagram",
ClassDiagramGraph.class);

      frame.setVisible(true);
      frame.readArgs(args);
}
```

Primeiramente é feita a verificação da versão do JDK instalado na máquina do usuário. Caso o JDK instalado na máquina seja inferior ao que está definido, é exibido ao usuário a uma mensagem dizendo que o funcionamento do Gabrielus está comprometido, já que ele (no caso o Gabrielus) sugere que se tenha a versão definida no aplicativo, que pode ser "maior" da que está instalado na máquina. O usuário saberá portando que, mesmo estando o aplicativo rodando, pode vir a acontecer alguma ação não esperada, a estilo de não salvar um diagrama, por causa da versão do JDK que ele tem instalado e que é inadequado ao funcionamento pleno do aplicativo.

Indo mais adiante no código, há a instanciação da classe `DataFlowEditorFrame`, classe esta que localiza-se dentro do pacote `br.com.viniciusmayer.gabrielus.framework`. A instanciação desta classe se dá com a passagem por parâmetro de uma `DataFlowEditor`, que em síntese, será usada apenas para obtermos o nome da classes "DataFlowEditor" e carregar dos arquivos de internacionalizações os conteúdos correspondentes ao "editor" (se é que podemos dizer assim!) do qual estamos tratando, no caso um *DataFlowEditor*.

Em seguida, adicionamos ao editor os diagramas que queremos disponibilizar para o usuário. No caso do Gabrielus e mais específico do código acima, estamos disponibilizando ao usuário apenas o acesso aos diagramas `DataFlowDiagram` e `ClassDiagram`. Este último, Diagramas de Classes, foi reaproveitado do *framework* `violet.jar` e, está sendo usuada aqui apenas para demonstração de como acontece a disponibilização, e/ou o processo inverso, de uma espécie de diagrama no editor.

Em fim, é feito com que o *frame* se torno visível ao usuário e, é mostrado em seguida a tela de inicialização do Gabrielus, que também poderá ser acessada pelo menu "Ajuda" e em seguida, na opção "Sobre". Lá encontramos informações com relação a versão do Gabrielus que está em execução, qual a data desta compilação/versão, os autores e também quanto a licença do Gabrielus. Esta licença, numa tradução não oficial da original para o português do Brasil, poderá ser acessada via menu "Ajuda" e, em seguida, na opção "Licença".

3.2.2 Framework

O pacote `br.com.viniciusmayer.gabrielus.framework` está relacionado com os seguintes arquivos, pacotes ou frameworks:

- `DataFlowEditorStrings`: Arquivo de internacionalização;
- `licence.txt`: Arquivo de texto que mantém a licença do Gabrielus;
- `gabrielus.png`: Arquivo de imagem da logo do Gabrielus;
- `br.com.viniciusmayer.gabrielus.animation`: Pacote responsável pela parte de animação;
- `br.com.viniciusmayer.gabrielus.protocol`: Pacote responsável pela parte de protocolo de comunicação entre objetos;
- `br.com.viniciusmayer.gabrielus.threads`: Pacote responsável pelo processamento *multithreading* e;
- `violet.jar`: *Framework* Violet.

No arquivo `DataFlowEditorString` encontramos as internacionalizações que dizem respeito aos textos que são mostrados nos menus do Gabrielus, os *mnemonics*, assim como as teclas

de atalho para estes menus. Há também os textos correspondentes as mensagens que são exibidas em eventos do tipo, fechando o Gabrielus sem ter salvo um diagrama que acabou de ser desenhado, problemas com versão do Java e/ou o texto que aparece na tela "Sobre" (Menu "Ajuda", item "Sobre") do aplicativo.

Acessando o menu "Ajuda" do Gabrielus e, em seguida no item "Licença", estaremos acessando a licença que protege o Gabrielus. A versão da licença que é exibida nesta tela é uma tradução para o português do Brasil, não oficial, da licença original da GNU General Public Licence. Esta tradução está sendo mantida no arquivo `licence.txt`, arquivo de texto que é armazenado no pacote `framework` e que é uma das dependencias que este mesmo pacote tem para que seu funcionamento seja feito sem problemas. Em seguida temos também uma dependência deste pacote com o arquivo `gabrielus.png`, que é o arquivo que mantém a logo do Gabrielus. A logo do Gabrielus foi desenhada por Eleonor Vinicius Dudel Mayer[19] e também, assim como o próprio Gabrielus, é guardada sobre a licença GNU General Public License.

19 Site de Eleonor Vinicius Dudel Mayer: http://viniciusmayer.googlepages.com/ e/ou e-mail viniciusmayer@gmail.com;

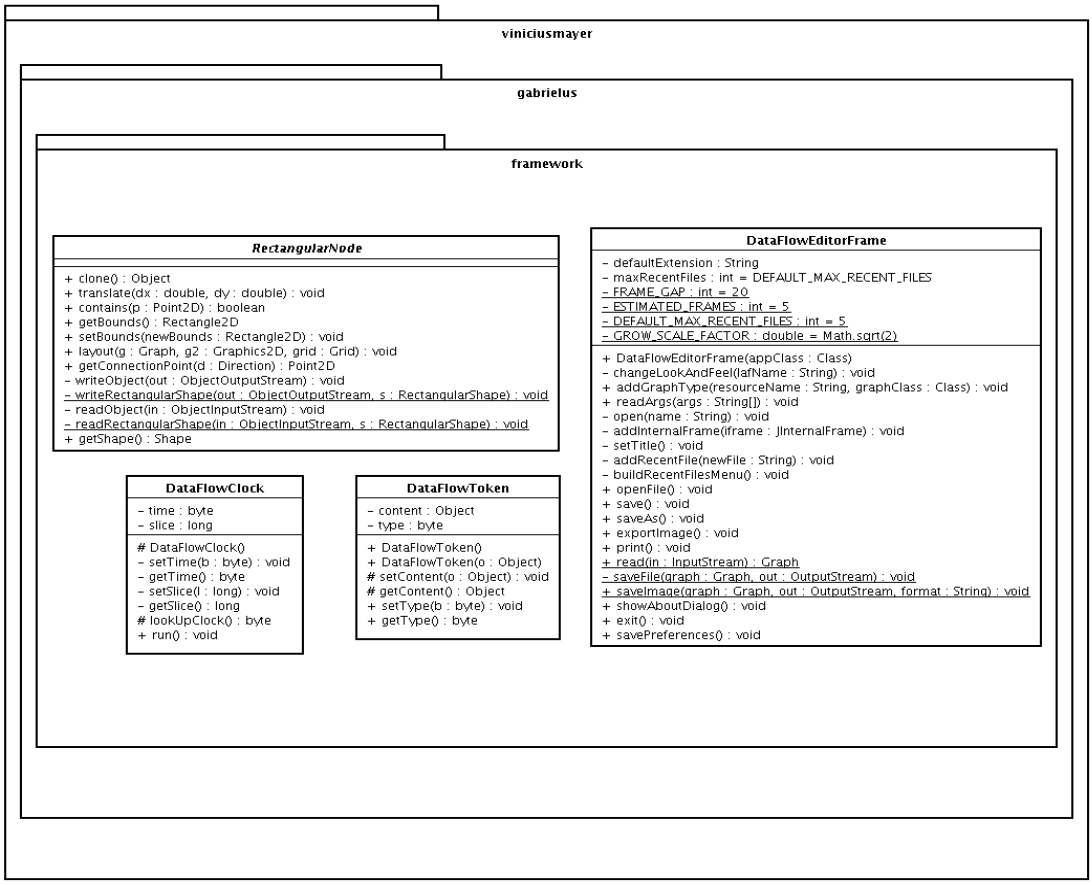

Diagrama 03. Diagrama de Classes de visão panorâmica do pacote `br.com.viniciusmayer.gabrielus.framework`.

O pacote `br.com.viniciusmayer.gabrielus.animation` é um dos pacotes que também se relacionam diretamente com o pacote `framework`, cujo funcionamento só é garantido uma vez que este (o pacote `br.com.viniciusmayer.gabrielus.animation`) esteja também em funcionamento. O pacote `br.com.viniciusmayer.gabrielus.animation` é o que se responsabiliza pela configuração das animações que estarão acontecendo na camada de visão do nosso aplicativo. Uma vez que o `framework` está fazendo todo o processamento propriamente dito das animações, e dando as coordenadas para a movimentação dos objetos na camada de visão, faz-se necessário que esteja tudo *OK* com o pacote `br.com.viniciusmayer.gabrielus.animation`, para que a representação gráfica da animação seja executada com sucesso e corretamente.

Da mesma forma, há dependencias também por parte do pacote `framework` dos pacotes `br.com.viniciusmayer.gabrielus.protocol` e `br.com.viniciusmayer.gabrielus.threads`. Este pacotes estão também diretamente envolvidos com o pacote `framework` por serem os

responsáveis pelas orientação necessárias à execução ou não de alguma animação. Enquanto o pacote `br.com.viniciusmayer.gabrielus.protocol` vare a representação interna do diagrama desenhado na camada de visão e define quais os processos de animação que deverão, necessariamente, acontecer após terminada a varedura, o pacote `br.com.viniciusmayer.gabrielus.threads` estará sendo responsabilizado pelo processamento, em paralelo, das animações que por ventura tiverem sido definidas pelo processamento realizado pelo pacote `br.com.viniciusmayer.gabrielus.protocol`.

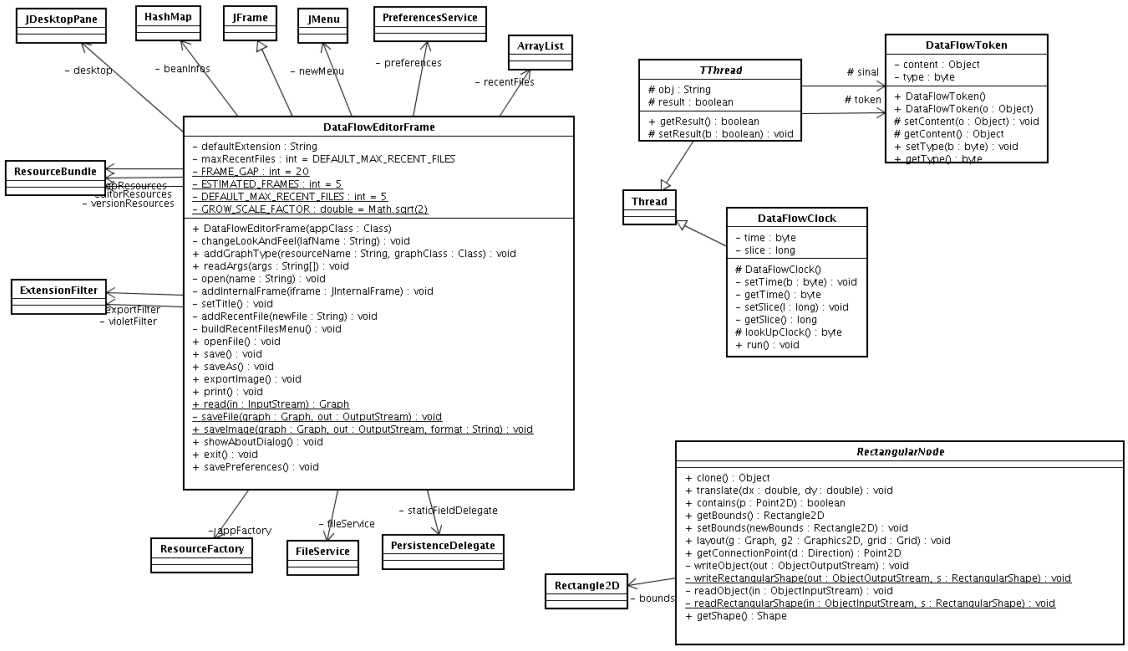

Diagrama 04. Diagrama de Classes detalhado do pacote `br.com.viniciusmayer.gabrielus.framework`.

Perceba que, no diagrama de classe detalhado que disponibilizamos logo acima, as classes que compõe o nosso pacote `framework` estão associadas com diversas classes externas ao nosso próprio pacote. Estas classes são providas por um dos *frameworks* que estamos utilizando na arquitetura do Gabrielus e, para o pacote `framework`, são classes exclusivamente disponibilizadas pelo framework `violet.jar`. Classes como `PersistenceDelegate`, `JDesktopPane`, `RectangularNode`, e as demais que aparecem no diagrama com exceção das que compõe o próprio pacote, são vindas do *framework* `violet.jar`. É claro, portanto, que é imprescindível que este *framework* esteja disponível e em pleno funcionamento.

3.2.3 Diagram

O pacote `br.com.viniciusmayer.gabrielus.diagram` é o responsável pela montagem do editor de diagramas *dataflow*. É ele que dispõe os objetos disponíveis para um diagrama *dataflow* na barra de objetos, mantem os *listeners* do *panel* onde estarão sendo desenhados, literalmente, os diagramas, assim como também é o responsável em manter os relacionamentos que vierem a existir entre os objetos que estarão sendo utilizados para montar o diagrama em questão.

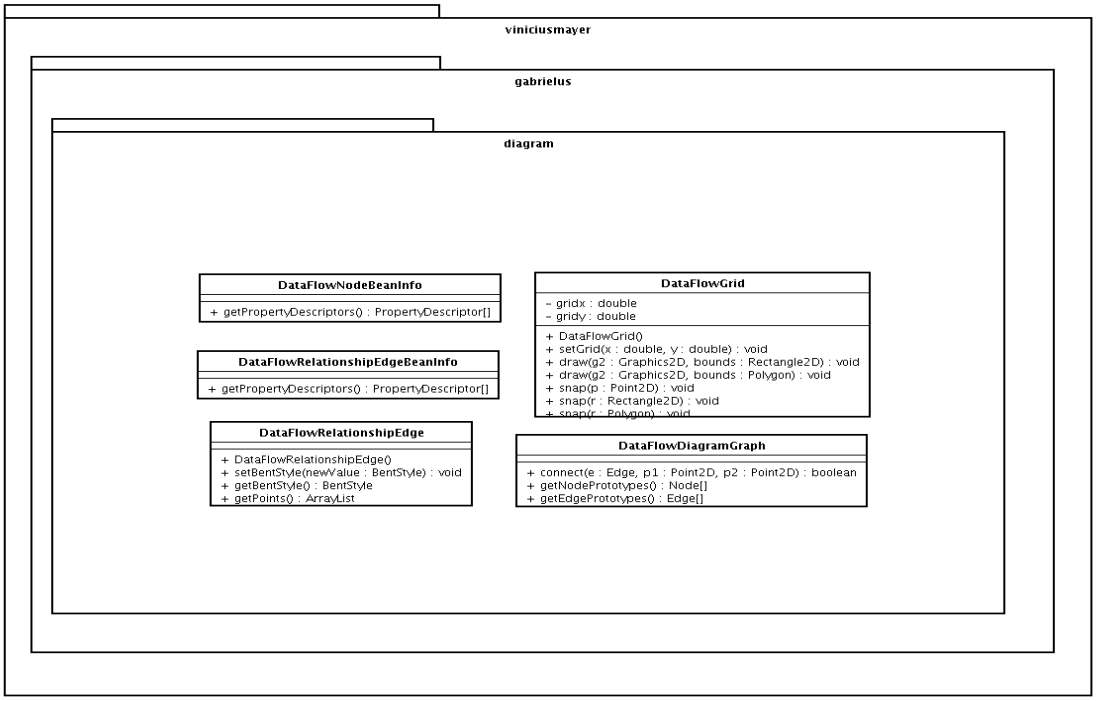

Diagrama 05. Diagrama de Classes de visão panorâmica do pacote `viniciusmayer.gabrielus.diagram`.

É de responsabilidade deste pacote também unir a representação interna de um objeto com o sua representação visual (desenho) na camada de visão e, para estes dois, a área de *listener* de ambos. Ou seja, existe na camada de visão um objeto em sua representação interna (que o usuário não vê), uma representação visual e, um *listener* que estará unindo essas representações. É possível, portanto que, se mal configurada esta camada no que dirá respeito a um tipo de objeto em específico, de termos as três partes que compõe no final um objeto do diagrama separadas e, nenhuma delas com uma funcionalidade que seja de alguma forma útil.

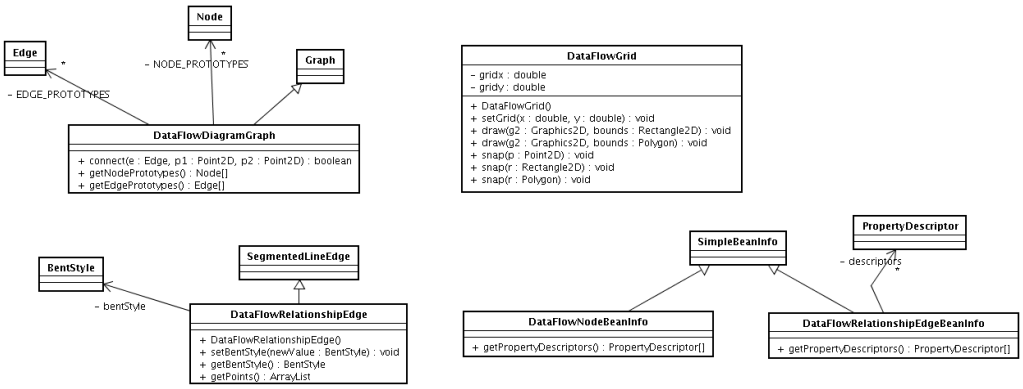

Diagrama 06. Diagrama de Classes detalhado do pacote `br.com.viniciusmayer.gabrielus.diagram.`

O pacote `diagram` está relacionado com o pacote `br.com.viniciusmayer.gabrielus.form` e também com o *framework* `violet.jar`. O pacote `br.com.viniciusmayer.gabrielus.form` nos trás as funcionalidades voltadas ao que diz respeito unicamente a representação visual de um objeto. E o *framework* `violet.jar`, neste caso, nos ampara nas questão que tratam dos relacionamentos entre os objetos. Tanto em manter estas ligações numa representação "interna" (aproveitando a idéia da representação interna dos objetos) destes relacionamentos, quanto a representação visual dos mesmos. Veja no diagrama que segue que são utilizadas classes como `SegmentedLineEdge, Edge, Node,` entre outras que são externas a este pacote `diagram` e que são, portanto, classes que o *framework* `violet.jar` nos fornece.

3.2.4 Form

Quanto ao pacote `br.com.viniciusmayer.gabrielus.form`, exite apenas dependência do *framework* `violet.jar`. O pacote `form` resolve um problema que não é tão simples quanto parece, que é a com relação a representação visual, na camada de visão do Gabrielus, dos objetos que estarão internamente em funcionamento, isso tanto quando se está desenhando e dando propriedades aos objetos, como também quando estes mesmos objetos estarão passeando pelo diagrama devido a uma animação.

É de responsabilidade deste pacote também dar funcionalidade então a esta representação

visual dos objetos que internamente já existem. Funcionalidades como por exemplo, setar o nome do de um objeto específico, mudar a cor do preenchimento de um objeto específico, entre outros.

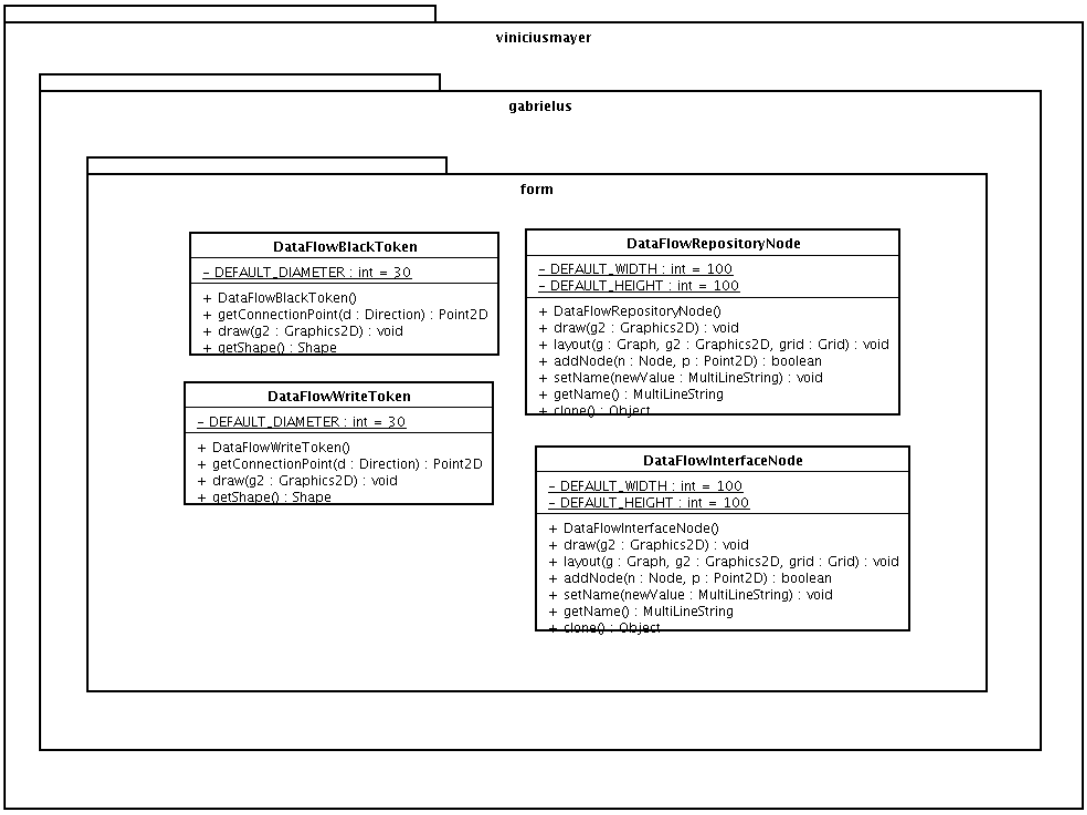

Diagrama 07. Diagrama de Classes de visão panorâmica do pacote `viniciusmayer.gabrielus.form`.

Abaixo segue um diagrama de classes detalhado do pacote `form`. Note que para os objetos que já estão disponíveis na barra de objetos do editor de diagramas *dataflow* (Repositório e Interface), há o uso de apenas duas classes externas, classes do *framework* `violet.jar`, e que todos herdão de `RectangularNode`, ou seja, todos hão de carregar as características internas de um nodo retangular (em inglês, *RectangularNode*), como por exemplo ter a capacidade de receber um nome, mas não necessariamente terão/herdarão a representação visual do mesmo.

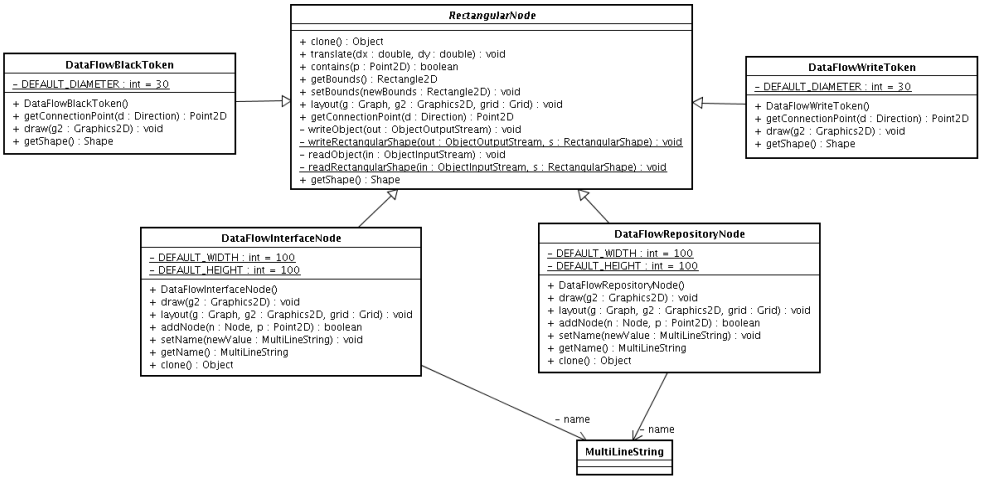

Diagrama 08. Diagrama de Classes detalhado do pacote `viniciusmayer.gabrielus.form`.

Cada objeto implementa a sua própria representação visual e, em síntese, todos os objetos que forem do tipo "ativo" estarão vinculados a esta estrutura, pois ele precisão, necessariamente, serem capazes de encapsular um objeto dentro deles, que poderá ser um *token* branco ou um *token* preto. Os demais não precisarão ter esta capacidade, pois o mais interessante nestes objetos será a implementação do protocolo de comunicação. Quanto as animações, o posicionamento no *panel* de uma representação visual de um objeto dito ativo será o inicio ou o fim da animação, os demais, apenas para que possamos desenhar o caminho a ser percorrido pelo *token* enquanto ele estiver viajando pelo diagrama do seu repositório origem (coordenadas de inicio da animação), até o seu repositório destino (coordenadas de fim da animação), e entre este os demais objetos que atuam como coordenadas intermediárias, coordenadas de referência, da animação propriamente dita.

3.2.5 Animation

O pacote `br.com.viniciusmayer.gabrielus.animation` se resume as configuração das animação do Gabrielus. Temos uma classe que mantém as propriedades relacionadas ao comportamento das animações (`DataFlowAnimationControlPanel`), outra classe (`DataFlowAnimationView`) que constrói a tela de configuração com uma panel de demostração da configuração em questão e ainda, a classe que é o nosso *listener* na aplicação (`DataFlowAnimationSetup`).

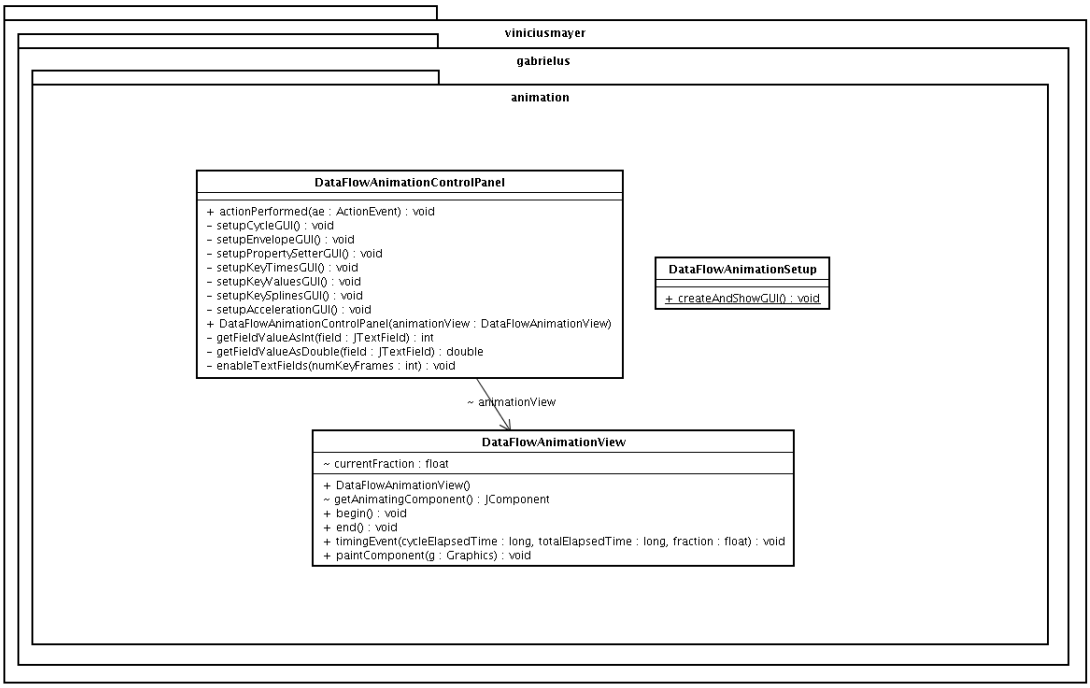

Diagrama 09. Diagrama de Classes de visão panorâmica do pacote `br.comviniciusmayer.gabrielus.animation.`

O pacote `animation`, cujo pacote `br.com.viniciusmayer.gabrielus.framework` depende, tem como dependencia sua o *framework* TimingFramework. Observe no diagrama abaixo que a classe `TimingController` esta associada a classe `DataFlowAnimationControlPanel` e que a *interface* `TimingTarget` é implementada pela classe `DataFlowAnimationView`, ambas as classes são providas pelo *framework* TiminFramework.

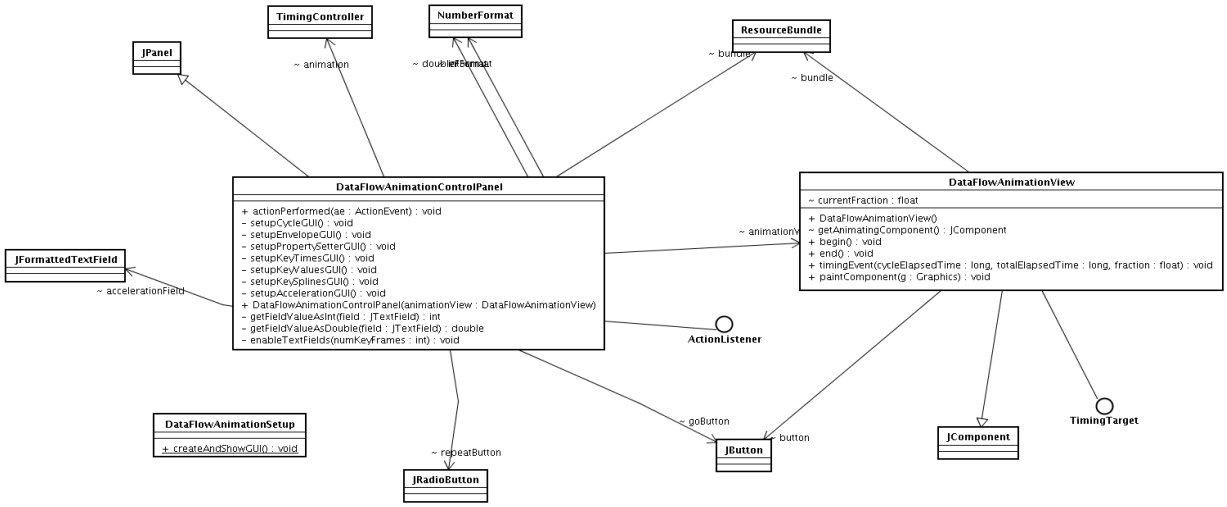

Diagrama 10. Diagrama de Classes detalhado do pacote `br.com.viniciusmayer.gabrielus.animation.`

As demais classes que este pacote se relaciona são classes da API do Java para a construção da camada de visão do módulo de configuração de animação e, quanto a dependência do *framework* TimingFramework, se resume as duas classes `TimingController` e `TimingTarget`. Tamanha simplicidade se deve ao fato de que, primeiro, o *framework* TimingFramework é bastante amigável e nos disponibiliza estas classes que abstraem toda e qualquer complicação com relação aos procedimentos de configuração e processamento da animação; segundo, porque conseguimos manter um bom nível de modularização na construção desta *feature* da arquitetura do Gabrielus e, para este módulo, portanto, resta a pessoa que estiver trabalhando com este módulo, acessar a classe `DataFlowAnimationSetup`.

3.2.6 Projeto labi

A seguir estamos apresentando os módulos do projeto desenvolvido na disciplina de Laboratório de Sistema I que foram aproveitados em sua totalidade no desenvolvimento do projeto em questão. Os módulos que nos referimos são, respectivamente, o módulo que trata especificamente da arquitetura do protocolo de comunicação entre os objetos e, o módulo que trata do processamento multithreading do aplicativo.

3.2.6.1 Protocol

O módulo do protocolo de comunicação entre objetos define, portanto, de que forma se realizará a comunicação entre os objetos ativos do sistema. A comunicação que desejamos que aconteça entre estes objetos ativos nada mais é do que a transferência do conteúdo de um objeto para outro. Os componentes que estão sendo considerados ativos, portanto, são o Repositório e a Porta de entrada; o repositório porque pode tanto querer transferir um objeto `Token` como também receber objetos e; a Porta de Entrada, pois, embora não seja capaz de receber objetos, estará gerando objetos e, conseqüentemente, querendo transferi-los. Então, estamos entendendo por objeto ativo todo aquele componente do sistema que é capaz de iniciar uma nova fase de transferência de `tokens` entre os objetos do sistema, ou seja, queira fazer com que este conteúdo, o `Token`, flua, ou continue fluindo por entre os objetos constituintes do sistema.

Baseamos o nosso protocolo de comunicação em um protocolo muito simples, o modelo de solicitação/resposta (em inglês, *request/response*). Este protocolo é a idéia que há por trás das estruturas mais comuns de sistemas distribuídos do tipo cliente-servidor. O cliente envia uma mensagem ao servidor solicitando algum tipo de serviço, como, no nosso caso, uma requisição de transferência; o servidor faz o trabalho e envia para o cliente os dados solicitados. Podemos então fazer uma analogia desta dinâmica com os nossos componentes onde, o cliente seria um objeto, um Repositório por exemplo, e o servidor poderia ser qualquer outro objeto, inclusive um Repositório. E isso é característico desse tipo de estrutura de serviço, pois tanto o cliente quanto o servidor mantêm o mesmo tipo de implementação e, portanto, tanto os clientes quanto os servidores podem executar processos como clientes.

Devido a esta estrutura extremamente simples é que conseguimos fazer com que o serviço de comunicação entre objetos, o protocolo propriamente dito, fosse reduzido a duas camadas apenas, uma que possibilita o envio de mensagens, a interface `PRequest`, e outra para recepção das mesmas, `PResponse`.

Diagrama 11. Diagrama de Classes de visão panorâmica do pacote `br.com.viniciusmayer.gabrielus.protocol`.

Para o processo de transferência de objetos então, subdividimos esta estrutura de conversações entre componentes do sistema em três estágios (ou camadas): a) estágio de pré-requisições, que fazem um levantamento prévio da situação dos objetos envolvidos na transferência, e, principalmente, sem fazer alterações no estado dos objetos envolvidos; b) estágio de requisições, e estas estarão de certa forma, estabelecendo contratos com os objetos destino e, por último; c) o estágio de transferências de `tokens` entre objetos: onde acontece, de fato, a manipulação do `Token` de um objeto ativo para outro, dependendo, é claro, dos resultados apresentados nos estágios anteriores.

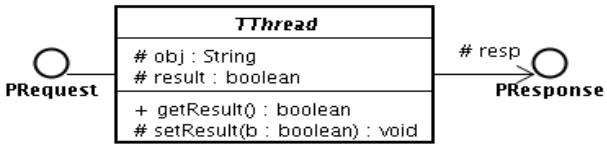

Diagrama 12. Diagrama de Classes detalhado do pacote `br.com.viniciusmayer.gabrielus.protocol`.

Para os objetos que pretenderem atender chamadas de pré-requisições, requisições ou mesmo transferências, definimos, com já mencionado uma interface `PResponse`. A implementação dos métodos definidos nesta interface, para cada objeto em especial, é a principal razão de ser de cada componente, já que é justamente esta implementação que dá ao componente a sua identidade; para os objetos que precisarão enviar pedidos de pré-requisição, requisição e/ou transferência definimos a interface `PRequest` e, a implementação dos métodos definidos em `PRequest` nada mais é do que o chamamento dos métodos de resposta no(s) objeto(s) que estiver(em) definidos como saída(s) do componente requisitante.

3.2.6.2 Threads

A classe abstrata `TThread` é a que define o modelo de classes do módulo de *multithreading* do protótipo. Há nela várias construções importantíssimas para o entendimento e, o que é de propósito maior, funcionamento do paralelismo no protocolo de comunicação. Temos, portanto a extensão da classe `Thread` e a implementação da interface `PRequest` (do módulo do protocolo de comunicação) na classe `TThread`. Uma vez que a classe `TThread` é abstrata, a implementação do método `run` herdado da classe `Thread` ficará para as suas subclasses e, também a implementação dos métodos definidos pela interface `PRequest`.

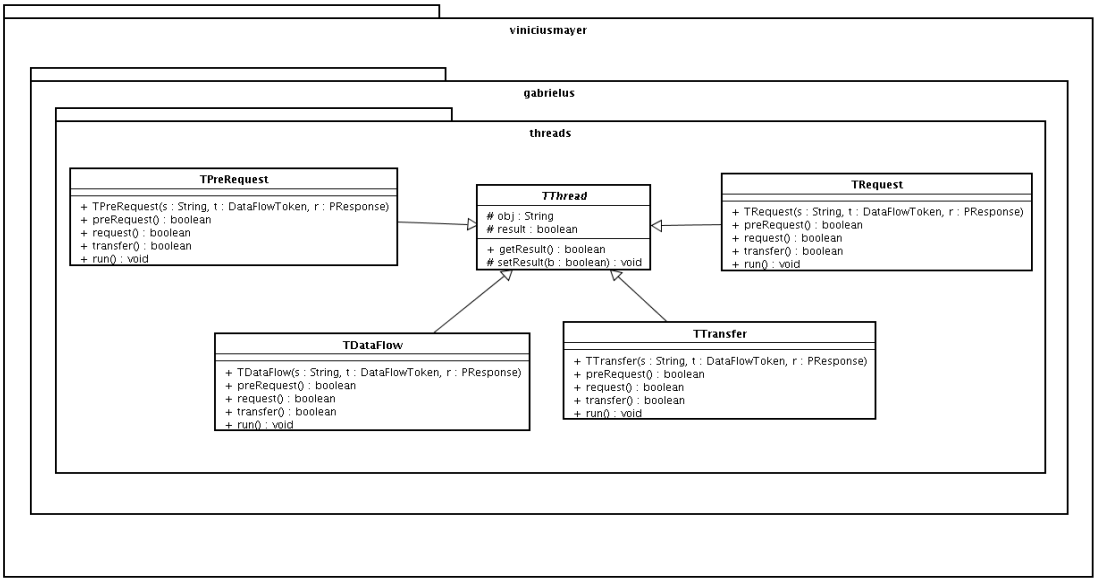

Diagrama 13. Diagrama de Classes de visão panorâmica do pacote `br.com.viniciusmayer.gabrielus.tthread`.

A classe `TDataFlow`, primeiramente, é subclasse de `TThread` e deve, necessariamente, implementar os métodos `run` (herdado de `Thread` por `TThread`) e `preRequest`, `request` e `transfer` (herdados – por interface – de `PRequest` por `TThread`). Esta classe implementa, portanto, todo o processo de fluxo de dados que estará acontecendo de Repositórios para Repositórios, Repositórios para Portas de Saída, de Portas de Entrada para Repositórios, e uma última possibilidade, de Porta de Entrada para Porta de Saída. E não que não estarão envolvidos os demais componentes do núcleo, mas eles apenas estão representando funções intermediárias, ou de condutores, no decorrer do fluxo, e não necessariamente se colocando como ponto de partida ou ponto de chegada de uma transferência, do fluxo de um `token`.

Já que é a classe `TDataFlow` que implementará todo o processo de, desde o estabelecimento de comunicação entre os objetos até o transporte propriamente dito do `token`, os três estágios do processo de transferência entre objetos estarão aninhados na implementação do método `run` desta, a pré-requisição, a requisição e a transferência, nesta ordem. Os únicos objetos que, neste cenário, usarão as funcionalidades fornecidas por esta classe serão o Repositório e a Porta de Entrada, visto que são apenas este os componentes capazes de iniciar um fluxo de dados e que, serão, também, parte dos componentes que podem vir a se tornar o final do fluxo, do qual basta ainda incluir a Porta de Saída.

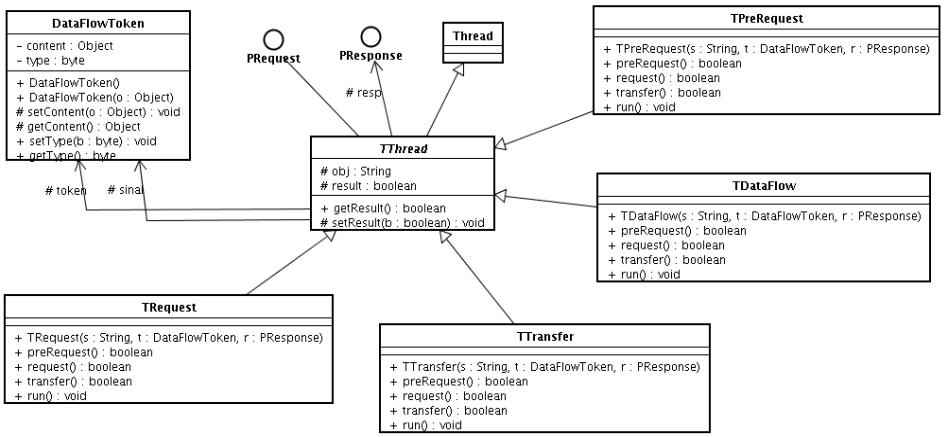

Diagrama 14. Diagrama de Classes detalhado do pacote `br.com.viniciusmayer.gabrielus.tthread`.

`TPreRequest` é a classe cujo método `run` (herdado de `TThread`) implementará o processo de pré-requisição para todos os demais componentes do protótipo, ou seja, para os Seletores, para os Polarizadores, para o Interruptor, a Interface, o Functor e o Multiplicador, ficando de fora o Repositório e a Porta de Entrada (que se aproveitarão da classe `TDataFlow`) e a Porta de Saída que não implementará processamento de transferência de `token` justamente por ser um dos componentes de final de fluxo. Haverá ainda de existir alguns componentes descendentes do tipo Função, mais estes serão composições destes objetos mais primitivos e farão uso, então, deste mesmo recurso para implementação do processo de pré-requisição.

A implementação da classe `TRequest` diz respeito única e exclusivamente pelo processamento das requisições que serão realizadas entre os objetos envolvidos na transferência de um `token`. Esta classe também deverá implementar os três métodos que foram definidos na interface `PRequest`, no entanto, apenas o método `request` que será utilizado realmente no processamento da requisição solicitada.

É o `TTransfer` que estará mantendo os dados necessário para o transporte do `token` de um objeto para outro durante o processo de fluxo de dados, o processo de transferência de `tokens`. Após os estágios anteriores terem varrido todo o diagrama, setado todas as variáveis necessárias para o fluxo perfeito do `token` por entre os objetos, e feito o que estamos chamando de contrato com os demais componentes envolvidos no fluxo de `token`, a classe `TTransfer` aparece para ser o

intermediário entre então, os componementes intermediário, condutores do `token` em questão.

Dá-se por encerrado fluxo de dados, a transferência de `token`, quando todos as `threads TTransfer` estiverem mortas e já também já tiverem sido consultadas por seus geradores.

4 Ambiente Web

O fenômeno da Internet trouxe para os desenvolvedores de sistemas pelo menos dois novos desafios: toda aplicação deve ser distribuída (ao contrário das aplicações tradicionais, "monolíticas", que rodam em um computador isolado); e toda aplicação é formada pela agregação de um conjunto de tecnologias e linguagens (tradicionalmente bastava dominar uma única tecnologia e linguagem de programação). Surgiu então o conceito de aplicação para *web* (em inglês, *web application*).

A construção de uma aplicação para *web* é um processo diferente do usado no desenvolvimento de aplicações tradicionais. Nestas, analistas e programadores têm à sua disposição uma biblioteca de classes (no paradigma orientado a objeto) que devem ser usadas para compor uma aplicação que seja executável. Cabe a eles tomar todas as decisões sobre como estruturar a nova aplicação (sua arquitetura, componentes, etc.).

Já para o desenvolvimento de aplicações para *web*, surge a figura do *web designer* que exerce uma função específica: projetar o *web site*. Analistas e programadores, por sua vez, partem de um contexto previamente definido: o sistema será distribuído normalmente usando o modelo cliente-servidor[20], HTTP será o protocolo de comunicação entre as aplicações (programa cliente e programa servidor), TCP/IP[21] será o protocolo de transporte de dados pela rede, etc. Além disso, em

20 Para que um utilizador possa navegar na internet, transferir ficheiros, enviar / receber correio eletrônico ou utilizar qualquer outro tipo de serviços da internet, necessita, antes de mais, de estabelecer uma ligação com um computador remoto que lhe forneça estes serviços. Neste cenário, o computador do utilizador é o cliente e o computador ao qual ele se conecta é o servidor ou prestador de serviços.

21 Os protocolos internet formam o grupo de protocolos de comunicação que implementa a pilha de protocolos sobre a qual a internet e a maioria das redes comerciais roda. Eles são algumas vezes chamados de protocolos TCP/IP, já que os dois protocolos mais importantes desse modelo são: o protocolo TCP - Transmission Control Protocol

muitos casos, o programa cliente, estritamente falando, é o próprio *browser* do usuário. Quanto ao programa servidor, muitas das tarefas rotineiras (carregamento inicial, configuração, definição de *logs*, mecanismos de autenticação de usuários, etc.) já estão previamente implementadas. Cabe aos analistas e programadores focar sua atenção quase que exclusivamente nas especificidades da aplicação a ser desenvolvida e saber interagir com o *web designer*.

E para tanto que desenvolvemos o site do projeto Gabrielus, para experimentar o desafio de tornar o Gabrielus não mais apenas uma aplicação que roda em um computador isolado e, portanto, tornar-se distribuída e acessível a quem estiver conectado à internet; e também para experimentar os desafios da agregação de diversas tecnologias e linguagens num único propósito, tornar simples o desenvolvimento e a disponibilização do Gabrielus. Segue diagrama de implantação do Gabrielus, onde mostra-se como que foram, então, arquiteturados os serviços que provêm um ambiente bastante simples de desenvolvimento e disponibilização do Gabrielus.

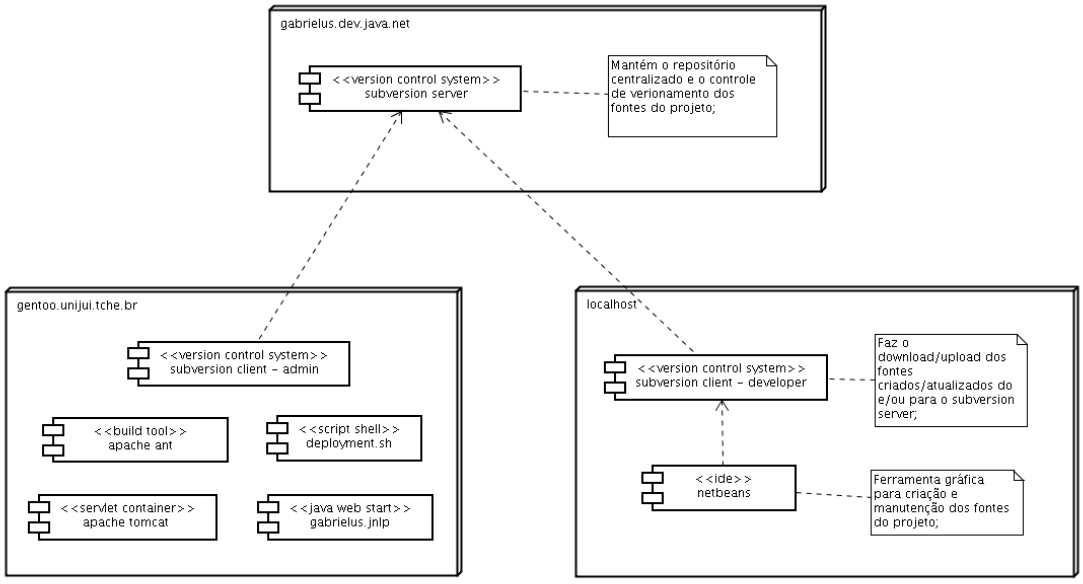

Diagrama 15. Diagrama de Implantação do Gabrielus, representa a configuração e a arquitetura do site do projeto, cujas ligações entre os respectivos componentes é representada.

Temos então três cenários distintos, o primeiro se refere ao ambiente que nos é disponibilizados pelo Java.Net, que é o servidor Subversion (`gabrielus.dev.java.net`); segundo é o servidor onde estamos hospedando o site do Gabrielus (`gentoo.unijui.tche.br`) e; terceiro, a

(Protocolo de Controle de Transmissão) - e o IP - Internet Protocol (Protocolo Internet). Esses dois protocolos foram os primeiros a serem definidos.

uma máquina local qualquer (`localhost`), onde estará sendo desenvolvimento o Gabrielus.

No nosso servidor Subversion, mantido pelo Java.Net, e sob o nome `gabrielus.dev.java.net`, que então os nosso dados, os nossos fontes, armazenados. Estamos utilizando este repositório de dados não apenas para fazer o controle de versionamento dos códigos fonte do Gabrielus, mas também para mantermos versionados todos os documentos que estão relacionados ao projeto em si. Estes documentos pode estar relacionados ao próprio Gabrielus e se trata dos fontes, pode tambéms er os arquivos que constituem o *site* do projeto, documentos como este, o relatório do projeto, e ou qualquer outro documento que tenha sido criado para o desenvolvimento de qualquer uma das "atividades" mencionadas.

No servidor Web, mantido pela UNIJUÍ, e com o nome `gentoo.unijui.tche.br`, que estamos hospedando o *site* do projeto. Também é neste servidor que implementamos os principais servicos de disponilização do Gabrielus. Há um cliente Subversion para que possamos obter as atualizações dos fontes do Gabrielus do Java.Net, o Apache Ant para a compilação e disposição dos arquivos necessários para o funcionamento do site nos lugares corretos, um Script Shell para automatiza os processos de atualização do *site* (no que diz respeito exclusivamente a disponibilização de uma nova versão do aplicativo), um *web server*, Apache Tomcat, para disponibilizar o nosso *site* e aplicação Java na internet e; por último, o Java Web Start, que é o serviço configurado que disponibiliza o Gabrielus no *site* do projeto. Mas detalhes dos trabalhos realizados nesta máquina serão descritos adiante, na seção deste trabalho que diz respeito exclusivamente a isso (item 4.5. Servidor).

E na máquina local, `localhost`, que estará sendo feito o desenvolvimento do Gabrielus e será necessária uma configuração bem simples para que se passa trabalhar efetivamente no desenvolvimento do mesmo. Primeiramente, para que possamos obter os fontes será necessário uma ferramenta de controle de versionamento de código, um cliente do Subversion. Uma vez que a gente tenha este cliente e consigamos fazer um controle efetivo do versionamento do código fonte do Gabrielus, precisaremos de uma IDE de desenvolvimento. Esta IDE de desenvolvimento fica a critério do desenvolver, embora seja sugerido o uso do NetBeans. O projeto do Gabrielus é NetBeans-*friendly* e é de fácil integração com o NetBeans, uma vez que, sendo adotada a versão 5.5 *beta* da IDE NetBeans, bastará o esforço de abrir o projeto que o cliente subversion traz para nós lá do servidor subversion.

Com relação ao demais documentos relacionados ao projeto, mais duas ferramentas seriam necessárias, embora não esteja definidas no diagrama de implantação disposto acima. Para os documentos de texto, estamos utilizando o OpenOffice2.0 e; para a elaboração da diagramação UML, utilizamos a ferramenta Jude. Logo, uma vez que se tenha este aplicativos em funcionamento na máquina do desenvolvedor, será possível trabalhar com qualquer parte do projeto, deste a codificação até aos arquivos HTML.

4.1 Subversion

Um sistema de controle de versão (ou de versões) é um software com a finalidade de controlar diferentes versões no desenvolvimento de um documento. Esses sistemas são muito usados no desenvolvimento de software para controlar as diferentes versões - histórico e desenvolvimento - dos códigos-fontes e também da documentação.

Cada sistema tem sua particularidade, mas a maioria deles compartilham alguns conceitos básicos. Apesar disso, é possível que algum sistema específico funcione de maneira totalmente diferente. A maior parte das informações - com todo o histórico - ficam guardadas num repositório (em inglês, *repository*), num servidor qualquer. Geralmente o acesso é feito por um cliente pela rede (via *socket*[22]) e pode ser feito localmente quando o cliente está na mesma máquina do servidor. O repositório armazena a informação - um conjunto de documentos - de modo persistente num sistema de arquivos ou num banco de dados qualquer. É possível que o armazenamento seja feito em outros dispositivos capazes de "eternizar" e resgatar facilmente a informação.

Cada servidor pode ter vários sistemas de controle de versão e cada sistema pode ter diversos repositórios, limitando-se na capacidade de gerenciamento do software e também no limite físico do hardware. Geralmente um repositório possui um endereço lógico que permite a conexão do cliente. Esse endereço pode ser um conjunto IP/porta, uma URL[23], um caminho do sistema de

22 Um soquete (do inglês socket) é, generalizado, uma tomada. Designa uma cavidade ou região usada para ligar algum artifício específico. Especificamente em computação, um soquete pode ser usado em ligações de redes de computadores para um fim de um elo bidirecional de comunicação entre dois programas. A interface padronizada de soquetes surgiu originalmente no sistema operacional Unix BSD (Berkeley Software Distribution); portanto, eles são muitas vezes chamados de Berkeley Sockets.

23 Uma URL (de Universal Resource Locator) em português significa (Localizador Uniforme de Recursos) é o endereço de um recurso (um ficheiro, uma impressora etc.), disponível em uma rede; seja a Internet, ou uma rede corporativa, uma intranet. Uma URL tem a seguinte estrutura: `protocolo://máquina/caminho/recurso`. O protocolo poderá ser HTTP, FTP, entre outros. A máquina designa o servidor que disponibiliza o documento ou

arquivos, entre outros. Cada desenvolvedor possui em sua máquina uma cópia local (em inglês, *working copy*) somente da última versão de cada documento. Essa cópia local geralmente é feita num sistema de arquivos simples. A cada alteração relevante do desenvolvedor é necessário "atualizar" as informações do servidor submetendo (em inglês, *commit*) as alterações. O servidor então guarda a nova alteração junto com todo o histórico mais antigo. Se o desenvolvedor quer atualizar sua cópia local é necessário atualizar as informações locais, e para isso é necessário fazer baixar novidades do servidor (em inglês, *update*).

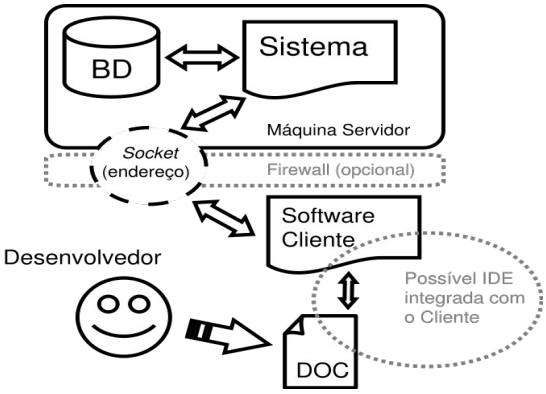

Figura 01. Esquema geral físico.

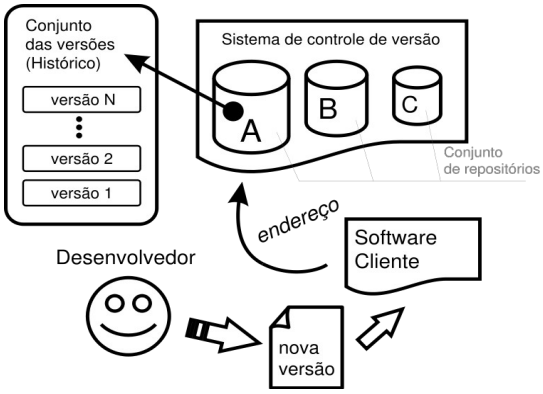

Figura 02. Esquema geral lógico.

4.2 Ant

Ant é uma das mais tradicionais e conhecidas ferramentas desenvolvida pelo projeto Jakarta. Seus objetivos estão ligados a automatização de processos de construção (em inglês, *build*), além de testes e distribuição (em inglês, *deploy*) de aplicações para projeto em Java. Isto é, necessidade de automatização de tarefas como compilação de projetos, montagem de `.jar` files ou até criação de Javadoc, estão nos limites de atuação do Ant.

O Ant esta ligado aos conceitos de integração continua, muito comum na metodologia XP[24]

recurso designado. O caminho especifica o local (geralmente num sistema de ficheiros) onde se encontra o recurso dentro do servidor.

24 Programação eXtrema (do inglês eXtreme Programming), ou simplesmente XP, é uma metodologia ágil para equipas pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança. Para isso, adota a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software. A metodologia XP promove como seus quatro valores fundamentais: comunicação, simplicidade, feedback e coragem. A partir desses valores, possui como princípios básicos: feedback rápido, assumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade.

e sua importância o fez ser incorporado em importantes IDEs do mercado, como Eclipse e/ou o NetBeans. O Ant tem entre suas maiores virtudes o fato de ser independente de Sistema Operacional (SO), pois é feito em Java. Ant além de ser independente de SO, também é independente de IDE e está integrado nas principais IDEs de desenvolvimento que encontramos hoje a disposição hoje.

A ferramenta Ant tem a capacidade de automatizar diversas tarefas que fazem parte do ciclo de vida de desenvolvimento de software. Outros exemplos, além dos já citados, são a capacidade de manipulação de diretórios, automatização de testes e controle de versão. Ant também pode ser útil para programação em linguagem C. É possível elaborar *tasks* complementares que fazem desde geração de código para integração de Java e C através da API JNI[25] até a possibilidade de checagem de estilo de código.

Para a instalação, deve-se fazer o *download* do Ant do site do projeto Apache[26] (ver referências). É necessário descompactar o arquivo em um diretório de sua preferência, a qual chamaremos `ANT_HOME`. Além disso, é necessário adicionar a variável de ambiente `ANT_HOME`, com valor apontando para o diretório onde você descompactou o Ant. Para finalizar, adicione o caminho `ANT_HOME/bin` à variável de ambiente `PATH`.

Depois de de descompactar você deve seguir os seguintes passos:

- Copie o diretório que foi descompactado para onde você desejar melhor, exemplo:

```
/usr/local/jakarta-ant-1.5.1
```

- Como root, edite o arquivo `/etc/profile` e crie uma variável de ambiente `ANT_HOME` apontando para o diretório onde esta o Ant, exemplo:

```
export ANT_HOME=/usr/local/jakarta-ant-1.5.1
```

- Agora devemos configurar a `PATH`:

```
export PATH=$PATH:$ANT_HOME/bin
```

25 JNI ou Java Native Interface é um padrão de programação que permite que a máquina virtual da linguagem Java acesse bibliotecas construídas com o código nativo de um sistema. Ela permite também que aplicações Java sejam embutidas em aplicações nativas.

26 Site do Apache Ant: http://ant.apache.org/

- Depois escreva `ant` na linha de comando, se o resultado for como o que se segue, Beleza, Ant instalado com sucesso!

```
Buildfile: build.xml does not exist!
Build failed
```

4.3 Tomcat

O Tomcat é um servidor de aplicações Java para *Web*. É software livre e de código aberto surgido dentro do conceituado projeto Apache Jakarta[27] e oficialmente endossado pela Sun[28] como a Implementação de Referência (RI) para as tecnologias Java Servlet[29] e JavaServer Pages[30] (JSP). Atualmente, o Tomcat tem seu próprio projeto dentro da Apache Software Foundation[31]. O Tomcat é robusto e eficiente o suficiente para ser utilizado mesmo em um ambiente de produção.

Tecnicamente, o Tomcat é um *container Web*, parte da plataforma corporativa Java Enterprise Edition[32] (J2EE ou Java EE) que abrange as tecnologias Servlet e JSP, incluindo tecnologias de apoio relacionadas como Realms e segurança, JNDI Resources e JDBC DataSources (para maiores informações visite o site do Tomcat[33]). O Tomcat tem a capacidade de atuar também como servidor *web*/HTTP, ou pode funcionar integrado a um servidor *web* dedicado como o Apache httpd[34] ou o Microsoft IIS[35].

27 Site do Apache Jakarta: http://jakarta.apache.org/
28 Site da Sun: http://www.sun.com/
29 Site do Java Servlet: http://java.sun.com/products/servlet/
30 Site do JavaServer Pages: http://java.sun.com/products/jsp/
31 Site do Apache Software Foundation: http://www.apache.org/
32 Site do Java Enterprise Edition: http://java.sun.com/javaee/
33 Site do Tomcat: http://tomcat.apache.org/
34 Site do Apache httpd: http://httpd.apache.org/
35 Site do Microsoft IIS: http://www.iis.net/

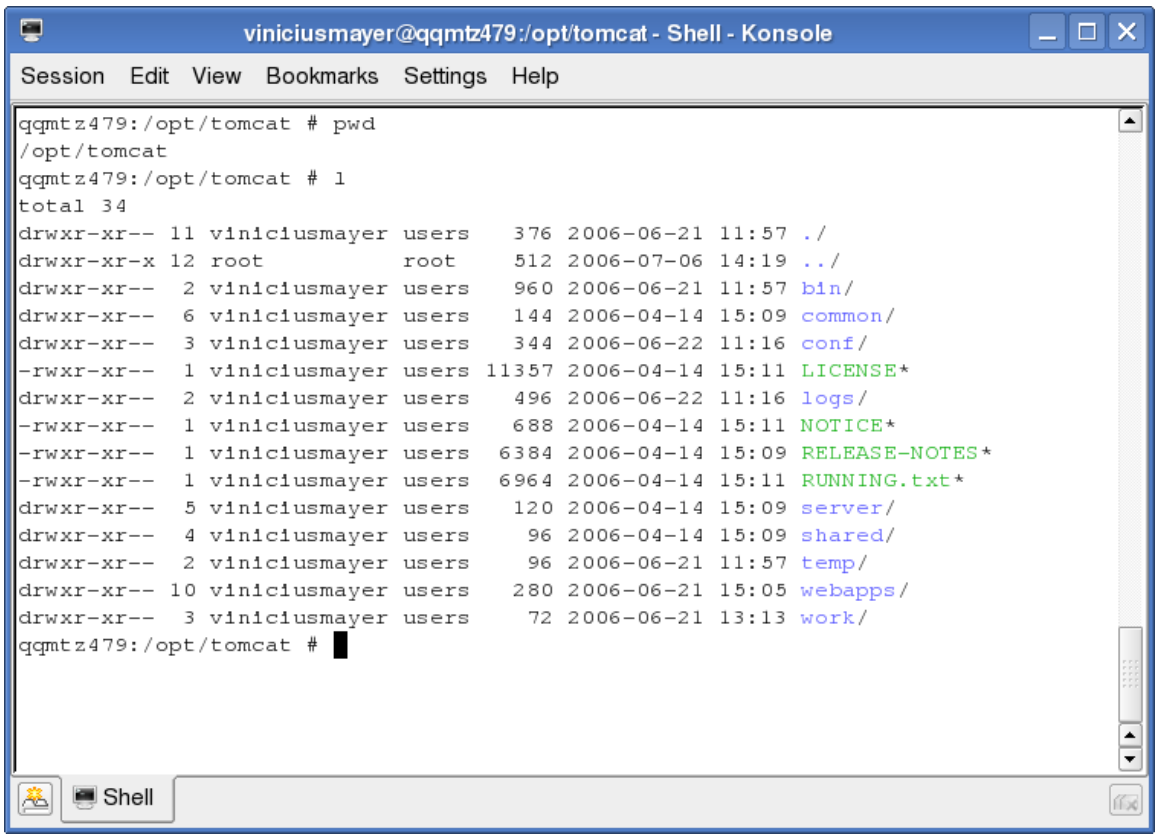

Figura 03. Screenshot da árvore de diretórios do Apache Tomcat.

Vamos analisar cada um destes diretórios separadamente:

- `bin`: Este diretório contém os arquivos binários executáveis. Na instalação da versão 5.5.9 para Windows você encontrará arquivos tais como `tomcat5.exe` e `tomcat5w.exe`. É comum nas versões para Linux, encontrarmos neste diretório os arquivos `.sh` necessários para a execução do servidor;
- `common`: Classes disponíveis tanto para o funcionamento interno do servidor quanto às diversas aplicações *web* instaladas. Geralmente este diretório contém subdiretórios chamados `classes` e `lib`. Classes colocadas nestes subdiretórios estarão disponíveis tanto para o Tomcat quanto para todas as aplicações *web* rodando no servidor; Classes já compiladas devem ser colocadas no diretório `classes` e arquivos JAR devem ser colocados no diretório `lib`. Lembre-se de que todas as classes e JARs do diretório `common` estarão acessíveis a todas as aplicações;
- `conf`: Contém arquivos de configuração. Dentre os vários arquivos neste diretório, os mais importantes são `server.xml`, `tomcat-users.xml` e `web.xml`. O arquivo `server.xml` é o arquivo de configuração principal do Tomcat. É usado para

configurar *logs,* conexões com outros servidores, a porta e *host* na qual o servidor está sendo executado e o local dos arquivos de cada aplicação *web*. O arquivo `tomcat-users.xml` é a base de dados padrão para a autenticação de usuários. Se um dia você esquecer a senha que você definiu na hora da instalação do Tomcat, é só abrir este arquivo e recuperá-la. O nome e local deste arquivo pode ser alterado no arquivo `server.xml`. O arquivo `web.xml` é o descritor de instalação (*deployment*) padrão para todas as aplicações *web*. O Tomcat processa este arquivo antes de processar os arquivos `web.xml` das aplicações *web* sendo instaladas;

- `logs` - O diretório *logs* é o local padrão para os arquivos de *logs*;
- `server` - O diretório server contém três subdiretórios: `classes`, `lib` e `webapps`. Estes diretórios são acessíveis apenas ao Tomcat, ou seja, classes colocadas no diretório classes do diretório `server` não estarão acessíveis a outras aplicações *web*. O diretório `webapps` é o local reservado para as aplicações de gerenciamento do Tomcat: *admin* e *manager*;
- `shared` - Este diretório contém dois subdiretórios: `classes` e `lib`. Classes nestes subdiretórios estarão disponíveis a todas as aplicações *web*, exceto ao Tomcat. Classes compiladas devem ser colocadas no diretório classes e arquivos JAR no diretório `lib`;
- `webapps` - Diretório padrão para a instalação de aplicações *web* no Tomcat;
- `work` - Este diretório é onde o Tomcat coloca o código da página JSP depois que este é convertido em um Servlet. Uma vez que uma página JSP é visitada pela primeira vez, o Tomcat armazenará o código compilado neste diretório também.

4.4 Java Web Start

Java Web Start é uma nova tecnologia para *deployment* de aplicações Java. Ela nos permite executar aplicações com um simples clique em uma página *Web*. O usuário pode baixar e executar aplicações sem passar por procedimentos complicados de instalação. Se a aplicação não estiver na máquina do cliente, o Java Web Start irá automaticamente baixar todos os arquivos necessários para a execução da aplicação. Se existir alguma versão da aplicação na máquina do cliente, esta aplicação estará sempre pronta para ser executada a qualquer hora que o usuário necessitar, através

de um ícone em seu *desktop* ou através de um *link* em um *Web Browser*. Independente da forma que a aplicação for invocada, uma versão mais recente da aplicação sempre estará presente para o usuário.

O Java Web Start utiliza a tecnologia Java Network Launching Protocol & API[36] (JNLP). A tecnologia JNLP define, além de outras coisas, um formato padrão de arquivos (arquivo JNLP) que descreve como a aplicação será executada.

Java Web Start suporta:

- Versionamento e atualização incremental;
- Operação *off-line*;
- Integração com *desktop*;
- *Sandboxing*, ambiente protegido com restrições de acesso a disco ou recurso de rede;
- *Code-signing*;
- Instalação automática de JRE e pacotes adicionais da aplicação.

O Java Web Start utiliza um arquivo JNLP que descreve como a aplicação será executada. Através de uma página *Web* com um *link* para este arquivos JNLP, o usuário com um simples clique ativa o Java Web Start. O Java Web Start irá então verificar se todos os recursos necessários para a execução da aplicação estão disponíveis na máquina do cliente, caso seja a primeira vez que a aplicação esteja sendo executada, o Java Web Start detectou que nenhuma versão da aplicação ou até mesmo da Java Runtime Environment[37] (JRE) estão presentes na máquina do cliente, neste caso ele irá primeiro fazer o *download* da JRE e depois da aplicação Java. Caso já exista uma versão da aplicação instalada no cliente e o Java Web Start identifique que houve alguma alteração da aplicação no servidor, imediatamente inicial o processo de atualização da versão mais recente da aplicação. Após todo o processo de verificação e atualização a aplicação é iniciada e o usuário já pode utilizá-la.

4.5 Servidor

36 Site do Java Web Start: http://java.sun.com/products/javawebstart/index.jsp
37 Downloado do Java Runtime Environment: http://www.java.com/en/download/manual.jsp

O servidor Web com o nome `gentoo.unijui.tche.br`, que estamos hospedando o site do projeto e, conforme já dito anteriormente, também é neste servidor que implementamos os principais servicos de disponilização do Gabrielus. Tanto que neste capitulo, e antes desta seção, fizemos uma breve apresentação deste serviços que utilizamos e agora, apresentamos a configuração realizada em cada um deste serviços.

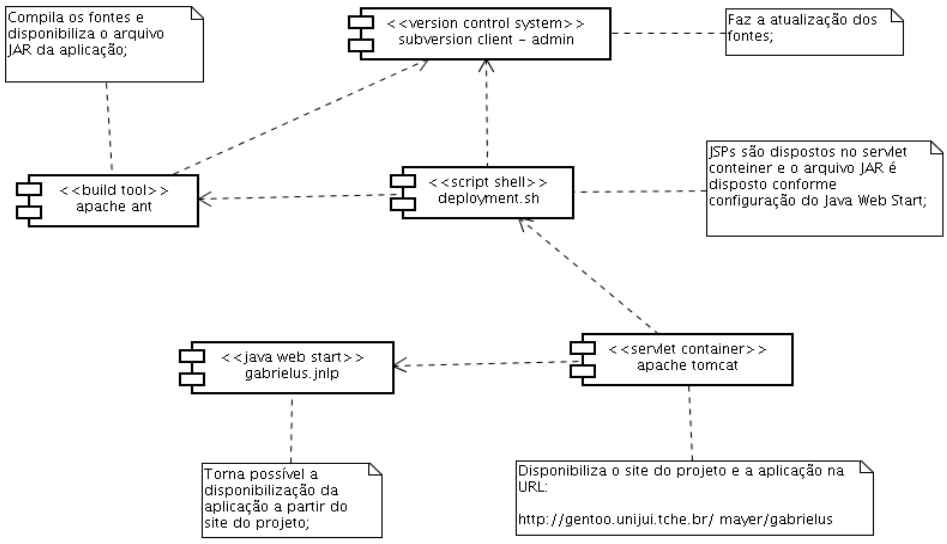

Diagrama 16. Diagrama de Componentes representando o conjunto de componentes e respectivas dependências para o Gabrielus.

No diagrama de componentes que apresentamos acima estão representados os servicos aos quais nos referimos anteriormente e, as dependências que existem entre eles. O processo todo é iniciado pelo Script Shell que elaboramos. Sendo que o objetivo principal do servidor `gentoo.unijui.tche.br` não é necessariamente a disponibilização do site em si e sim do aplicativo, consideramos que há dois momentos distintos ao estarmos trabalhando com este. Num primeiro momento estariamos atualizando o aplicativo e, num segundo momento, disponibilizando esta nova versão do Gabrielus.

Com o cliente subversion obteriamos as últimas versões dos fontes do Gabrielus no Java.Net, o Apache Ant faria a compilação e disposição dos arquivos Java relacionados ao projeto (`gabrielus.jar` – o aplicativo, `gabrielus.jnlp` – o arquivo para acesso via Java Web Start e; as bibliotecas `violet.jar` e `timingframework.jar`) nos seus devidos lugares e este seria o primeiro momento de uma atualização do aplicativo que é disposto no *site*. Em seguida viria o processo de

inicialização e/ou reinicialização do servidor *web* Apache Tomcat para disponibilização do *site* do projeto e, que é o mais importante, a disponibilização do Gabrielus via Java Web Start.

Para estas etapas/momentos da atualização do aplicativo que é disponibilizado via Java Web Start no *site* do projeto que criamos um Script Shell que aumotamiza para nós estes processos. Conforme é mostrado no diagrama de componentes acima, o *script* depende da carga dos fontes atualizados do Java.Net (`gabrielus.dev.java.net`) e do processamento realizado pelo apache ant para poder disponibilizar com sucesso uma nova versão do aplicativo no *site* do projeto. Vale lembrar que, uma atualização no *site* também depende de todo o processo envolvido com a atualização do aplicativo que é disposto no *site*, uma vez que o próprio site também está versionado em `gabrielus.dev.java.net` e é preciso carregar de lá a última versão dos arquivos que foram modificados.

4.5.1 Configuração do Subversion

Depois de baixado o arquivo subversion-1.3.2.tar.gz[38], vamos instalá-lo:

```
# tar -xzvf subversion-1.3.0.tar.gz
# cd subversion-1.3.0
# ./configure
# make
# make install
```

A instalação do Subversion é muito simples e convencional. Por padrão o cliente Subversion só irá pedir senha durante a primeira conexão com o servidor Subversion, pois o cliente do Subversion irá armazenar a senha e outras informações de autenticação em `~/.subversion/auth`. Isto cria possíveis pontos de exploração do servidor Subversion, para mudar isto edite o arquivo em `~/.subversion/config` e altere a seguinte linha de:

```
store-passwords = yes
```

Para:

```
store-passwords = no
```

38 Link para download: http://subversion.tigris.org/downloads/subversion-1.3.2.tar.gz

Caso a linha não exista ou esteja comentada (começando com #), basta acrecentar ela dentro da seção `[auth]`. Caso a conexão com o servidor Subversion seja atraves de um servidor *proxy*, edite o arquivo `~/.subversion/servers` e acrescente as seguintes linhas:

```
[global]
http-proxy-host = proxy.exemplo.com.br
http-proxy-port = 3128
```

Caso exista algum servidor Subversion que deva ser acessado sem servidor proxy, acrescente a linha abaixo ao arquivo dentro da seção `[global]`, colocando cada servidor separado por vírgula.

```
http-proxy-exceptions = localhost, 192.168.0.1
```

4.5.1.1 Ciclo básico de trabalho

Segue abaixo exemplos de uso das operações básicas a serem realizadas com relação ao controle de versionamento de código que temos na máquina local e o repositório que temos no Java.Net:

- Criar uma copia local do repositório (esta etapa só é necessária ser executada uma vez):

```
$ svn checkout http://svn.exemplo.com.br/svn/
```

- Atualizando a copia local do repositório:

```
$ svn update
```

- Enviando as alterações para o repositório:

```
$ svn commit
```

Podemos alterar nosso método de acesso de `http://` para `https://`, assim as transações com o servidor serão criptografadas. Para tanto, no servidor Subverion, o Apache HTTPd deve estar configurado para tratar conexões HTTPS. Caso o Apache2 esteja utilizando um certificado não assinado por uma certificadora autorizada será mostrada as informações do certificado e será necessário aceita-lo manualmente. Para isto responda `t` para aceitar temporariamente, a para aceitar definitivamente ou `r` para rejeitar definitivamente.

A opção de aceitar definitivamente o certificado só existe caso as informações de

autenticação estejam sendo gravadas, para mudar isto edite o arquivo em `~/.subversion/config` e altere a seguinte linha de:

```
store-auth-creds = no
```

Para:

```
store-auth-creds = yes
```

Caso a linha não exista ou esteja comentada (começando com #), basta acrecentar ela dentro da seção `[auth]`.

Quanto ao servidor Subversion do Gabrielus, que é mantido pelo Java.Net, para criar uma copia local do repositório, o seguinte comando deve ser executado:

```
svn checkout
https://gabrielus.dev.java.net/svn/gabrielus/trunk
gabrielus --username viniciusmayer
```

Onde:

- `svn checkout`: ação que você deseja executar. No caso, criar uma cópia local do projeto;
- `https://gabrielus.dev.java.net/svn/gabrielus/trunk`: *Link* para acesso direto ao servidor Subversion, observe que é usada conexão segura via protocolo https;
- `gabrielus`: O nome da pasta que será criada na máquina local com o conteúdo que será carregado do servidor Subverion e;
- `--username viniciusmayer`: O nome do usuário que estará conectando ao servidor Subversion para fazer uma cópia do projeto para a máquina local. Este acesso, ou melhor, este usuário é criado pelo mantenedor do projeto e, o acesso é exclusivo deste usuário que são criados e cujas permissões ao repositório Subversion são dadas.

4.5.2 Configuração do Ant

O Ant trabalha com arquivos XML[39] chamados de *buildfiles*, eles são interpretados pelo ANT, para que ele possa executar as tarefas que estão descritas nesses arquivos. O *buildfile* é um arquivo XML geralmente chamado de `build.xml`, este arquivo está normalmente organizado desta maneira:

```
<project>
     <propety1/>
     <propety2/>

     <target1>
          <task/>
     </target1>

     <target2>
          <task/>
     </target>
</project>
```

- Um **`project`** é a *tag* raiz do `build.xml`, ele representa todo o projeto e só pode existir um por *buildfile*;
- Um **`target`** é uma coleção de tarefas que desejamos aplicar em determinado momento e encadeando junto com outras tarefas;
- Um `task` é uma tarefa que desejamos que seja feita dentro do `target`, o Ant já disponibiliza tarefas prontas como:

```
<property name="build" value="bin" />
```

Para acessar o valor da `property` que foi criada:

```
...
<echo>Compilando classer para o diretório:
${build}</echo>
...
```

Isto faz o ant procurar o `build.xml` no diretório base e executa alvo *default*:

```
ant
```

39 XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. XML é um subtipo de SGML (Standard Generalized Markup Language - Linguagem Padronizada de Marcação Generica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet.

Executa o alvo *default* de `arquivo.xml`:

```
ant -buildfile arquivo.xml
```

Roda o alvo desejado e dependências relacionadas:

```
ant nome_do_alvo
```

Segue em seguida um exemplo de configuração do arquivos build.xml do nosso projeto:

```
<?xml version="1.0"?>
<project name="Gabrielus" default="compile">
    <target name="clean" description="remove arquivos
intermediarios">
        <delete dir="classes"/>
    </target>
    <target name="compile"
     description="compila o código fonte Java do
arquivo class">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile"
     description="cria o arquivo Jar para a aplicação">
        <jar destfile="gabrielus.jar">
            <fileset dir="classes"
includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class"
value="br.com.viniciusmayer.gabrielus.editor.DataFlowEd
itor"/>
            </manifest>
        </jar>
    </target>
</project>
```

Neste arquivo de exemplo de `build.xml` do Gabrielus, demostramos como podemos automatizar o processo de build do projeto usando o Ant. Podemos ver neste exemplo como é gerado o arquivo `.jar` que virá a ser o arquivo a ser acessado via Java Web Start pelo usuário que estiver navegando no *site*. Inclusive, neste exemplo de arquivo de *build* do Ant, não mostramos como o arquivo `.jnlp` que descreve com o Java Web Start deve agir para conseguir iniciar o

aplicativo é gerado. Até o momento tal funcionalidade não havia sido implementada e, portanto, estamos criando o arquivo `.jnlp` manualmente. Pretendemos, assim que possível, implementar mais esta funcionalidade e aproveitar ainda mais as facilidades que esta ferramenta (o Apache Ant).

4.5.3 Configuração do Script Shell

Antes de saber o que é um *script* em *shell* é importante saber antes o que é um Shell. Na linha de comandos de um *shell* podemos utilizar diversos comandos um após o outro, ou mesmo combiná-los numa mesma linha. Se colocarmos diversas linhas de comandos em um arquivo texto simples teremos em mãos um *Shell Script*, ou um *script* em *shell*, já que *Script* é uma descrição geral de qualquer programa escrito em linguagem interpretada, ou seja, não compilada.

Uma vez criado, um *ShellScript* pode ser reutilizado quantas vezes for necessário. Seu uso, portanto, é indicado na automação de tarefas que serão realizadas mais de uma vez. Todo sistema Unix e similares são repletos de *scripts* em *shell* para a realização das mais diversas atividades administrativas e de manutenção do sistema.

Os arquivos de lote (*batch* - arquivos `*.bat`) do windows são também exemplos de *ShellScripts*, já que são escritos em linguagem interpretada e executados por um *Shell* do Windows, em geral o `command.com` ou hoje em dia o `cmd.exe`. Os *Shells* do Unix, porém, são inumeras vezes mais poderosos que o interpretador de comandos do windows, podendo executar tarefas muito mais complexas e elaboradas.

A seguir apresentamos um fragmento do *script shell* criamos para automatiza a atualização tanto do *site* quanto da aplicação que estamos disponiblizando no *site* via o Java Web Start:

```
#!/bin/bash

clear

echo ""
echo "Desejas realmente fazer update do projeto
gabrileus?"
echo "[yes|no] ";
   read x
```

```
echo ""

case $x in

   yes | y)

      echo "Executando ação 'svn update gabrielus'..."
      echo "Por favor, aguarde!"
         svn update
/home/viniciusmayer/projects/gabrielus/
      echo ""
      echo "Ação 'svn update gabrielus' realizada com
sucesso!"
      echo ""
#aqui os demais processamentos programados
      echo ""
      echo "Update do projeto gabrielus concluida com
sucesso!"
      echo ""

   ;;

   no | n)
      echo "Cancelando update do projeto gabrielus!"
      echo "Ação 'update gabrielus' cancelada com
sucesso!"
      echo ""
   ;;

esac
```

No fragmento de *script* que apresentamos acima, estamos trabalhando apenas com o procedimento de carga das últimas versões dos códigos fonte do projeto no Java.Net. Observe que tal ação se dá pela ativação do comando `svn update` na pasta que mantém a cópia local dos fontes do projeto que estamos trabalhando. Alertamos apenas para o fato de que o *script* não trata necessariamente as exceções que podem por ventura acontecer. Por exemplo, poderiamos enfrentar algum tipo de conflito de versões entre algum arquivo de código fonte do projeto na máquina com um arquivo de código fonte no servidor subversion e, a única "ação" que teriamos por parte deste script e a exibição dos resultados do processamento feito. Logo, qualquer evento que fuja a um

ambiente ideial, terá de ser tratado manualmente.

4.5.4 Configuração do Tomcat

Consideremos agora nosso ambiente no servidor *Web* que irá disponibilizar a nossa aplicação. Para o Gabrielus estaremos utilizando como servidor Web o Jakarta Tomcat, mas vale lembrar que o Java Web Start funciona com qualquer servidor *Web*. Para o Java Web Start funcionar devemos:

- no arquivo `web.xml` que se encontra dentro da pasta `conf` do nosso servidor, verificar a existência do MIME-TYPE, JNLP, caso o servidor não possua esse MIME-TYPE, devemos adicionar as seguintes linhas:

```
<mime-mapping>
      <extension>jnlp</extension>
      <mime-type>application/x-java-jnlp-file</mime-
type>
</mime-mapping>
```

- no arquivo `server.xml` que também se encontra dentra da pasta `conf` do nosso servidor, devemos adicionar um novo contexto para a nossa aplicação. Inclua entre a *tag* `Host`, as seguintes linhas:

```
<Context path=”/gabrielus” docBase=”gabrielus”
debug=”0” reloadable=”true”/>
```

- devemos criar também dentro da pasta `webapps` do nosso servidor a pasta da nossa aplicação, que neste caso irá se chamar `gabrielus`. Após criar esta pasta, podemos copiar o arquivos JAR gerado anteriormente para dentro desta pasta;

4.5.5 Configuração do Java Web Start

O Java Network Lauching Protocol (JNLP) descreve como nossa aplicação será executada. Ela nada mais é do que um arquivo XML com a extensão `.jnlp`. Apesar de ser um arquivo XML é

imprescindível que a extensão seja `.jnlp`.

Abaixo o arquivo JNLP do Gabrielus:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp
  spec="1.0+"
  codebase="http://gentoo.unijui.tche.br/~mayer/gabriel
us/www"
  href="../jws/gabrielus.jnlp">
  <information>
    <title>Gabrielus</title>
    <vendor>E. Vinicius D. Mayer & Andre S.
Lemos</vendor>
    <homepage href="../www/index.html"/>
    <description>Gabrielus - DataFlow Diagrams
Editor</description>
    <description kynd="short">Gabrielus DataFlow
Editor</description>
    <icon href="../images/giv.png"/>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="1.5+"/>
    <jar href="../dist/Gabrielus.jar" main="true"/>
    <jar href="../lib/TimingFramework.jar"/>
    <jar href="../lib/Violet.jar"/>
    <property name="propriedade" value="exemplo de
propriedade"/>
  </resources>
  <application-desc main-
class="br.com.viniciusmayer.gabrielus.editor.DataFlowEd
itor"/>
</jnlp>
```

O arquivo JNLP acima possui as *tags* necessárias para o funcionamento do Java Web Start, portanto iremos ver mais detalhadamente todas as *tags* que um arquivos JNLP pode conter:

- Atributos do elemento `jnlp`:
  - `spec`: Por padrão o valor é “1.0+”, podendo ser versões 1.0 ou superiores do

release que está trabalhando;

- `codbase`: URL do servidor *Web* onde a aplicação deverá ser iniciada;
- `href`: URL que aponta a localização do próprio arquivo JNLP. O Java Web Start necessita deste atributo para identificar as características de execução de um aplicativo;

- Atributos do elemento `information`:
  - `title`: Nome da aplicação;
  - `vendor`: Nome do "dono" da aplicação;
  - `home page`: Contém um simples atributo, `href`, que aponta para a URL da página que inicia a aplicação. Esse elemento é usado pelo *Application Manager* para permitir ao usuário ir até a página da aplicação para obter mais informações;
  - `description`: Uma pequena descrição da aplicação. O elemento `description` é opcional. O atributo `kind` é utilizado para definir como a descrição deve ser usada. Ele pode ter os seguintes valores: `on-line, short, tooltip`;
  - `icon`: Contém o atributo `href`, onde informamos uma URL que aponta para uma imagem correspondente ao icone da aplicação. O atributo opcional `kind="splash"`, pode ser usado para informar que uma imagem será usada na *splash screen* na inicialização da aplicação;
  - `offline-allowed`: Este elemento é opcional e indica se a aplicação pode ser executada *off-line*; Se este elemento não for informado, a aplicação só poderá ser executada *on-line*, fazendo com que o Java Web Start sempre verifique a versão da aplicação para saber se deve ou não atualizá-la e depois executá-la. Agora, se este elemento for informado, o Java Web Start irá tentar verificar se existe uma nova atualização disponível, entretanto se já existir uma versão da aplicação na máquina do cliente, essa verificação irá terminar em poucos segundos e a aplicação existente no cliente será executada;
- Atributos do elemento `security`: Toda aplicação é por padrão executada em um ambiente restrito, similar ao *applet Sandbox*. O elemento `security` pode ser utilizado para solicitar acesso irrestrito. Se o elemento `all-permissions` for especificado, a aplicação terá acesso total a máquina do cliente e a rede, porém se uma aplicação solicita acesso total, todos os arquivos JAR deverão ser assinados[40].
- Atributos do elemento `resource`: O elemento `resource serve` para especificar

40 http://java.sun.com/developer/Books/javaprogramming/JAR/sign/signing.html

todos os recursos, como classes java, bibliotecas nativas e propriedades do sistema, que fazem parte da aplicação. A definição de um recurso pode ser restrita para um específico sistema operacional, arquitetura ou uso local dos atributos: `os, arch e locale`; Ele possui seis sub-elementos que são:

- `jar`: O elemento `jar` especifica um arquivo JAR utilizado pela aplicação;

```
<jar href="myjar.jar"/>
```

- `nativelib`: O elemento `nativelib` especifica um JAR que contém bibliotecas nativas. Exemplo:

```
<nativelib href="lib/windows/corelib.jar"/>
```

- `j2se`: O elemento `j2se` especifica qual a versão da JRE a aplicação suporta, bem como os parâmetros padrão para a JVM. O atributo `href` pode ser setado com a URL para o Java Web Start poder fazer o *download* da JRE caso o cliente não possua instalado, por padrão devemos sempre setar com a URL do site da Sun. Exemplo:

```
<j2ee version="1.5"
href="http://java.sun.com/products/autod1/j2se"/>
```

- `property`: O elemento `property` define uma propriedade do sistema que poderá ser acessado pelo método `getProperty` da classe `System` (`System.getProperty`). Este elemento requer dois atributos: `name` e `value`. Exemplo:

```
<property name="greeting" value="Olá Mundo!"/>
```

- Atributos do elemento `application-desc`: Esse elemento informa que o arquivo JNLP está executando uma aplicação e não uma Applet. Ele possui um atributo opcional, `main-class`, que é usado para especificar o nome da classe principal da aplicação, isto é, a classe que possui o método `public static void main(String args[])`. O atributo `main-class` pode ser omitido caso o primeiro JAR definido contenha o arquivos `Manifest` informando a classe principal. Argumentos podem ser especificados:

```
<application-desc main-class="Main">
     <argument>arg_um</argument>
     <argument>arg_dois</argument>
</application-desc>
```

- Atributos do elemento `applet-desc`: Java Web Start também possui suporte para execução de Applets Java. Exemplo:

```
<applet-desc
      documentBase="http://..."
      name="TimePilot"
      main-class="Time.Pilot.TimePilotApp"
      width="527"
      height="428">
            <param name="key1" value="value1"/>
            <param name="key1" value="value1"/>
</applet-desc>
```

4.6 Site

O *site* do projeto foi desenvolvido exclusivamente para a disponilização do Gabrielus via Java Web Start. Além é claro da tecnologia do Java Web Start, o site foi desenvolvido utilizando-se o que havia de melhor no que diz respeito aos padrões de desenvolvimento *web*, como por exemplo Tableless, CSS e *frameworks* que simplificam e padronizam tarefas um tanto quanto complicadas de se resolver padronizadamente a exemplo do arredondamentos de um *conteiner* do *site*. Abaixo apresentamos um *screenshot* do *site* do projeto, onde podemos ver a parte do cabeçalho com o acesso ao aplicativo via Java Web Start (clique na logo do Gabrielus), o menu do *site* e, o *conteiner* onde estará sendo disposto o conteúdo do menu acessado.

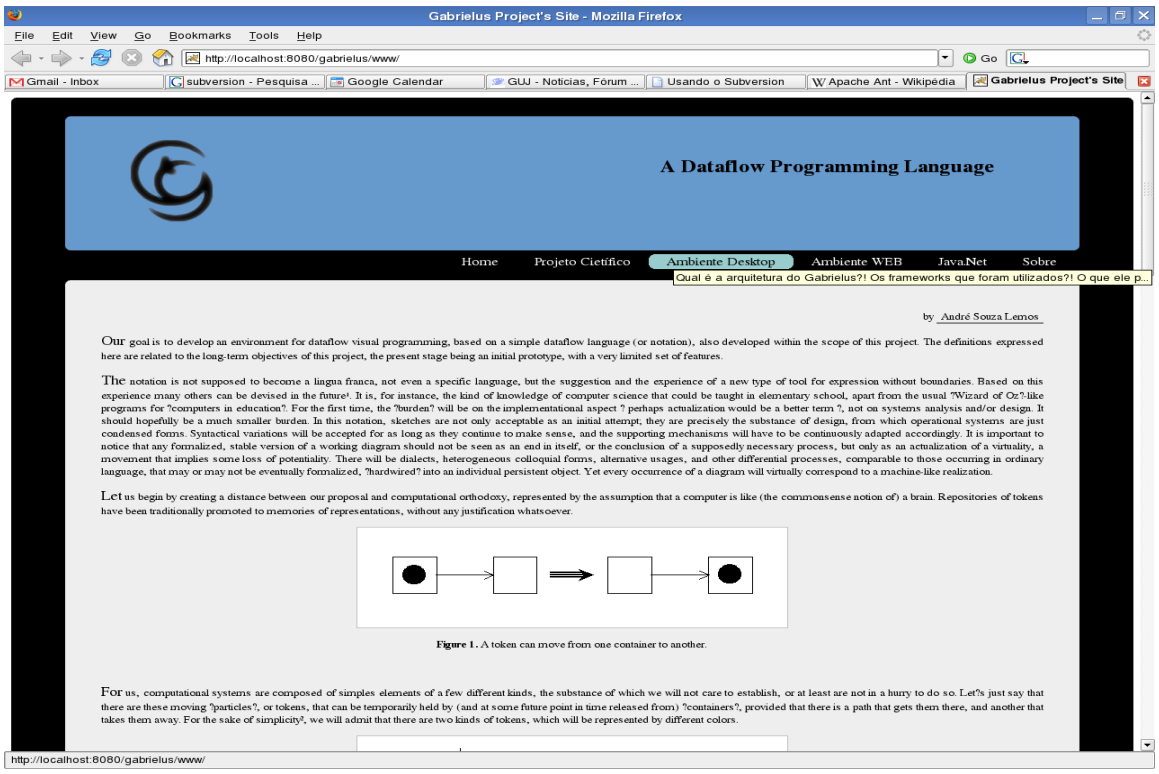

Figura 04. Screenshot "panorâmico" do site do projeto.

4.6.1 Tableless

Há uma controvérsia entre desenvolvedores na hora de definir o termo Tableless. Muitos acreditam que é somente uma característica de *sites* que não usam tabelas para *layout*, outros no entanto acreditam que Tableless faz parte de um conceito muito maior e que engloba os *web standards* em geral. Dentre as possíveis vantagens da metodologia, estão a diminuição do peso da página e melhora da acessibilidade, embora não são muitos os *sites tableless* hoje em dia que carregam essa segunda característica, pois poucos desenvolvedores captaram a mensagem corretamente, criando *sites tableless* distantes daquilo que engloba o Tableless, a padronização de tecnologias. Tableless seria somente a filosofia de construir *web-sites* sem tabelas, e não necessariamente seguindo os padrões?

Hoje em dia, falar simplesmente que Tableless é meramente uma filosofia, é extremamente errado, pois no código HTML as tabelas são para estruturar formulários e não para definir *layout*. E, uma vez considerado o "advento" da tecnologia, os *sites* estruturados em tabelas têm certo problema quanto à transferência de dados, geralmente quando se acessa um *site* que foi desenhado através das tabelas, o mesmo quebra e bagunça todo o conteúdo. Com Tableless é diferente, pois

todo o conteúdo, em geral, se mantem "limpo" e claro.

Na linguagem oficial do HTML, por exemplo, as *tags* `ul li`, surgiram para formar-se as listas. O que pretende se explicar aqui é que, cada *tag* tem sua funcionalidade. O desenvolver do *layout* fica em relação à CSS simplesmente. Antes, para alguns até hoje, para se trocar a cor da fonte era/é uma complicação, já estruturando o *site* em Tableless, a dificuldade é quase zero, pois basta mudar no CSS, por exemplo, `font: black;` por `font: blue;`.

Muitos desenvolvedores costumam criticar a linguagem (original) Tableless por conta da dificuldade que se encontra inicialmente para estruturar um *site*, todo em texto, no *layout* desejado, por CSS. E falar que os *sites* desenvolvidos em Tableless fogem ao padrão não é válido, pois todos ou a maioria dos códigos são válidos seguindo as recomendações da W3C[41].

4.6.2 CSS

Cascading Style Sheets, ou simplesmente CSS, são estilos para páginas *web* e envolvem um conceito inovador: possibilitam a mudança da aparência simultânea de todas as páginas relacionadas com o mesmo estilo. Ao invés de colocar a formatação dentro do código, o programador cria uma ligação (em inglês, *link*) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta portanto modificar apenas um arquivo. Exemplo:

```
/* comentário em css (igual à linguagem Java) */
body {
     font-family: Arial, Verdana, sans-serif;
     background-color: #FFF;
     margin: 5px 10px;
}
```

O código acima define fonte padrão `Arial`; caso não exista substitui por `Verdana` e; em últimos caso, se não existir, define a fonte `sans-serif`. Define também a cor de fundo do corpo da página e margens.

41 Site da W3C (Word Wide Web Consortium): http://www.w3.org/ ou em português algo a respeito da W3C na Wikipedia, http://pt.wikipedia.org/wiki/W3C;

Sua necessidade veio do fato do HTML aos poucos ter deixado de ser usado apenas para criação de conteúdo na *web*, e portanto havia uma mistura de formatação e conteúdo textual dentro do código de uma mesma página. Contudo, na criação de um grande portal, fica quase impossível manter uma identidade visual, bem como a produtividade do desenvolvedor. É nesse ponto que entra o CSS.

As especificações do CSS podem ser obtidas no *site* da W3C, um consórcio de diversas empresas que buscam estabelecer padrões para a internet.

4.6.3 Framework Nifty Corners Cube

Para quem trabalha com a dobradinha xHTML e CSS, sabe da "dor de cabeça" que é para arredondar os quatro cantos de um bloco/*container* de um *site*. No mínimo, você teria que usar quatro elementos, com um *background* para cada canto. Pois bem, existe um *framework* alternativo para resolver este probleminha do arredondamento dos cantos de um *conteiner* de um *site*. O *framework* que nos referimos é o Nifty Corners Cube[42], cujo funcionamento é muito interessante.

Em primeiro lugar, você não precisa de imagens para fazer o arredondamento. O seu funcionamento é um tanto quanto simples: no evento `onLoad` da janela é feito a chamada a função do Nifty. Exemplo:

```
<script type="text/javascript">
     window.onload=function(){
                Nifty("div#box","big");
     }
</script>
```

O primeiro parâmetro da função é o nome da camada onde qual será aplicado o efeito. O segundo parâmetro é uma palavra-chave que indica o tipo de arredondamento que será feito (vide tabela abaixo). No exemplo acima, aplicou-se o arredondamento de tamanho maior (`big`) na camada `#box`. Pode-se ainda aplicar o efeito em várias camadas, separando-as por vírgula. Da mesma forma, pode-se aplicar utilizando várias palavras-chaves, separando-as por espaço.

42 Site do Nifty Corners Cube: http://pro.html.it/niftycube/

Palavra-chave	Significado
`tl`	canto superior esquerdo
`tr`	canto superior direito
`bl`	canto inferior esquerdo
`br`	canto inferior direito
`top`	cantos superiores
`bottom`	cantos inferiores
`left`	cantos da esquerda
`right`	cantos da direita
`all (padrão)`	todos os cantos
`none`	nenhum dos cantos
`small`	tamanho dos cantos pequenos (2px)
`normal (padrão)`	tamanho dos cantos normal (5px)
`big`	tamanho dos cantos maior (10px)
`transparent`	aplica transparência do canto para o preenchimento interno (veja um exemplo)
`fixed-height`	para ser aplicado quando uma altura (height) é fixa e definida no CSS

Tabela 01. Relação de palavras-chave versus significado dos atributos do Nifty Corners Cube.

São inúmeras as configurações possíveis que o Nyfty Corners Cube nos proporciona e, numa última versão do *framework* que foi disponibilizada, vários efeitos foram adicionados, como por exemplo o de tranparência, abas e configuração de colunas. No *site* do *framework* é possível encontrar exemplos de uso de todas as funcionalidades do *framework*. Para maiores informações, visite o *site* do *framework* Nifty Corners Cube e deixe a criatividade fluir.

4.6.4 Versões Minimal, Standard e Full do Site

Foram criadas três versões do *site* do projeto, uma versão mínima, uma versão padrão e outra

completa. Na versão mínima, usou-se apenas HTML e desabilitados quaisquer tecnologias além desta, como por exemplo CSS e/ou JavaScript[43]. Também foi dispensado o uso do *framework* de arredondamento dos cantos, já que ele trabalha com as folhas de estilo CSS e JavaScript.

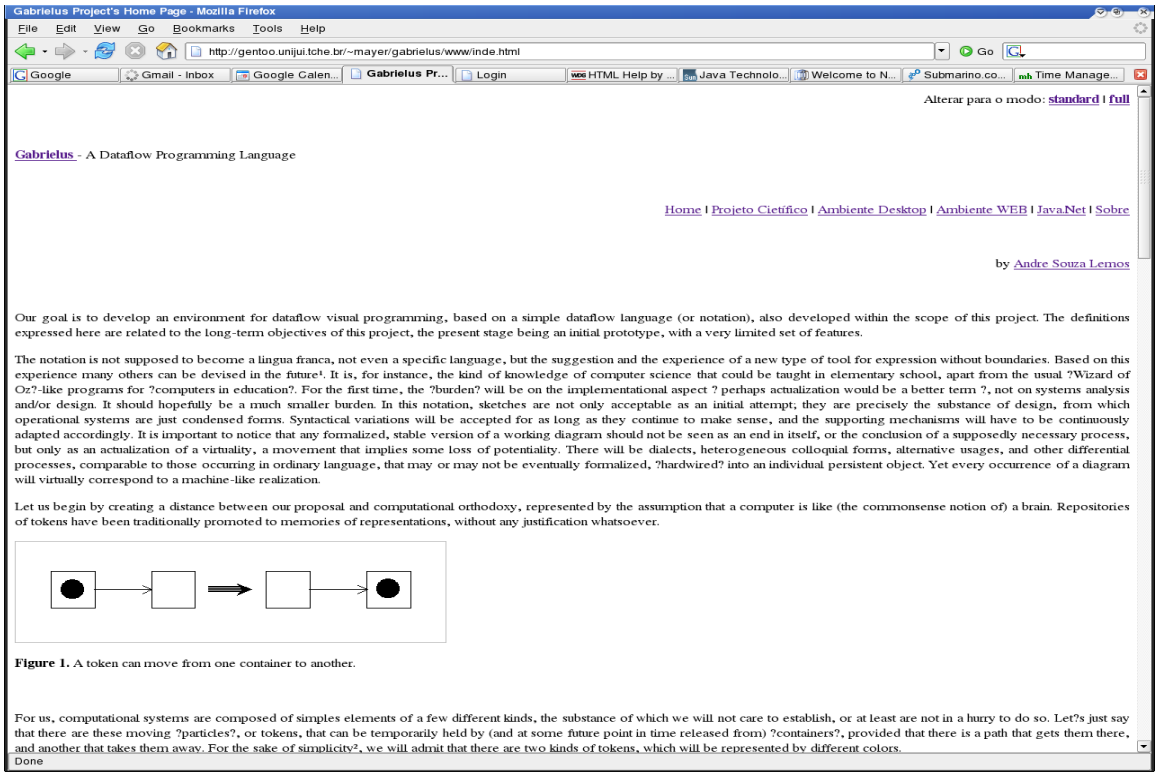

Figura 05. Site na versão mínima. Apenas HTML. CSS, JavaScript e Framework desabilitados.

Para a versão padrão, habilitaram-se as folhas de estilo CSS e continuam desabilitados o JavaScript e o *framework* que trabalha arredondado os cantos dos *conteiners* do *site*, já que ele depende do JavaScript para funcionar.

43 JavaScript é uma linguagem de programação criada pela Netscape em 1995, que a princípio se chamava LiveScript, para atender, principalmente, as seguintes necessidades: a) Validação de formulários no lado cliente (programa navegador) e; b) Interação com a página. Assim, foi feita como uma linguagem de script. Javascript que tem sintaxe semelhante a do Java, mas é totalmente diferente no conceito e no uso.

Figura 06. Site na versão padrão. Apenas CSS habilitado, JavaScript e Framework desabilitados.

E na versão completa, todas as tecnologias da camada *Web* foram habilitadas, as folhas de estilo CSS, o JavaScript e, então, o *framework* de arredondamento dos cantos dos *conteiners* do *site*.

Figura 07. Site na versão completa. CSS, JavaScript e Framework habilitados.

Estas opções foram criadas para que se pudesse observar no próprio site o quão modular e o quão "plugáveis" são as tecnologias utilizadas para a construção do mesmo. Na versão mínima, mostra-se o que de fato é o site. O conteúdo, no sentido do texto, não se altera em nenhuma versão e, mesmo na versão mínima do site, continuam sendo disponibilizadas as principais funcionalidades do site que são, propriamente dito, o conteúdo (textos), acesso aos documentos disponibilizados, vale lembrar da própria navegabilidade do *site* e, o mais importante, o acesso a aplicação Gabrielus via o link para o Java Web Start da mesma.

Na versão padrão faz-se apenas a importação dos arquivos de folha de estilo CSS. São então dadas as propriedades padrões para todo o *site*. Cor de fundo preta, *conteiners* de cabeçalho e rodapé azul, ação de fundo verde para a ativação de um *link* nestes *conteiners* e, para o *conteiner* principal do *site*, onde será disposto o conteúdo das páginas do *site*, com a cor cinza-claro e texto preto. Para este *conteiner* a ação da ativação de um *link* se diferencia dos demais *conteiners* de cabeçalho e rodapé, adotandos-se a cor cinza-escuro e bordas pretas.

E, por último, a versão completa carrega também o *framework* que faz o arredondamento

dos cantos dos *conteiners* do *site*. Tanto o *conteiner* de cabeçalho e rodapé, quanto o *conteiner* principal e o próprio *conteiner* do *site* (o *conteiner* definido com a cor preta e que envolve todos os demais *conteiners* do *site*) tem seus cantos arredondados assim como também as ações para os *links* que existem no cabeçalho e rodapé. Para as ações quando da ativação de um *link* no *conteiner* principal, estes foram mantidos da forma padrão, com fundo cinza-escuro e bordas pretas (sem cantos arredondados).

5 Conclusão

Apresentamos portanto, no decorrer do texto deste trabalho, grande parte do todo desenvolvido neste projeto de pesquisa e, com grande satisfação que chegamos a conclusão de que os objetivos e metas traçados para este projeto foram todos alcançados e, ainda, que superamos algumas de nossas expectativas. Ficam, é claro, inúmeros "trabalhos futuros" e a certeza de que muito trabalho ainda há de ser feito.

Como trabalhos futuros, teríamos algumas tarefas que dizem respeito a três escopos diferentes. Primeiro, os que poderiam dizer respeito a este trabalho escrito seriam: a) escrever uma espécie de Guia do Desenvolvedor, ou seja, como é que a pessoa que estará se envolvendo com o desenvolvimento do Gabrielus faz para conseguir trabalhar efetivamente? Quais as ferramentas que se fazem necessárias e onde ele consegue encontrá-las? E como instalá-las e plugar os projetos do Gabrielus nestas ferramentas? b) escrever uma espécie de Manual do Sistema, para a pessoa que for usar o Gabrielus no desenvolvimento de alguma tarefa, trabalho e/ou atividade que diz respeito ao que trata o Gabrielus. Como é que eu salvo um diagrama? Como recupero este diagrama? Como faço para gerar uma imagem do diagrama que desenhei? E perguntas deste escopo que poderiam ser esclarecidas para o usuário, propriamente dito, do Gabrielus. Já como forma de orientação, sugeriria que se usasse o JavaHelp[44] para o desenvolvimento deste "Manual do Sistema". O JavaHelp é um projeto Java da Sun que está disponível para *download* no *site* do projeto deste aplicativo no Java.Net[45].

Segundo, que diz respeito ao desenvolvimento do *site* do Gabrielus, o menu ficou a

44 Site do JavaHelp: http://java.sun.com/products/javahelp/
45 Site do Projeto JavaHelp no Java.Net: https://javahelp.dev.java.net/

princípio sem funcionalidade. Teríamos então duas alternativas para construírmos o mecanismo do menu do *site* do Gabrielus: a) Tiles[46] do Struts[47]: O Tiles é o mecanismo de *templates* do Struts, e torna possível reutilizar *layouts* de páginas de forma estruturada. O Tiles vem integrado e é fácil de usar com o Struts, ficando acessível através de uma *taglib* embutida. Sites baseados no Tiles são montados com peças, ou "ladrilhos". Assim, você pode modelar uma vez e replicar o *layout* em todo um grupo de páginas, ou mesmo em todas as páginas, de sua aplicação [Silva, 2006]. E; b) AJAX[48]: é o uso sistemático de JavaScript e XML (e derivados) para tornar a navegação mais interativo com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações *web* mais dinâmicas e criativas. AJAX não é uma tecnologia, são realmente várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades.

Destas duas tecnologias, Tiles e AJAX, o Tiles seria então uma tecnologia mais voltada para *sites* de grande porte, no nível de portais (a exemplo do próprio *site* da UNIJUI[49]) e também do nível de um *e-commerce* por exemplo (a estilo do Submarino[50]). Já o AJAX é uma tecnologia que, no nosso entendimento, está mais perto da realidade do *site* do nosso projeto que estamos desenvolvendo e, portanto, sugerimos que seja usada tal tecnologia para o desenvolvimento do mecanismo do menu do *site* do Gabrielus. Vale também dizer a idéia do AJAX é utilizar JavaScript para transformar suas páginas em aplicações, de modo que não precise recarregar a tela cada vez que o usuário clicar em alguma coisa. Você pode recarregar apenas a área que precisa ser alterada pela ação realizada.

E, por último, o terceiro escopo de tarefas futuras a serem realizadas, dizem respeito especificamente ao próprio Gabrielus, ou seja, o aplicativo *desktop*. Foi desenvolvida toda um estrutura com relação aos objetos que estarão sendo disponibilizados nos diagramas *dataflow* e, apenas alguns objetos foram desenvolvidos com forma de demostração. Portanto, resta ainda desenvolver a parte visual dos demais objetos que serão disponibilizados para a elaboração de um diagrama do tipo *dataflow*[51]. Quanto as animações, está também toda estruturada a parte que cuida do comportamento das animações no Gabrielus e, resta implementar a interface do usuário com as

46 Site do Tiles: http://struts.apache.org/1.x/struts-tiles/index.html
47 Site do Struts: http://struts.apache.org/
48 AJAX: acrónimo em língua inglesa de Asyncronous Javascript And XML; Site do AJAX: http://www.tableless.com.br/artigos/ajaxdemo/
49 Site da UNIJUI: http://www.unijui.tche.br
50 Site do Submarino: http://www.submarino.com.br
51 Você poderá encontrar mais referente aos objetos que estarão sendo disponibilizados nos Diagramas DataFlow em [MAYER];

animações propriamente ditas.

Também poderíamos pensar no seguinte, o que faz um novo membro da equipe de desenvolvimento do Gabrielus para estar apto a trabalhar no desenvolvimento do projeto? Poderíamos elaborar um passo-a-passo do desenvolvedor (em inglês, *developer how-to*) de como deverá proceder este novo integrante da equipe para estar pronto para ajudar no desenvolvimento do Gabrielus. Perguntas do tipo, como obter uma cópia do projeto lá no Java.Net, como conectar o projeto na ferramenta de desenvolvimento, como manter o repositório local (a cópia do projeto) e remoto (repositório do projeto no Java.Net) atualizado, entre outras ação básicas do desenvolvedor, poderiam então estar sendo tratadas neste documento.

Especificada toda a arquitetura do Gabrielus, deste a arquitetura *desktop* até arquitetura *web*, e dito como fazer para participar do desenvolvimento do Gabrielus, há de ser escrito um Estudo de Caso, para mostrar como é que um usuário acessa o Gabrielus e como utilizá-lo, agora numa espécie de passo-a-passo do usuário (em inglês, *user how-to)* bem simples do aplicativo. Um diagrama fechado cuja execução é cíclica poderia ser adotado como referência e já com o Gabrielus "em mãos", para exemplificar, mostraríamos como construir este diagrama básico.

6 Referências

SEBESTA, Robert W.; *Conceitos de Linguagens de Programação;* Bookman; Porto Alegre; 2000;

FILHO, Antonio Mende da Silva; *Introdução à Programação Orientada a Objetos*; In: Revista Espaço Acadêmico; No. 35; Abril de 2004;

VOSS, Greg; *Object-Oriented Programming: An Introduction;* McGraw-Hill; 1991; Apud: http://www.training.com.br/lpmaia/pub_prog_oo.htm; Último acesso em Julho de 2006;

RUMBAUCH, James; BLAHA, Michael; PREMERLANI, Willian; EDDY, Frederick; LORENSEN, Willian; *Modelagem e Projetos Baseados em Objetos*; Editora Campus; Rio de Janeiro, 1994;

GOSLING, James; MCGILTON, Henry; *The Java Language Environment*; White Paper; 1996; In: http://java.sun.com/doc/language_environment; Último acesso em Julho de 2006;

HORSTMANN, Cay S.; CORNELL, Gary; *Core Java: Volume I – Fundamentos;* Pearson Makron Books; São Paulo, 2003;

HORSTMANN, Cay S.; CORNELL, Gary; *Core Java: Volume II – Recursos Avançados;* Pearson Makron Books; São Paulo; 2004;

MATHIAS, Eder; *Introdução à Linguagem de Programação Java;* UNIJUÍ – Departamento de Tecnologia; 2004;

LEMOS, André S.; *A Maquina Criança – O Ensino Fundamental em uma Cultura Tecnológica;* In: Contexto Educação – Revista de Educación em América Latina y el Caribe: Ciências, Tecnologia e Educação; Ano 18, No. 69, p. 85-112; Unijuí; 2003;

JOHNSTON, Wesley M.; HANNA, J. R. Paul; MILLIAR, Richard J.; *Advanced in DataFlow Programming Languages;* In: ACM Computing Surveys*;* Vol. 36, No. 1, p. 1-34; University of Ulster; 2004;

ZANIN, Fabio A.; TIBOLA, Leandro R.; LANGSCH, Carla W. K.; *CODE***;** UFRGS – Instituto de Informática (Programa de Pós-Graduação em Computação), 1998;

PILLA, Maurício L.; *Um Simulador de Arquitetura DataFlow;* UFRGS – Instituto de Informática (Programa de Pós-Graduação em Computação), 1999;

RUMBAUCH, James; BOOCH, Grady; JACOBSON, Ivar; *UML – Guia do Usuário;* Campus, 2000; Rio de Janeiro;

CERUZZI, Paul E.; *A history of modern computing*; Cambridge, Massachusetts; MIT Press; 1998.

NARDI, Bonnie A.; *A small matter of programming: perspectives on end user computing;*. Cambridge, Massachusetts; MIT Press; 1993.

TANENBAUM, Andrew S.; *Sistemas Operacionais Modernos;* LTC Editora; Rio de Janeiro; 1992;

SILVA, Edgar A.; *Struts em Pedaços: Reutilize Design em JSPs com Tiles;* JavaMagazine; Edição 19, Ano III;

MAYER, E. Vinicius D.; *Protótipo de Núcleo de Ambiente de Programação Visual DataFlow;* Trabalho apresentado como requisito à aprovação na disciplina de Laboratório de Sistemas I sob orientação do Prof. André Souza Lemos; DETEC – Departamento de Tecnologia, UNIJUI – Universidade Regional do Noroeste do Estado do Rio Grande do Sul; Junho de 2005;

STEIL, Rafael; *Introdução a programação gráfica em Java com Swing;*

http://www.guj.com.br/java.tutorial.artigo.38.1.guj; Último acesso em Julho de 2006;

HAASE, Chet; *Timing is Everything*; http://today.java.net/lpt/a/168; Último acesso em Julho de 2006;

HAASE, Chet; *Time Again*; http://today.java.net/lpt/a/274; Último acesso em Julho de 2006;

WIKIPEDIA: Enciclopédia On-Line Livre; http://pt.wikipedia.org/; Último acesso em Julho de 2006;

Espaço para Anotações