UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA CONVERSÃO DE INTERFACES JAVA SWING EM HTML E JAVASCRIPT

MARCIELE FERNANDA SEVERO

BLUMENAU
2008

2008/2-14

MARCIELE FERNANDA SEVERO

FERRAMENTA PARA CONVERSÃO DE INTERFACES JAVA SWING EM HTML E JAVASCRIPT

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Profa. Joyce Martins, Mestre - Orientadora

BLUMENAU
2008

2008/2-14

FERRAMENTA PARA CONVERSÃO DE INTERFACES JAVA SWING EM HTML E JAVASCRIPT

Por

MARCIELE FERNANDA SEVERO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB
Membro:	Prof. Adilson Vahldick, Mestre – FURB

Blumenau, 11 de fevereiro de 2009

Dedico este trabalho aos amigos que me ajudaram diretamente na realização deste e a minha família que sempre me deu apoio.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que sempre me deu apoio.

Aos meus amigos, pelos empurrões e cobranças.

À minha orientadora, Joyce Martins, pela excelente orientação.

O sucesso geralmente vem para aqueles que estão muito ocupados para estarem procurando por ele.

Henry David Thoreau

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta para conversão de componentes de interfaces Java Swing em componentes equivalentes em HTML. A ferramenta converte, dentre todos os componentes existentes, apenas os 14 mais utilizados. Também são convertidos alguns tratadores de eventos para JavaScript. Tem-se como entrada da ferramenta um arquivo de código fonte Java, contendo todas as informações necessárias para a conversão em HTML e JavaScript. No desenvolvimento da ferramenta utilizou-se análises léxica, sintática e semântica para identificação dos *tokens*, das estruturas sintáticas e das informações que devem ser extraídas para geração do código de saída. A saída é gerada a partir de *templates*, interpretados pelo motor de *templates* Velocity.

Palavras-chave: Java Swing. HTML. *Templates*. JavaScript.

ABSTRACT

This work presents the development of the tool for conversion of components of interfaces Java Swing in components equivalents in HTML. The tool does not convert all the existing components, only the most used ones. It is also converted some events handling for JavaScript. It has as an entrance of tool a file with extension .java, containing all the necessary information for the conversion in HTML and JavaScript. In the development of the tool it was used lexica, syntactic and semantic analysis for identification of the tokens, of the syntactic structures and of the informations that must be extracted to generates the exiting code. The exit is generated of templates, interpreted by the engine of templates Velocity.

Keywords: Java Swing. HTML. *Templates*. JavaScript.

LISTA DE ILUSTRAÇÕES

Quadro 1 – Código fonte Java 17
Figura 1 – Exemplo de interface 18
Quadro 2 – Código HTML 19
Figura 2 – Exemplo de interface 20
Quadro 3 – Elementos da linguagem de *templates* Velocity 22
Quadro 4 – Exemplo de *template* Velocity com diretiva (`#parse`) 22
Quadro 5 – Exemplo de *template* Velocity (`templateJButton.vm`) 23
Quadro 6 – Exemplo de criação e inicialização do Velocity 23
Quadro 7 – Exemplo de criação de contexto 23
Quadro 8 – Exemplo de definição do *template* a ser utilizado 23
Quadro 9 – Exemplo de passagem de valores às variáveis 23
Quadro 10 – Exemplo de uso do método `merge` 23
Figura 3 – Ferramenta AjaxSwing 24
Figura 4 – Interface Java 25
Figura 5 – Interface convertida pela ferramenta AjaxSwing 25
Figura 6 – Ferramenta DelphiToWeb 26
Figura 7 – Interface gerada pela ferramenta DelphiToWeb 27
Figura 8 – Interface .DFM 27
Figura 9 – Ferramenta Converte Forms 28
Figura 10 – Interface Oracle Forms 28
Figura 11 – Interface convertida pela ferramenta Converte Forms 29
Quadro 11- Requisitos funcionais 31
Quadro 12 – Requisitos não funcionais 31
Quadro 13 – Componentes de interface 31
Quadro 14 – Tratadores de eventos 32
Quadro 15 – Mapeamento eventos Java X eventos JavaScript 32
Quadro 16 – Mapeamento componentes de interface Java Swing X componentes de interface HTML 33
Figura 12 – Diagrama de casos de uso 34
Quadro 17 – Detalhamento do caso de uso `UC01 - Especificar arquivo` 34
Quadro 18 – Detalhamento do caso de uso `UC02 - Especificar template` 35

Quadro 19 – Detalhamento do caso de uso `UC03 - Converter código` 35
F igura 13 – Diagrama de classes para componentes de interface Java Swing..... 36
Figura 14 – Diagrama de classes para geração de código..... 37
Figura 15 – Diagrama de seqüência do caso de uso `Converter código` 38
Quadro 20 – Conversão de código 40
Figura 16 – Tela da ferramenta SwingToHtml..... 41
Figura 17 – Seleção de arquivo `.java` 42
Figura 18 – Arquivos `.java` selecionados 42
Figura 19 – Seleção de *template*..... 43
Figura 20 – Conversão de arquivos 43
Figura 21 – Tela Sobre a ferramenta SwingToHtml 44
Figura 22 – Interface Java Swing 44
Figura 23 – Interface HTML 44
Figura 24 – Interface da calculadora em Java Swing 45
Figura 25 – Interface da Calculadora em HTML 45
Quadro 21 – Eventos em `.java` 45
Quadro 22 – Eventos em JavaScript..... 46
Quadro 23 – *Template* utilizado na conversão da calculadora 47
Quadro 24 – *Template* `templateJbutton.vm` 47
Quadro 25 – Comparação SwingToHtml X trabalhos correlatos..... 48
Quadro 26 – Código Java: Calculadora..... 57
Quadro 27 – Código HTML e JavaScript gerado: Calculadora 60
Quadro 28 – Atributos e métodos..... 63
Quadro 29 – Métodos 66
Quadro 30 – *Template* com todos os componentes 69
Quadro 31 – Gramática do Java 5.0 72

LISTA DE SIGLAS

API – *Application Programming Interface*

AJAX – *Asynchronous JavaScript And XML*

AWT – *Abstract Windowing Toolkit*

BNF – *Backus-Naur Form*

CSS – *Cascading Style Sheets*

EA – *Enterprise Architect*

HTML – *HyperText Markup Language*

JSP – *Java Server Pages*

LGPL – *Lesser General Public License*

MVC – *Model-View-Controller*

PL/SQL – *Procedural Language / Structured Query Language*

UML – *Unified Modeling Language*

VTL – Velocity Template Language

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 GERADORES DE CÓDIGO 15
2.2 COMPONENTES DE INTERFACE E TRATADORES DE EVENTO EM JAVA 16
2.3 HTML E JAVASCRIPT 18
2.4 MOTOR DE *TEMPLATES* VELOCITY 20
2.5 TRABALHOS CORRELATOS 23
2.5.1 AjaxSwing 24
2.5.2 DelphiToWeb 26
2.5.3 Converte Forms 27
3 DESENVOLVIMENTO 30
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 31
3.2 COMPONENTES DE INTERFACE E EVENTOS CONVERTIDOS 31
3.3 CÓDIGO DE SAÍDA 32
3.4 ANÁLISE DO CÓDIGO DE ENTRADA 33
3.5 ESPECIFICAÇÃO 34
3.5.1 Casos de uso 34
3.5.2 Diagrama de classes 35
3.5.3 Diagrama de seqüência 38
3.6 IMPLEMENTAÇÃO 38
3.6.1 Técnicas e ferramentas utilizadas 39
3.6.2 Implementação da ferramenta 39
3.6.3 Operacionalidade da implementação 41
3.7 RESULTADOS E DISCUSSÃO 48
4 CONCLUSÕES 50
4.1 EXTENSÕES 51
REFERÊNCIAS BIBLIOGRÁFICAS 52
APÊNDICE A – Código de entrada e saída: Calculadora 54
APÊNDICE B – Atributos e métodos 61

APÊNDICE C – *Template* com todos os componentes 67
ANEXO A – Gramática do Java 5.0 70

1 INTRODUÇÃO

A dependência das informações torna o uso dos sistemas de informação essencial para a sobrevivência das organizações, que ultimamente, conforme afirma Souza (2005, p. 10), têm demanda por sistemas para web. Uma das vantagens oferecidas pelas aplicações web reside no fato de não exigir a instalação da aplicação nas estações de trabalho dos usuários, que necessitam apenas de um navegador. Como as aplicações são instaladas em um servidor, o desenvolvimento, a manutenção e a atualização são centralizados (MACORATTI, 2006). Além da demanda pelo desenvolvimento de sistemas computacionais para web, "as empresas ainda possuem várias aplicações que foram desenvolvidas para *desktop*" (SOUZA, 2005, p. 10), que não aproveitam a infra–estrutura de comunicação da web e, às vezes, não têm interface gráfica. Isso tem levado as empresas de software a migrarem seus sistemas *desktop* utilizando tradutores ou geradores automáticos de código.

Conforme Sommerville (2003, p. 536), o meio mais simples de proceder à reengenharia de software é a tradução do programa, quando o código-fonte em uma linguagem de programação é automaticamente traduzido para o código-fonte em uma outra linguagem. Sommerville (2003, p. 536) afirma ainda que a estrutura e a organização do programa em si permanecem inalteradas, sendo que a linguagem-alvo pode ser uma versão atualizada da linguagem original ou pode ser uma tradução para uma linguagem completamente diferente. Levando em consideração o custo exigido para migrar uma aplicação, a falta de profissionais e a falta de padrões na codificação dos sistemas, o que muitas vezes produz códigos ilegíveis, o uso de uma ferramenta automática para migração de sistemas tem se tornado uma alternativa para economia de tempo e dinheiro. Conforme observa Soares Filho (2003), estas ferramentas "não fazem 100% do trabalho automaticamente, mas conseguem traduzir de 80% a 90% da antiga plataforma, restando aos programadores um trabalho de adaptação que representa 10% a 20% do total do código".

Diante do exposto, propõe-se o desenvolvimento de uma ferramenta que auxilie a conversão de interfaces de aplicações desenvolvidas em Java para páginas HTML. A ferramenta proposta deverá também converter tratamentos de eventos de interface para a linguagem JavaScript.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é construir uma ferramenta para conversão de interfaces de aplicações Java Swing em páginas HTML e eventos para JavaScript e JSP.

Os objetivos específicos do trabalho são:

a) converter alguns componentes de interface Swing, tais como `JFrame`, `JPanel`, `JLabel`, `JButton`, `JTextField`, `JTextArea`, `JList`, `JComboBox`, `JCheckBox`, `JRadioButton`, `JMenu`, `JMenuItem`, `JMenuBar` e `JTabbedPane`, mantendo o *layout* original;

b) converter tratamentos de eventos de interface;

c) utilizar *templates* para configurar o formato do código de saída;

d) analisar o grau de compatibilidade entre o código gerado e o código de entrada da ferramenta.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em 4 capítulos. O próximo capítulo apresenta a fundamentação teórica, contextualizando os temas abordados no desenvolvimento do trabalho, tais como: geradores de código, componentes de interface e tratadores de evento em Java, HTML e JavaScript, motor de *templates* Velocity e trabalhos correlatos. No terceiro capítulo é apresentado o desenvolvimento da ferramenta, contendo desde a especificação até a operacionalidade. O último capítulo traz os resultados alcançados com o desenvolvimento do trabalho, bem como sugestões de extensões para a ferramenta.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos relacionados ao trabalho. Trata de geradores de código e as etapas para o seu desenvolvimento. Na seqüência apresenta componentes de interfaces e tratamento de eventos em Java, que constituem a entrada para a ferramenta desenvolvida. Aborda ainda HTML e JavaScript, saída do gerador de código, bem como o motor de *templates* Velocity, utilizado para configurar a saída. Por último, relata as principais características dos trabalhos correlatos, os quais constituem a base para a criação da ferramenta desenvolvida.

2.1 GERADORES DE CÓDIGO

Geradores de código são ferramentas que a partir de uma entrada geram os arquivos fontes para uma determinada aplicação. Podem ser desde simples formatadores de código até ferramentas que geram aplicações complexas a partir de modelos abstratos, auxiliando no processo de desenvolvimento de software e proporcionando ganhos de produtividade e redução de tempo e de custos.

> Durante muito tempo, ainda, vamos ter programadores escrevendo código fonte. Mas não há dúvida de que uma parte cada vez maior da tarefa − a parte menos criativa e mais repetitiva - será construída de forma automática a partir da especificação, por um ou mais tipos de geradores. Os programadores terão que subir degraus na escada da competência e trabalhar em problemas mais abstratos do que as declarações de variáveis e trechos de código que são obrigados a repetir dia após dia, hoje. (MEIRA, 2004).

Para facilitar o desenvolvimento de geradores de código, Herrington (2003, p. 77) sugere que as seguintes etapas devem ser seguidas:

a) construir o código de saída: deve-se construir o código de saída manualmente, para determinar o que e como deve ser a saída do gerador. Esta etapa facilita a visualização do que deve ser extraído dos arquivos de entrada;
b) projetar o gerador: deve-se determinar qual será o formato de entrada, como a entrada será analisada e como o código de saída será gerado, usando *templates* ou simplesmente escrevendo o código;
c) analisar a entrada: deve-se implementar o código para ler o arquivo de entrada e

extrair as informações necessárias para construir a saída;

d) especificar *templates*: uma vez que a análise da entrada está definida, o próximo passo é criar os *templates* que serão utilizados para gerar a saída;
e) gerar a saída: este é o último passo, onde deve ser implementada a geração de código utilizando os *templates* especificados na etapa anterior.

Para a realização da etapa (c), conforme afirma Schvepe (2006, p. 20), podem ser usados analisadores léxico, sintático e semântico. O analisador léxico tem como objetivo identificar seqüências de caracteres que formam as unidades léxicas (*tokens*). Para tanto, lê o programa fonte da esquerda para a direita, caractere a caractere, verificando se os caracteres lidos fazem parte da linguagem, identificando os *tokens* e enviando-os para o analisador sintático. Esse recebe os *tokens* e agrupa-os em frases gramaticais, que são usadas pelo compilador para sintetizar a saída. Já o analisador semântico tem a função de verificar o significado da estrutura hierárquica determinada pela fase de análise sintática.

2.2 COMPONENTES DE INTERFACE E TRATADORES DE EVENTO EM JAVA

Java é uma linguagem de programação orientada a objeto com sintaxe similar à da linguagem C++, independente de plataforma, que foi anunciada formalmente pela Sun em 1995 (DEITEL; DEITEL, 2003, p. 59). Java fornece um extenso conjunto de bibliotecas, conhecidas também como API. Para a implementação das interfaces gráficas podem ser usadas duas APIs: AWT e Swing.

Nas primeiras versões da linguagem Java, segundo Steil (2006a), a única forma de desenvolver interfaces gráficas era através da AWT, uma biblioteca de baixo nível que depende do código nativo da plataforma onde a aplicação roda. Isso traz alguns problemas de compatibilidade entre as plataformas, sendo que nem sempre a aplicação tem a mesma aparência em todos os sistemas operacionais. Apesar da AWT ainda estar disponível no Java, recomenda-se que seja usado o Swing.

Ao contrário da AWT, o Swing fornece componentes de mais alto nível, possibilitando assim uma melhor compatibilidade entre os vários sistemas operacionais onde as aplicações Java são executadas. Deitel e Deitel (2003, p. 606) afirmam que os componentes Swing são inteiramente escritos em Java, não contendo uma única linha de código nativo. Além disso, é possível fornecer uma aparência e um comportamento particular para cada plataforma onde a

aplicação será executada, ou seja, a interface é configurável. No quadro 1 é apresentado o código Java correspondente a uma interface (figura 1) com componentes Swing: `JFrame`, janela principal da aplicação; `JPanel`, painel onde são colocados os demais componentes da interface; `JButton`, botão que aciona o evento associado, quando for pressionado.

```
public class Exemplo extends JFrame {
  private JPanel jContentPane = null;
  private JPanel jPanel = null;
  private JButton jButton = null;

  public Exemplo() {
     super();
     initialize();
  }

  private void initialize() {
     this.setSize(200, 65);
     this.setContentPane(getJContentPane());
     this.setTitle("JFrame");
  }

  private javax.swing.JPanel getJContentPane() {
     if (jContentPane == null) {
        jContentPane = new JPanel();
        jContentPane.setLayout(new BoxLayout(jContentPane, BoxLayout.Y_AXIS));
        jContentPane.add(getJPanel(), null);
     }
     return jContentPane;
  }

  private JPanel getJPanel() {
     if (jPanel == null) {
        jPanel = new JPanel();
        jPanel.add(getJButton(), null);
     }
     return jPanel;
  }

  private JButton getJButton() {
     if (jButton == null) {
        jButton = new JButton();
        jButton.setPreferredSize(new Dimension(66, 26));
        jButton.setText("Teste");
        jButton.addMouseListener(new MouseListener(){
           public void mouseClicked(MouseEvent e){}
           public void mouseEntered(MouseEvent e){}
           public void mouseExited(MouseEvent e){}
           public void mousePressed(MouseEvent e){
              jButton.setText("Ok");
           }
           public void mouseReleased(MouseEvent e){}
        });
     }
     return jButton;
  }

  public static void main(String[] args) {
     Exemplo e = new Exemplo();
     e.setLocationRelativeTo(null);
     e.setDefaultCloseOperation(Exemplo.EXIT_ON_CLOSE);
     e.setVisible(true);
  }
}
```

Quadro 1 – Código fonte Java

Figura 1 – Exemplo de interface

Segundo Cohen (2008, p. 2), “a interface gráfica em Java é orientada a eventos”. Cada componente de interface tem seu conjunto específico de eventos. Assim, cada vez que um usuário clica em um botão, seleciona um item em uma lista ou pressiona uma tecla, o sistema operacional gera um evento. Quando a aplicação está interessada em determinado evento, solicita ao sistema operacional que “escute” o evento. Para isso faz uso de *listeners* (tratadores de eventos), que nada mais são do que classes que tratam eventos. Assim, segundo Deitel e Deitel (2003, p. 611), o mecanismo de tratamento de eventos em Java é formado por três partes:

a) o objeto origem, isto é, o componente de interface que gerou o evento;
b) o evento propriamente dito, ou seja, um objeto que encapsula as informações necessárias para processar o evento gerado;
c) o objeto ouvinte (*listener*), ou seja, o objeto que é notificado pelo objeto origem quando ocorre um evento e que usa a informação da notificação para responder ao evento.

Para um programa estar preparado para processar eventos, precisa realizar duas tarefas: registrar o *listener* para cada componente de interface e implementar um método de tratamento de evento. Cada *listener* implementa uma ou mais interfaces *listeners* dos pacotes `java.awt.event` e `javax.swing.event` (DEITEL; DEITEL, 2003, p. 611).

Java oferece 5 tipos básicos de tratadores de eventos: `WindowListener`, `MouseListener`, `MouseMotionListener`, `ActionListener` e `KeyListener`. Para cada um, Java oferece uma classe ou interface que pode ser utilizada na implementação das aplicações para tratar eventos. Cada um deles possui uma série de diferentes métodos para tratar eventos específicos, sendo que o programador deve implementar apenas os métodos correspondentes ao evento que deseja tratar.

2.3 HTML E JAVASCRIPT

HTML é uma linguagem de marcação utilizada para compor documentos, formatando

texto e criando ligações entre páginas web. As páginas HTML podem ser interpretadas por um navegador ou um *browser* (HTML, 2008), sendo que a estrutura de um documento HTML está baseada no uso de *tags*, normalmente definidas em pares, que delimitam o texto que sofrerá algum tipo de formatação. A estrutura básica de um documento HTML deve conter as seguintes *tags* (HTML, 2008): `<html>` e `</html>`, que define, respectivamente, o início e fim do documento; `<head>` e `</head>`, `<body>` e `</body>`, que delimitam, respectivamente, o cabeçalho e o conteúdo principal do documento. No quadro 2 é apresentado o código HTML de uma página (figura 2) com componentes equivalentes àqueles da figura 1.

```
<HTML>
   <HEAD>
   <TITLE> JFrame </TITLE>
   </HEAD>
<BODY>
   <script>
          function mousePressedjButton(){
                 document.Exemplo.jButton.value='Ok'
          }
   </script>

   <fieldset
          name= jContentPane
          style="position: absolute;
          font-family: dialog;
          font-size: 12;
          color: #000000;
          background-color: #CCCCCC;
          left: 0; top: 0;
          width: 200; height: 50;">

   <form name= Exemplo>
          <fieldset
                 name = jPanel
                 style="position: absolute;
                 font-family: dialog;
                 font-size: 12;
                 width: 200; height: 20;
                 border: 0;
                 left: 0; top: 0; text-align=">

          <button
                 name = jButton id = idjButton
                 style="position: relative;
                 font-fmily: dialog;
                 font-size: 12;
                 width: 66; height: 26;
                 left: 64; top: 0;"
                 onClick=mousePressedjButton()>
          Teste
          </button>
          </fieldset>
   </form>
   </fieldset>
</BODY>
</HTML>
```

Quadro 2 – Código HTML

Figura 2 – Exemplo de interface

Páginas HTML são estáticas e, por este motivo, são limitadas para o desenvolvimento de aplicações web. Para criação de páginas dinâmicas surgiram várias linguagens, como por exemplo Java, através da tecnologia dos Applets, que segundo Deitel e Deitel (2003, p.136) são programas Java que podem ser embutidos dentro de documentos HTML(Deitel e Deitel, 2003). Também foi desenvolvida, pela Netscape, uma linguagem de programação chamada LiveScript, que permitia criar pequenos programas nas páginas HTML. Antes de lançar a primeira versão da LiveScript, a Netscape fez uma aliança com a Sun Microsystems para desenvolver em conjunto essa nova linguagem. Dessa aliança originou-se o JavaScript (CRIARWEB, 2004).

JavaScript é uma linguagem que permite a programação de *scripts* orientados a objetos, com estruturas de dados complexas, permitindo controlar cada evento que ocorre nas páginas HTML. De acordo com Alvarez (2008), entre as ações típicas que podem ser realizadas em JavaScript, existem dois tipos: os efeitos especiais sobre páginas web para criar conteúdos dinâmicos e elementos da página que tenham movimento; e as instruções executadas como resposta às ações do usuário, com as quais podem-se criar páginas interativas com programas como calculadoras, agendas ou tabelas de cálculo. Para programar em JavaScript não é necessário um *kit* de desenvolvimento, nem tão pouco a compilação dos *scripts* ou a implementação dos mesmos em arquivos externos ao código HTML, como ocorreria com os Applets.

2.4 MOTOR DE *TEMPLATES* VELOCITY

Conforme sugerido por Herrington (2003, p. 77) e já citado neste trabalho, uma das etapas para geração de código envolve a especificação de *templates*. O uso de *templates* na

camada de apresentação de aplicações web simplifica a manutenção (SILVEIRA, 2008).

Segundo Rocha (2005), *templates* são mecanismos de software que visam possibilitar a separação entre código dinâmico e código de interface de forma que a construção desses possa ocorrer de forma independente no desenvolvimento de aplicações. *Templates* são arquivos que contêm variáveis e blocos especiais que são dinamicamente transformados a partir de dados enviados pelo motor de *templates*. Para tanto, deve existir um programa responsável por instanciar o motor, associando valores às variáveis e aos blocos especiais do *template*.

Várias aplicações podem utilizar motores de *templates*, desde a geração de um simples arquivo texto à criação de arquivos binários, tendo como restrição seguir um determinado padrão de formatação. Usar *templates*, segundo Silveira (2008), melhora a divisão de tarefas, pois os programadores ficam concentrados na lógica do sistema e os *designers* se preocupam apenas com o *layout,* o que reduz o risco de alterarem indevidamente o código da aplicação. Além disso, os *templates* ficam em arquivos separados, possibilitando mudar o visual da aplicação sem a necessidade de re-compilar ou reiniciar o sistema. Existem ferramentas disponíveis no mercado que fazem uso de motores de *templates*: uma delas é a ferramenta Poseidon for UML (SILVEIRA, 2008), desenvolvida em Java, utiliza *templates* para a geração de código a partir de diagramas UML; a outra é o Velocity, também desenvolvido em Java, é um subprojeto do projeto Jakarta, da Apache Foundation.

Para desenvolvimento da ferramenta proposta optou-se pela utilização do motor de *templates* Velocity, o qual contém algumas diretivas bastante simples de serem usadas para escrever documentos. Projetado para ser uma ferramenta de fácil utilização e que servisse como ferramenta genérica de *templates,* o Velocity é ideal em qualquer tipo de programa Java que necessite de formatação de dados e apresentação dos mesmos. Segundo Steil (2006b), algumas das razões para usá-lo são: tem sintaxe clara e fácil de entender; integra-se fácil a qualquer tipo de aplicação baseada em Java; não é restrito à internet; podendo ser usado em qualquer área.

Quando se utiliza *templates* para a geração de código, em algum momento deve-se processar o *template*, ou seja, deve-se desenvolver código para que algumas partes do *template* sejam substituídas por dados extraídos pela aplicação. Para escrever um *template*, além de variáveis, utiliza-se algumas diretivas ou comandos, usando uma pequena linguagem que o *template* deve respeitar. Em Velocity, existe a VTL, que é a linguagem utilizada para incorporar conteúdo dinâmico no *template*. Toda diretiva é precedida pelo caracter `#`. No quadro 3, são apresentadas as diretivas mais utilizadas e nos quadros 4 e 5 tem-se exemplos

de *templates* Velocity que podem ser utilizados para gerar o código apresentado no quadro 2.

DIRETIVA	DESCRIÇÃO
`#set`	atribuição de valor à variável
`#foreach`	comando de repetição
`#if`	comando condicional
`#else`	comando condicional
`#elseif`	comando condicional
`#include`	inclusão de texto estático no *template*
`#parse`	comando usado para interpretar o código (estático e dinâmico) contido em outro arquivo, permitindo construir *templates* em partes

Quadro 3 – Elementos da linguagem de *templates* Velocity

```
<HTML>
   <HEAD>
   <TITLE> $!{JFrameGetTitle} </TITLE>
   </HEAD>
<BODY>
$!{funcJavaScript}
   <fieldset
          name= $!{jcontentPanelGetName}
          style="position: absolute;
          font-family: $!{jcontentPanelGetFont};
          font-size: $!{jcontentPanelGetFontSize};
          color: $!{jcontentPanelGetFontColor};
          background-color: $!{jcontentPanelGetBackgroudColor};
          left: $!{jcontentPanelGetPositionLeft};
          top: $!{jcontentPanelGetPositionTop};
          width: $!{jcontentPanelGetWidth};
          height: $!{jcontentPanelGetHeight};">

   <form name= $!{JFrameGetName}>
   #foreach ($ComponentJPanel in $arrayJPanel)
          <fieldset
                 name= $!{ComponentJPanel.getName()}
                 style="position: absolute;
                 font-family: $!{ComponentJPanel.getFontFamily()};
                 font-size: $!{ComponentJPanel.getFontSize()};
                 width: $!{ComponentJPanel.getSizeWidth()};
                 height: $!{ComponentJPanel.getSizeHeight()};
                 border: 0;
                 left: $!{ComponentJPanel.getPositionLeft()};
                 top: $!{ComponentJPanel.getPositionTop()};
                 text-align= $!{style}">
          #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
          #foreach ($Component in $ComponentJPanel.getChildrenComponents())
          #if ($!{Component.getType()} == "JButton")
                 #parse("templateJButton.vm")
          #end
          #end
          #end
          </fieldset>
   #end
   </form>
   </fieldset>
</BODY>
</HTML>
```

Quadro 4 – Exemplo de *template* Velocity com diretiva (`#parse`)

```
<button
   name= $!{Component.getName()} id= id$!{Component.getName()}
   style="position: relative;
   font-family: $!{Component.getFontFamily()};
   font-size: $!{Component.getFontSize()};
   width: $!{Component.getSizeWidth()}; height: $!{Component.getSizeHeight()};
   left: $!{Component.getPositionLeft()}; top: $!{Component.getPositionTop()};"
   $!{Component.getFunctionJavaScript().getMethod()}>
$!{Component.getText()}
</button>
```

Quadro 5 – Exemplo de *template* Velocity (`templateJButton.vm`)

Depois de escrito o *template*, deve-se escrever o código que irá ler e processar o *template*. Para escrever este código, segundo Steil (2006b), alguns passos devem ser seguidos:

a) cria-se e inicializa-se o Velocity (quadro 6);

b) cria-se o contexto que liga o código Java ao *template* (quadro 7);

c) define-se o *template* a ser usado (quadro 8);

d) informa-se os valores a serem passados às variáveis do *template* (quadro 9);

e) usa-se o método `merge` para substituir o código dinâmico do *template* (quadro 10).

```
VelocityEngine ve = new VelocityEngine();
ve.init();
```

Quadro 6 – Exemplo de criação e inicialização do Velocity

```
VelocityContext context = new VelocityContext();
```

Quadro 7 – Exemplo de criação de contexto

```
Template templateVE = ve.getTemplate("Exemplo.vm");
```

Quadro 8 – Exemplo de definição do *template* a ser utilizado

```
context.put("JFrameGetTitle", recClass.getTitle());
```

Quadro 9 – Exemplo de passagem de valores às variáveis

```
StringWriter writer = new StringWriter();
templateVE.merge(context, writer);
```

Quadro 10 – Exemplo de uso do método `merge`

2.5 TRABALHOS CORRELATOS

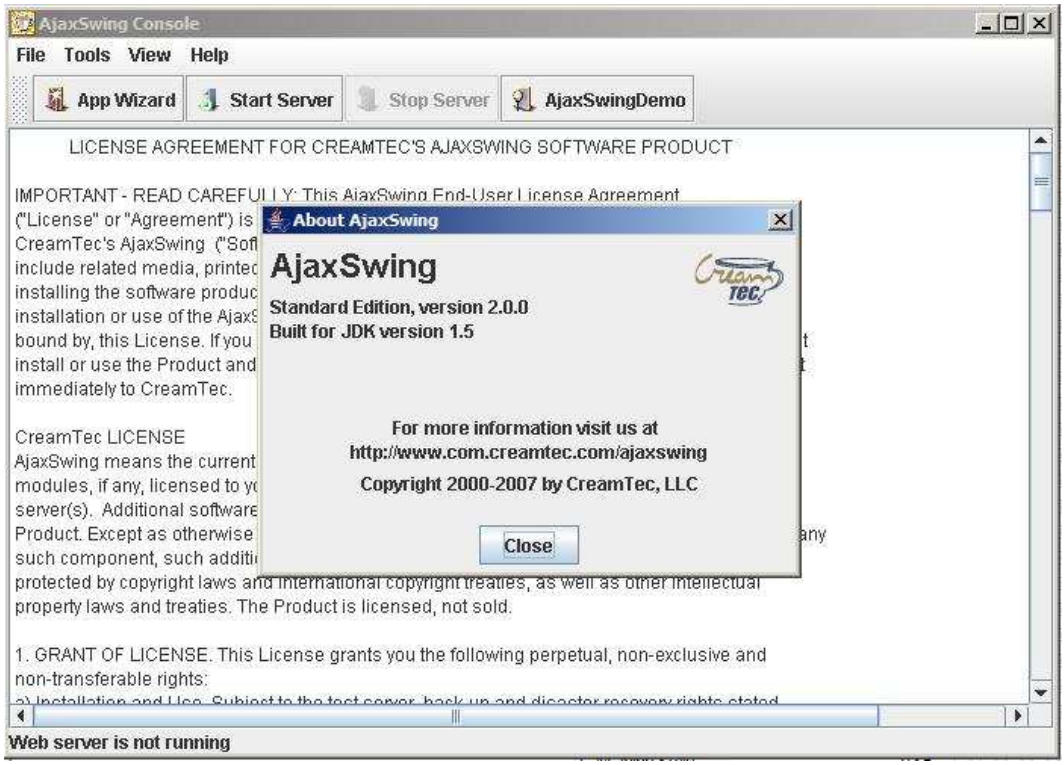
Existem aplicações com funcionalidades semelhantes às da ferramenta proposta neste trabalho: AjaxSwing, DelphiToWeb e Converte Forms. Mesmo desempenhando um papel semelhante, cada uma delas possui particularidades, descritas a seguir.

2.5.1 AjaxSwing

AjaxSwing, segundo CreamTec (2008), é uma ferramenta que converte aplicações Java (Swing ou AWT) em HTML e AJAX. Vem sendo utilizada por várias empresas desde 2000. A conversão envolve a instalação da ferramenta e criação de um arquivo de configuração para a aplicação a ser convertida. Algumas das principais características da ferramenta são (CREAMTEC, 2008):

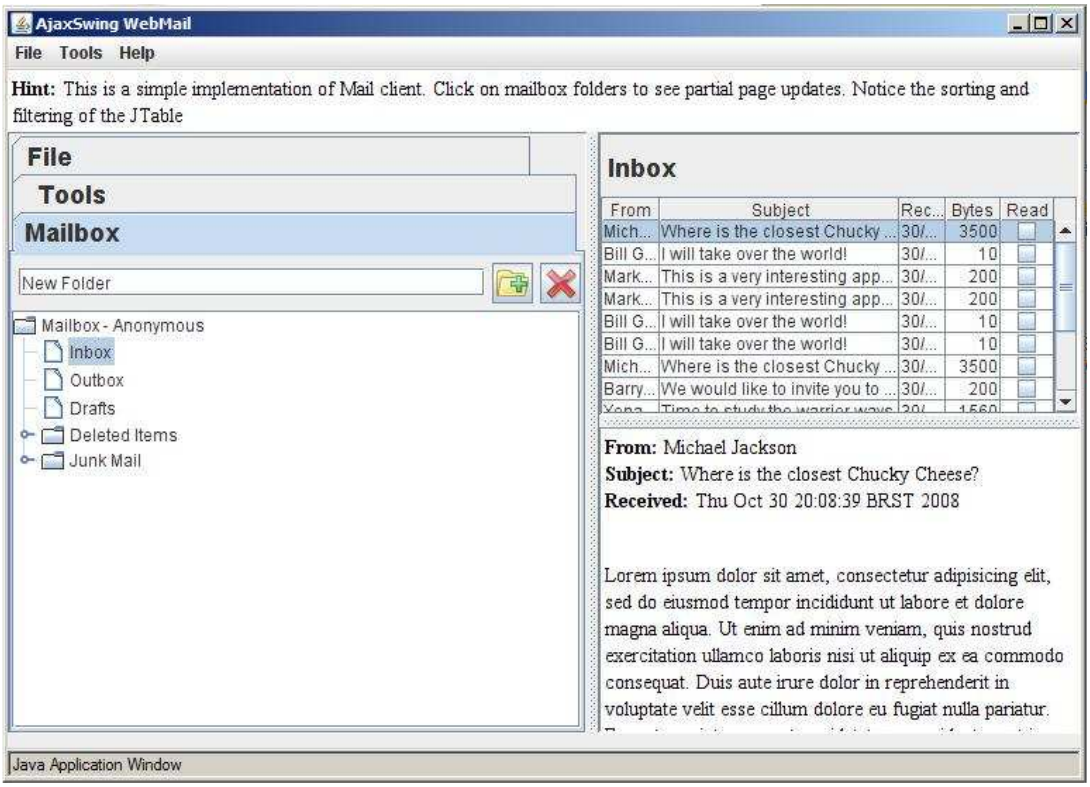
a) conversão de aplicações Java em HTML com o mínimo de modificações;
b) não é necessário conhecimento de HTML, CSS ou JavaScript para usar a ferramenta;
c) suporte a HTML 4.0, CSS2 e JavaScript 1.2;
d) conversão de aplicações para Internet Explorer 4 ou superior, FireFox 1.2 ou superior;
e) customização das páginas HTML geradas (configuração de cores, fontes, estilos);
f) suporte à internacionalização.

Na figura 3 é apresentada a tela principal da ferramenta, a figura 4 apresenta uma interface Java e a figura 5 apresenta a mesma interface Java convertida em HTML pela ferramenta AjaxSwing.



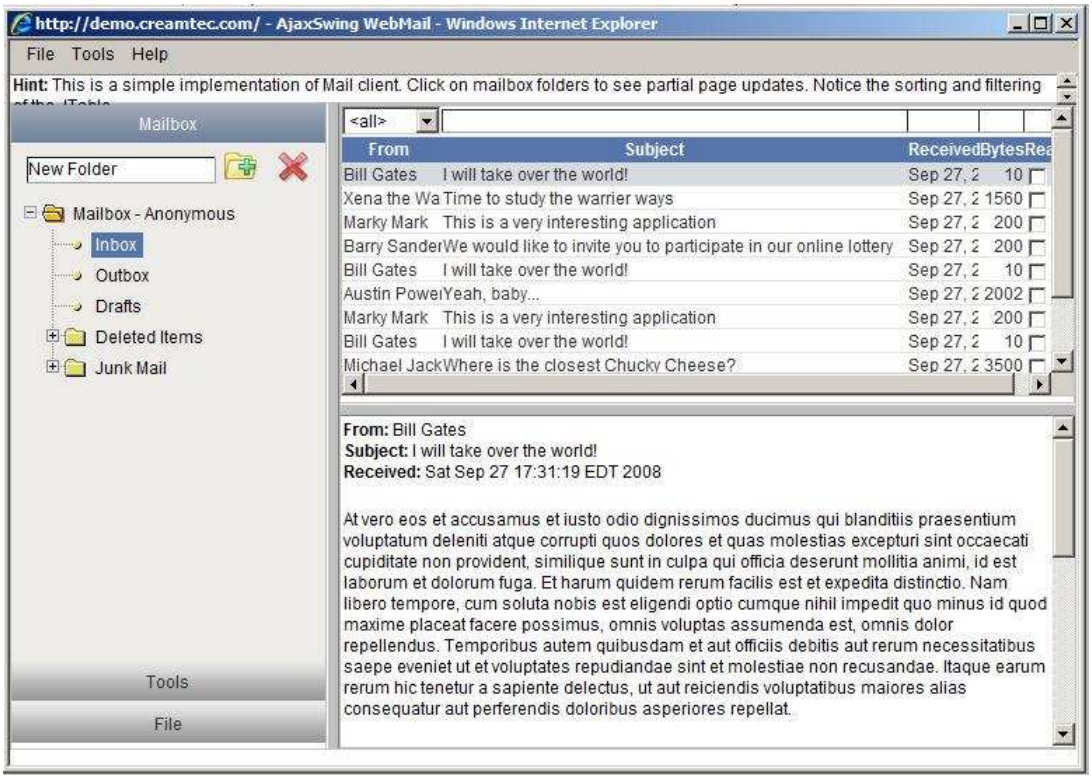
Fonte: CreamTec (2008).

Figura 3 – Ferramenta AjaxSwing



Fonte: CreamTec (2008).

Figura 4 – Interface Java



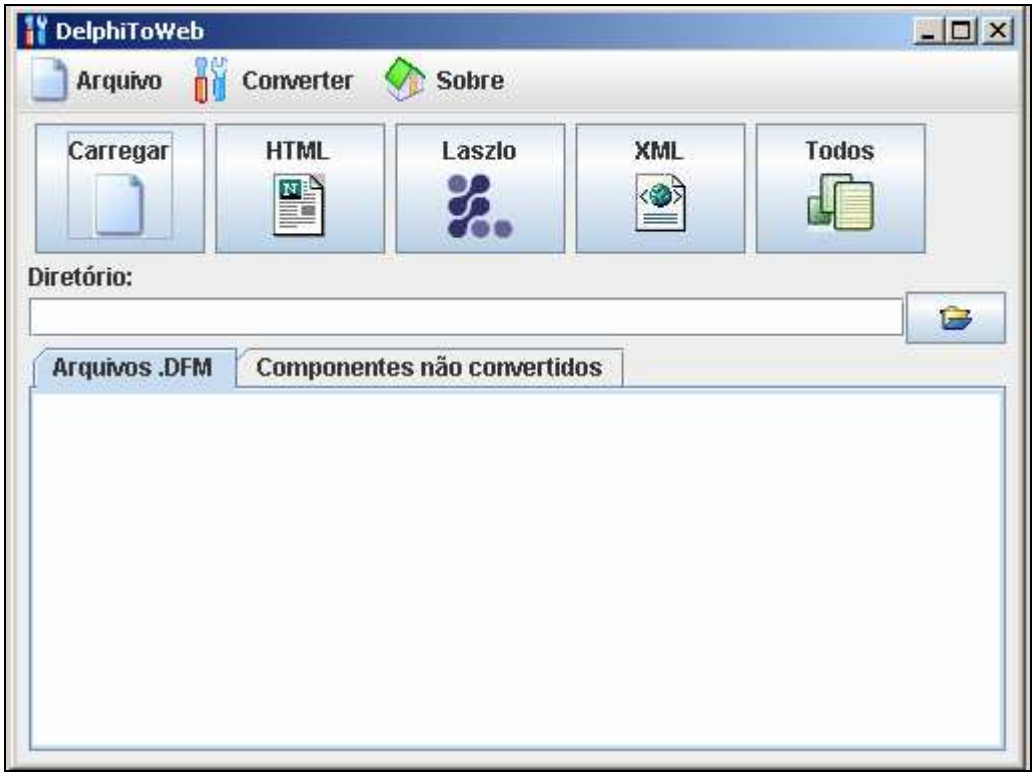
Fonte: CreamTec (2008).

Figura 5 – Interface convertida pela ferramenta AjaxSwing

2.5.2 DelphiToWeb

DelphiToWeb é uma ferramenta para converter formulários Delphi em páginas HTML (SOUZA, 2005). A ferramenta pode ser usada para gerar código da camada de interface de uma aplicação web a partir de uma aplicação desenvolvida em Delphi, sendo o resultado da conversão páginas HTML ou arquivos .LZX. Outra opção da ferramenta é a geração de uma representação intermediária em arquivos XML, contendo os dados dos formulários Delphi (SOUZA, 2005, p. 57).

A entrada do gerador são arquivos com extensão .DFM, contendo as propriedades dos componentes de interface para construir código com as funcionalidades para a camada de interface de uma aplicação web. A análise dos arquivos de entrada foi feita utilizando analisadores de linguagens (léxico, sintático e semântico) e a formatação do código de saída a ser gerado está "embutida" no código da ferramenta, já que não foram utilizados *templates*. Na figura 6 é apresentada a tela da ferramenta DelphiToWeb e, em seguida, na figura 7 a interface gerada em HTML, a partir do arquivo .DFM apresentado na figura 8.



Fonte: Souza (2005, p. 54).

Figura 6 – Ferramenta DelphiToWeb

Fonte: Souza (2005, p. 59).

Figura 7 – Interface gerada pela ferramenta DelphiToWeb

Fonte: Souza (2005, p. 58).

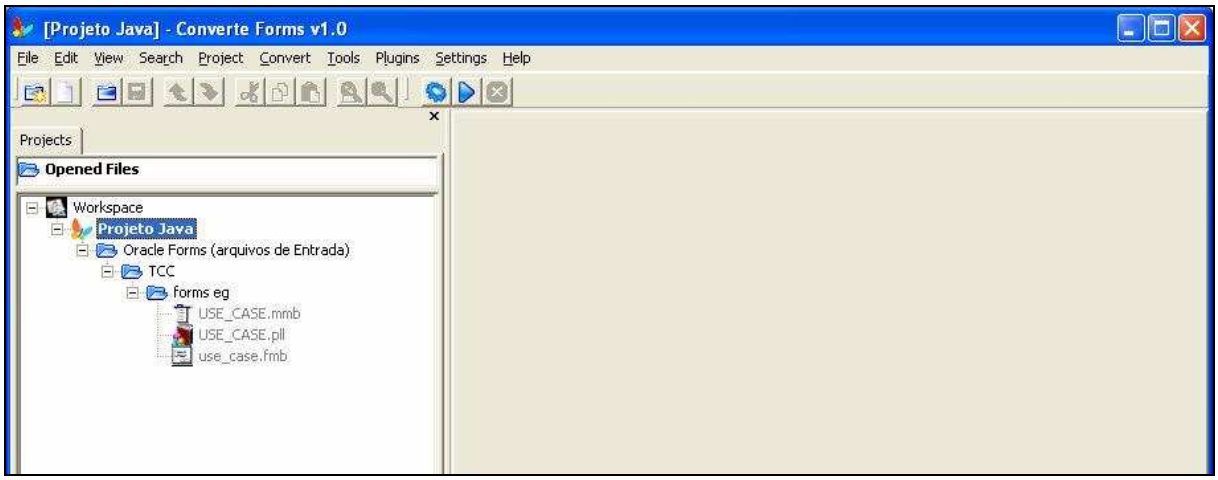
Figura 8 – Interface .DFM

2.5.3 Converte Forms

Converte Forms é, segundo Schvepe (2006, p. 13), uma ferramenta para geração de código que permite automatizar parte do processo de migração de aplicações desenvolvidas em Oracle Forms 6i para aplicações *desktop* em Java. A entrada do gerador são arquivos do Oracle Forms 6i, enquanto a saída são classes Java. Observa-se que a ferramenta traduz apenas as construções procedurais do código PL/SQL, converte 18 componentes de interface e faz o mapeamento de 28 eventos de controle dos componentes de interface para eventos

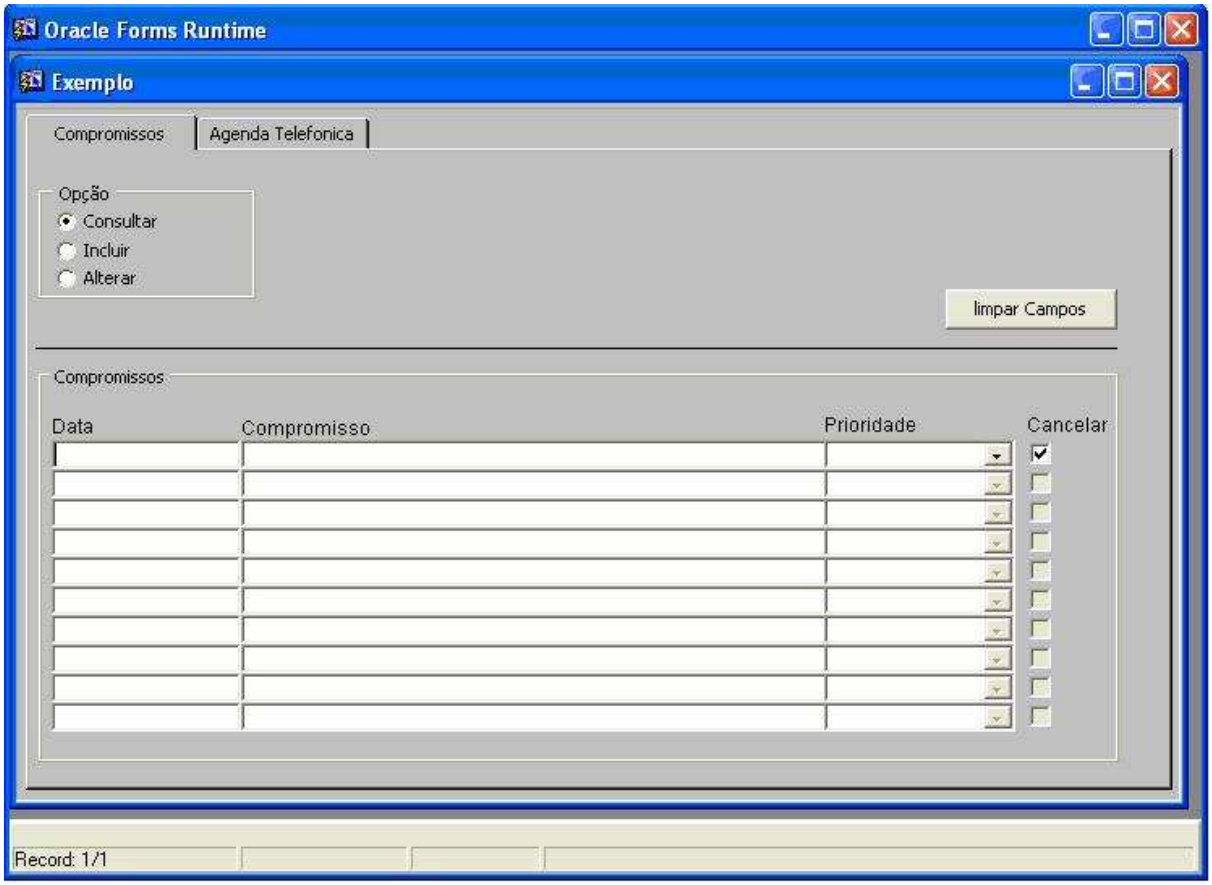
Java.

Desenvolvida em C++, a ferramenta utiliza analisadores de linguagens na análise dos arquivos de entrada e o motor de *templates* eNITL para geração do código de saída. Na figura 9 é apresentada a tela principal da ferramenta, a figura 10 apresenta uma interface em Oracle Forms a ser convertida para Java (figura 11).



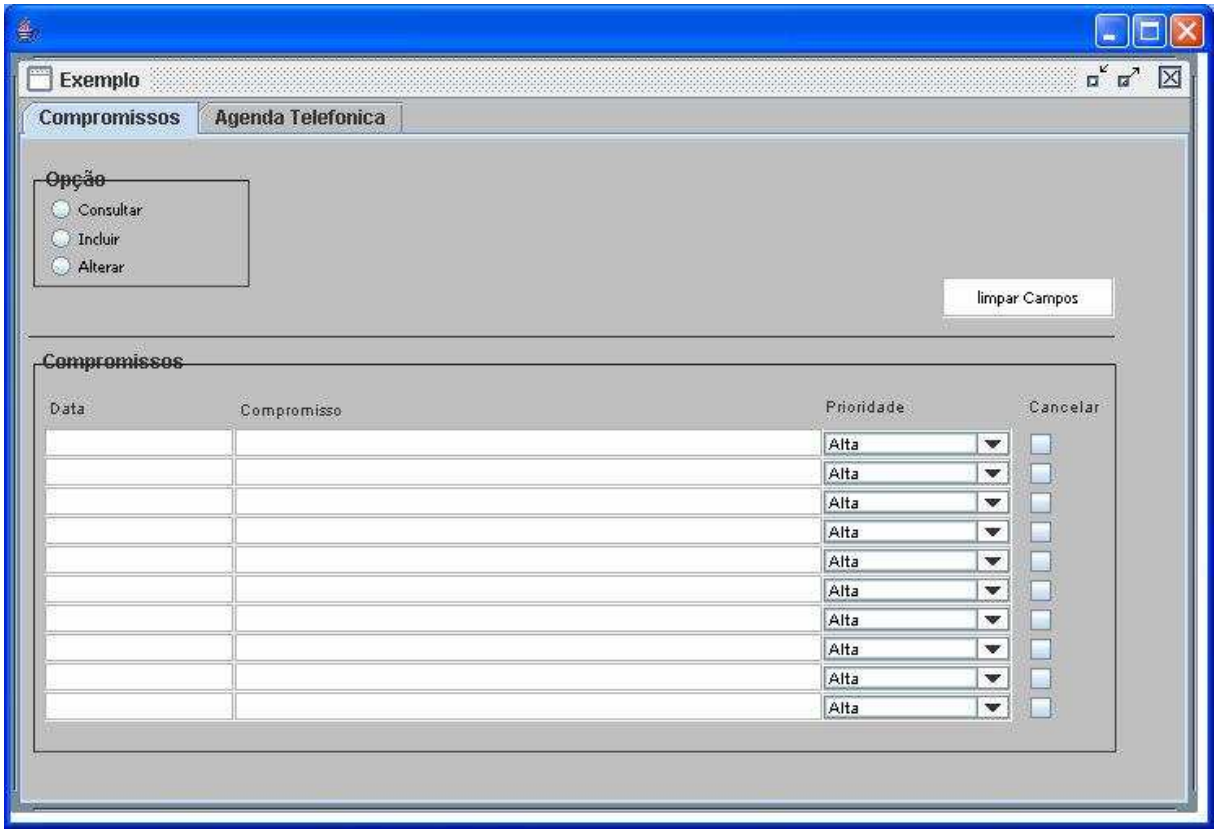
Fonte: Schvepe (2006, p. 61).

Figura 9 – Ferramenta Converte Forms



Fonte: Schvepe (2006, p. 63).

Figura 10 – Interface Oracle Forms



Fonte: Schvepe (2006, p. 64).

Figura 11 – Interface convertida pela ferramenta Converte Forms

3 DESENVOLVIMENTO

Este capítulo contextualiza e descreve o desenvolvimento da ferramenta proposta, denominada SwingToHtml. Foram realizadas as seguintes etapas:

a) elicitação dos requisitos: foram reavaliados e detalhados os requisitos funcionais e não funcionais da ferramenta inicialmente identificados;
b) identificação do código a ser convertido: fez-se necessário analisar componentes de interface Java Swing, identificando as propriedades daqueles que serão traduzidos. Também foi necessário analisar o tratamento de eventos de interface para determinar quais eventos serão convertidos;
c) definição do código de saída da ferramenta: foram desenvolvidos manualmente páginas HTML e *scripts* JavaScript com componentes de interface e tratadores de eventos equivalentes aos identificados na etapa anterior, para definir o mapeamento entre o código de saída e o código de entrada da ferramenta;
d) definição da análise do código de entrada: estudou-se a estrutura da BNF do Java a fim de definir como serão extraídas, do código de entrada, as informações necessárias para gerar a saída;
e) especificação dos *templates*: foram elaborados os *templates* Velocity utilizados para gerar o código de saída;
f) especificação da ferramenta: foi especificada com análise orientada a objeto utilizando a UML. Com a ferramenta EA foram desenvolvidos dos diagramas de casos de uso, de classes e de seqüência;
g) implementação: para ler e recuperar as informações dos arquivos de entrada (`.java`), foram utilizados os analisadores léxico e sintático gerados pela ferramenta JavaCC a partir da BNF adaptada de Gesser (2007). Para gerar a saída, utilizou-se o motor de *templates* Velocity 1.5. Na implementação da ferramenta foi usado o ambiente de desenvolvimento Eclipse 3.2.2;
h) testes: os testes foram realizados durante todo o processo de desenvolvimento da ferramenta.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A seguir são apresentados requisitos funcionais e requisitos não funcionais. O quadro 11 apresenta os requisitos funcionais e sua rastreabilidade, ou seja, vinculação com o caso de uso associado. No quadro 12 são relacionados os requisitos não funcionais.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: Converter componentes de interface Java Swing para HTML.	UC03
RF02: Converter tratadores de eventos da linguagem Java para JavaScript.	UC03

Quadro 11- Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Manter o *layout* original dos componentes de interface e a funcionalidade dos tratadores de evento.
RNF02: Utilizar *templates* para a geração do código de saída.
RNF03: Utilizar linguagem Java 5.0 para implementação da ferramenta.
RNF04: Executar em ambiente Windows XP e Vista.
RNF05: Executar no navegador Internet Explorer 7.

Quadro 12 – Requisitos não funcionais

3.2 COMPONENTES DE INTERFACE E EVENTOS CONVERTIDOS

Nesta seção são apresentados os componentes e os eventos convertidos pela ferramenta SwingToHtml. Para determinar os componentes de interface (quadro 13) e tratadores de eventos (quadro 14) que serão convertidos, identificou-se os que são mais usados e os que tinham componentes equivalentes em HTML e JavaScript.

COMPONENTE	DESCRIÇÃO	*Tag* HTML
`JFrame`	formulário ou janela da aplicação	`<fieldset>`
`JPanel`	painel para melhor organizar os componentes	`<fieldset>`
`JMenuBar`	área destinada a armazenar menus	`<div>`
`JMenu`	menu principal	`<div>, <button>`
`JMenuItem`	opções do menu principal	`<div>, <button>`
`JTabbedPane`	página que armazena painéis	`<table>, <div>`
`JLabel`	texto descritivo.	`<font>`
`JTextField`	caixa de digitação de texto de linha única	`<input>`
`JTextArea`	caixa de texto com barra de rolagem	`<textarea>`
`JButton`	botão, não há tratamento de imagens na ferramenta	`<button>`
`JRadioButton`	botão em forma de círculo seguido de uma descrição	`<input>`
`JCheckBox`	caixa de seleção	`<input>`
`JComboBox`	caixa de seleção de um único item, com uma lista de opções	`<input>`
`JList`	caixa com uma lista de opções com barra de rolagem	`<select>`

Quadro 13 – Componentes de interface

EVENTOS	DESCRIÇÃO	JavaScript
`mouseClicked`	disparado quando o botão do *mouse* é clicado (pressionado e liberado) em um componente	`onClick`
`mouseEntered`	disparado quando o *mouse* entra na área de um componente	`onMouseOver`
`mouseExited`	disparado quando o *mouse* sai da área de um componente	`MouseOut`
`mouseReleased`	disparado quando o botão do *mouse* é liberado sobre um componente	`onClick`
`focusGained`	invocado quando o componente recebe o foco	`onFocus`
`actionPerformed`	invocado quando uma ação ocorre	`onClick`
`keyTyped`	invocado quando uma tecla é pressionada: a ferramenta converte apenas a chamada ao método, sendo que o evento de capturar a tecla deve ser implementado pelo programador	`onKeyPress`
`keyPressed`	invocado quando uma tecla é pressionada	`onKeyPress`
`keyReleased`	invocado quando uma tecla é liberada (após ser pressionada)	`onKeyUp`

Quadro 14 – Tratadores de eventos

3.3 CÓDIGO DE SAÍDA

Para definir o código de saída da ferramenta foram desenvolvidos manualmente páginas HTML e *scripts* JavaScript com componentes de interface e tratadores de eventos equivalentes aos identificados na seção 3.2. Como resultado dessa etapa, tem-se o mapeamento entre o código de saída e o código de entrada da ferramenta. No quadro 15 é apresentado o código do evento `mouseClicked` em Java e o correspondente em JavaScript. No quadro 16 são apresentados os códigos de algums componentes mais utilizados.

EVENTO EM JAVA

```
jbutton.addMouseListener
                  (new MouseListener() {
  public void mouseClicked(MouseEvent e)
  {
    textfield2.setText("*");

    if (op != ""){
      val1 = textfield4.getText();

      textfield1.setText(val1);

      textfield3.setText("");

    }
    op = "*";
  }
  ...
}
```

EVENTO EM JAVASCRIPT

```
function mouseClickedjbutton(){



    document.FrCalculator2.
                    textfield2.value= '*'
    if ( op != '' ) {
      val1 = document.FrCalculator2.
                        textfield4.value
      document.FrCalculator2.
                   textfield1.value= val1
      document.FrCalculator2.
                     textfield3.value= ''
    }
    op =  '*'
}
```

Quadro 15 – Mapeamento eventos Java X eventos JavaScript

COMPONENTE	CÓDIGO JAVA SWING	CÓDIGO HTML EQUIVALENTE
JButton	jbutton = new JButton(); jbutton.setText("Limpar"); jbutton.setPreferredSize (new Dimension(250, 30));	<button name=jbutton id=idjbutton style="position: absolute; font-family:dialog; font-size:12; width:250; height:30; left:2; top:0;" > Limpar </button>
JLabel	jlabel = new JLabel(); jlabel.setText("Nome:"); jlabel.setPreferredSize(new Dimension(64,10));	 Nome:
JTextField	jtextField = new JTextField(); jtextField.setPreferredSize(new Dimension(200,20));	<input type="edit" name= jtextField style= "position: absolute; font-family: dialog; font-size: 12; width: 200; height: 20; left: 69; top: 0;"></input>

Quadro 16 – Mapeamento componentes de interface Java Swing X componentes de interface HTML

Observa-se que para criação dos componentes em HTML utiliza-se CSS para definir o estilo dos componentes, tais como, cor, tamanho, tipo de fonte e posições (`top`, `left`).

3.4 ANÁLISE DO CÓDIGO DE ENTRADA

Para analisar o código de entrada, estudou-se a estrutura da BNF do Java a fim de definir como serão extraídas, do código de entrada, as informações necessárias para gerar a saída. Para tanto, utilizou-se a gramática apresentada no Anexo A, extraída do trabalho de Gesser (2007). Nela foram acrescentados métodos correspondentes a ações semânticas utilizadas para reconhecer componentes, propriedades e eventos necessários para a geração do código de código, conforme mapeamento identificado.

3.5 ESPECIFICAÇÃO

Esta seção apresenta a especificação da ferramenta, contendo os casos de uso, os diagramas de classes e de seqüência que foram especificados utilizando a ferramenta EA.

3.5.1 Casos de uso

No diagrama de casos de uso foram especificados os casos de uso que ilustram todas as etapas necessárias para a conversão de interfaces de aplicações Java Swing em páginas HTML e eventos para JavaScript. Na figura 12 são apresentados os casos de usos relevantes ao processo de conversão, que vão desde a seleção dos arquivos necessários até a conversão propriamente dita. Nos quadros 17 a 19 são apresentados os detalhamentos de cada caso de uso.

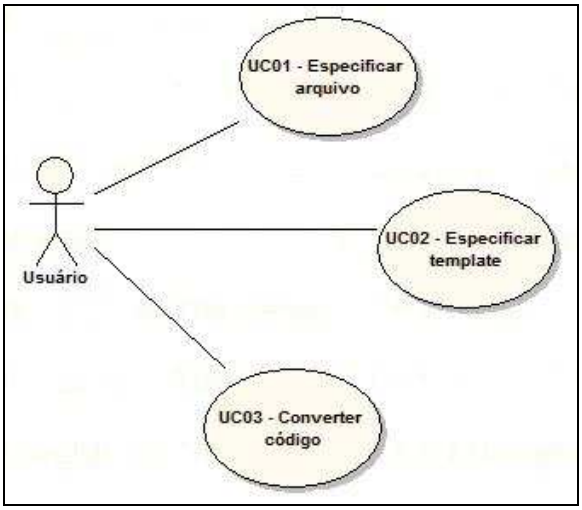


Figura 12 – Diagrama de casos de uso

UC01 – Especificar arquivo
Pré-condições: deve existir ao menos um arquivo `.java`. **Cenário principal:** 1 - Usuário clica no botão **Carregar arquivo**. 2 - A ferramenta apresenta a relação de arquivos `.java` existentes. 3 - O usuário escolhe um arquivo. 4 - A ferramenta armazena o nome do arquivo selecionado. **Pós- condições:** um arquivo foi incluído na lista de arquivos a serem convertidos.

Quadro 17 – Detalhamento do caso de uso `UC01 - Especificar arquivo`

UC02 – Especificar *template*
Pré-condições: deve existir ao menos um *template*. **Cenário principal:** 1 - Usuário clica no botão **Carregar template**. 2 - A ferramenta apresenta a relação dos *templates* que o usuário pode utilizar. 3 - O usuário escolhe um *template*. 4 - A ferramenta armazena o nome do *template* selecionado. **Pós- condições:** um *template* foi selecionado para gerar código de saída.

Quadro 18 – Detalhamento do caso de uso `UC02 - Especificar template`

UC03 – Converter código
Pré-condições: deve existir ao menos um arquivo `.java` na relação de arquivos a serem convertidos, bem como um *template* selecionado para gerar código de saída. **Cenário principal:** 1 - Usuário clica no botão **Converter**. 2 - A ferramenta analisa o *template* selecionado. 3 - A ferramenta analisa o código dos arquivos `.java` relacionados na lista de arquivos a serem convertidos. 4 - A ferramenta gera os arquivos de saída conforme dados extraídos do(s) arquivo(s) `.java` e do código especificado no *template*. **Pós- condições:** um ou mais arquivos `.html` foram gerados no diretório base da ferramenta.

Quadro 19 – Detalhamento do caso de uso `UC03 - Converter código`

3.5.2 Diagrama de classes

No diagrama de classes da figura 13 encontram-se modeladas as classes que armazenam as informações que serão utilizadas para conversão, ou seja, classes que armazenam dados dos componentes de interface Java Swing.

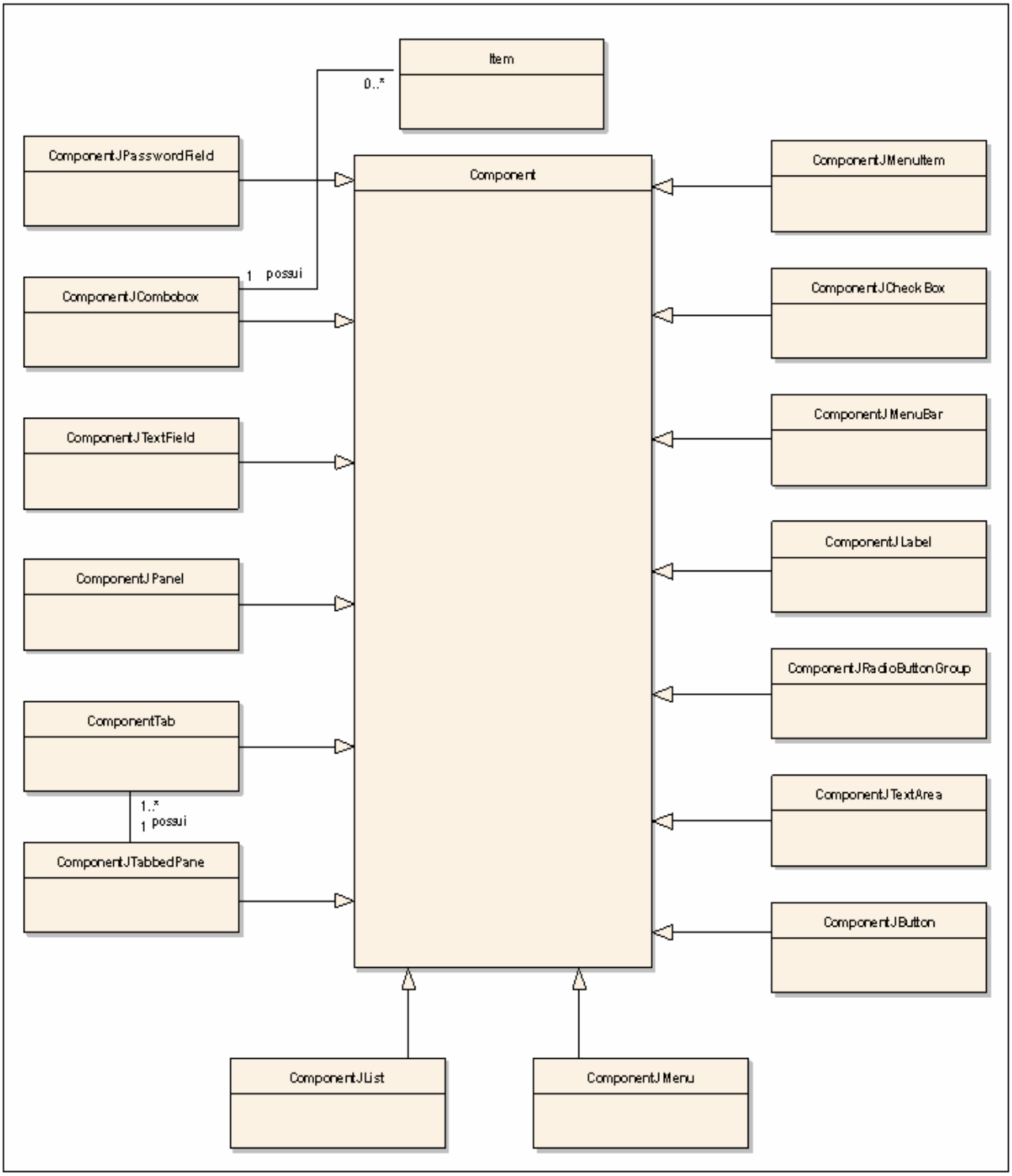


Figura 13 – Diagrama de classes para componentes de interface Java Swing

Já no diagrama de classes da figura 14 foram modeladas as classes responsáveis pela conversão de código. A classe principal, `FrMain`, é responsável pela entrada dos dados (arquivos `.java` a serem convertidos e *template* a ser utilizado no processo de conversão) e pela chamada dos métodos para analisar a entrada e acionar o motor de *templates* Velocity para efetuar a conversão dos arquivos selecionados. Tem-se as classes `ASTParser`,

`ASTParserConstants`, `ASTParserTokenManager`, `ParseException`, `JavaCharStream`, `Token` e `TokenMgrError`, que efetuam a análise da entrada extraindo as informações necessárias para gerar a saída, através de analisadores léxico e sintático para a BNF do Java (Anexo A). No diagrama são apresentadas ainda as classes `Function` e `Identifier`, que armazenam, respectivamente, o corpo das funções JavaScript e os identificadores reconhecidos nos métodos Java que serão convertidos para JavaScript. Também é apresentada a classe `RecClass`, que guarda informações como o nome da classe (arquivo `.java` que será convertido) e atributos como `width` e `heigth`.

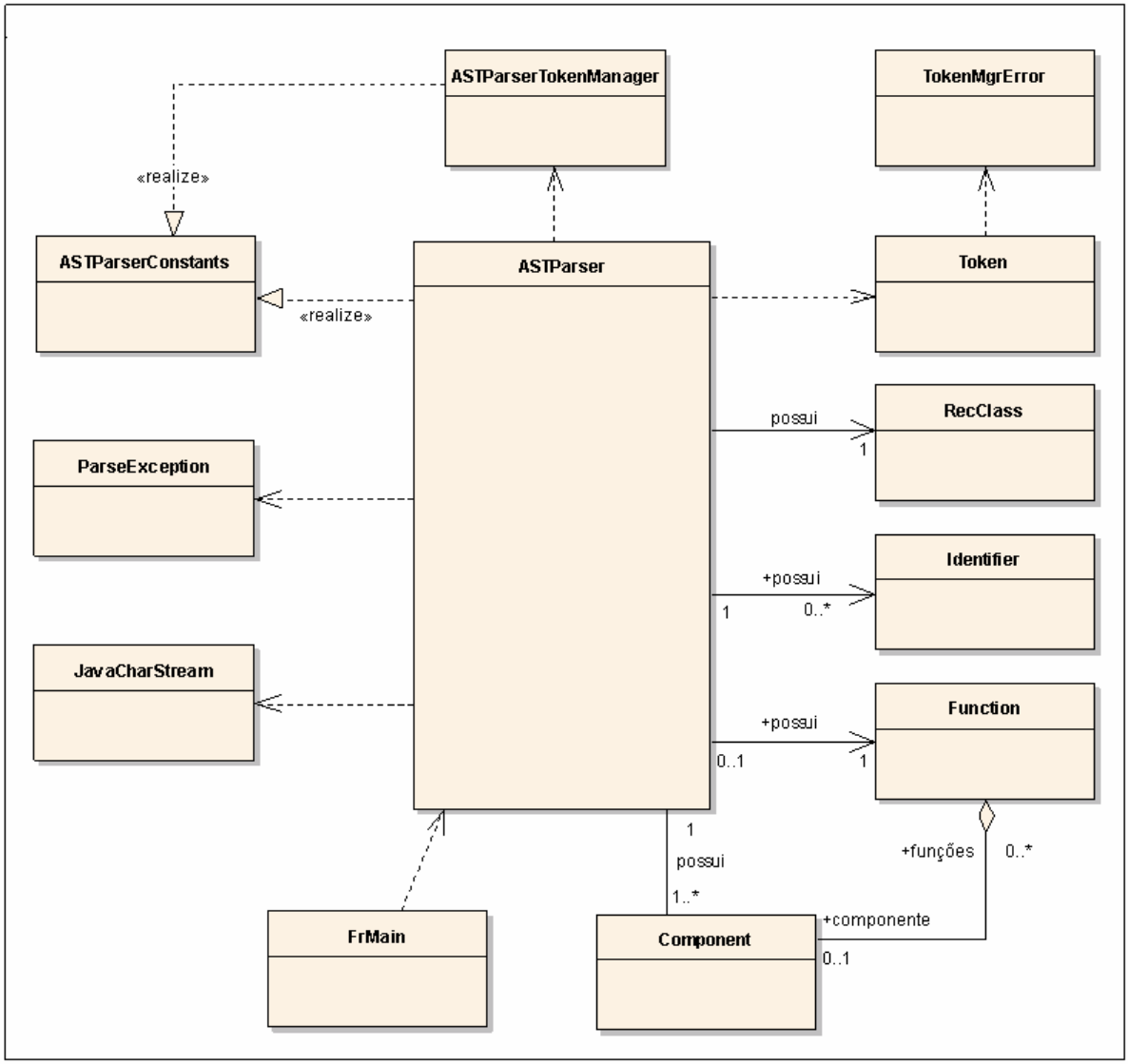


Figura 14 – Diagrama de classes para geração de código

No Apêndice B, são relacionados os atributos e métodos das classes apresentadas nas figuras 13 e 14.

3.5.3 Diagrama de seqüência

O diagrama de seqüência da figura 15 detalha o evento que ocorre ao pressionar o botão **Converter**. Depois de selecionado um ou mais arquivos para serem convertidos e um *template* para formatar o código de saída, pode-se iniciar o processo de conversão. Nesse caso, a classe `FrMain` invoca a classe `ASTParser`, que por sua vez analisa o arquivo `.java` a ser convertido, armazena os dados a serem utilizados na conversão em classes criadas para este fim (figura 13), invoca o motor de *templates* Velocity, passando os dados extraídos do arquivo `.java` e o *template* modelo.

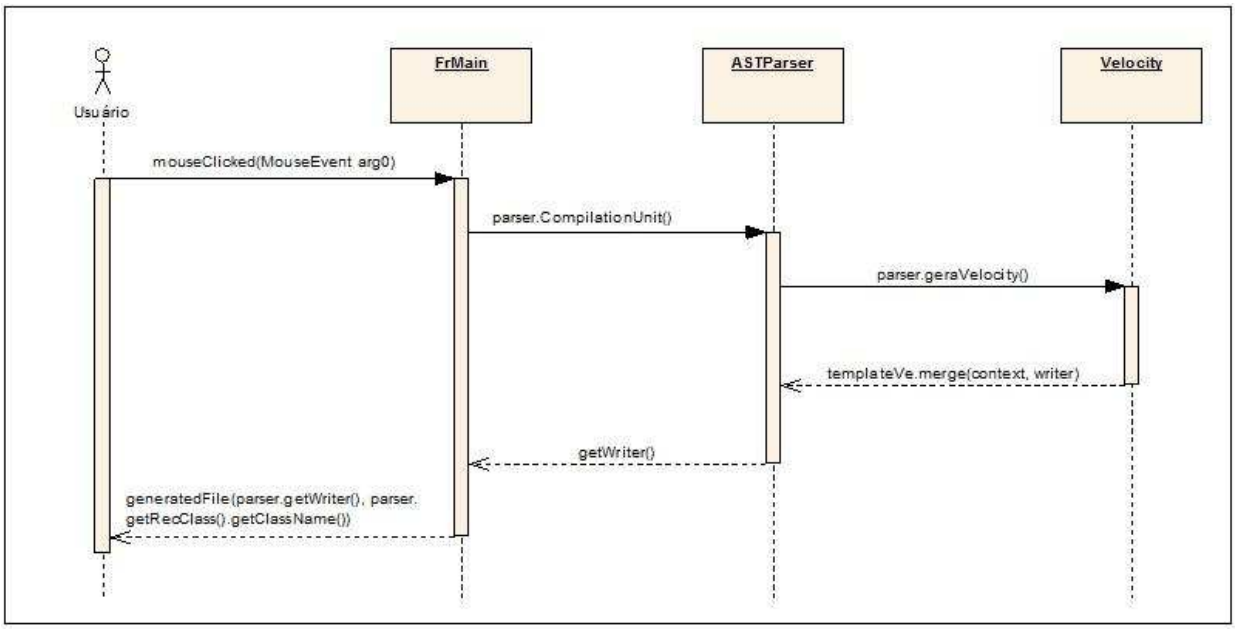


Figura 15 – Diagrama de seqüência do caso de uso `Converter código`

3.6 IMPLEMENTAÇÃO

Nesta seção são descritas as ferramentas e técnicas utilizadas, bem como a implementação do trabalho.

3.6.1 Técnicas e ferramentas utilizadas

A ferramenta aqui apresentada foi construída utilizando a linguagem Java 5.0, com ambiente de desenvolvimento Eclipse 3.2.2 e a biblioteca gráfica Swing. O motor de *templates* Velocity foi utilizado através da biblioteca `velocity-dep-1.5.jar`. Também fez-se uso do JavaCC para gerar os analisadores léxico e sintático.

3.6.2 Implementação da ferramenta

O primeiro passo para a implementação da ferramenta foi identificar quais propriedades dos componentes deveriam ser armazenadas para a geração do código em HTML. Para análise do código `.java` foram utilizados analisadores léxico e sintático. Para gerar as classes responsáveis pelos analisadores léxico e sintático foi adaptada a gramática do Java especificada por Gesser (2007, p. 77). A ferramenta assume que o arquivo `.java` está semanticamente correto, não faz análise semântica, apenas utiliza o analisador semântico para extrair os dados necessários para a conversão. Para tanto, definiram-se métodos para as ações semânticas, para reconhecimento de *tokens* utilizados na geração de código desejada.

No quadro 20 é apresentado um trecho do código presente na classe `FrMain`. Este código é executado toda vez em que o botão **Converter** é pressionado.

```
public void mouseClicked(MouseEvent arg0){
  compile.ASTParser parser = null;
  try {
    if (files.size() == 0) {                                          trecho 01
      OptionPane.showMessageDialog(null, "Selecione um arquivo.");
    } else {
      if (template == null) {                                         trecho 02
        JOptionPane.showMessageDialog(null, "Informe um template.");
      } else {
        for (int i = 0; i < files.size(); i++) {                      trecho 03
          String nameFile = files.get(i);
          parser = new ASTParser(new  java.io.FileInputStream(nameFile));
          try {
            parser.CompilationUnit();
            try {
              parser.positionJPanels();
              try {
                parser.positionComponents();
                parser.setTemplate(template);
                parser.geraVelocity();
                try{
                  generatedFile(parser.getWriter(),
                  parser.getRecClass().getClassName());
                  JOptionPane.showMessageDialog(null,"Arquivo convertido ...!")
                } catch (Exception e) {
                  JOptionPane.showMessageDialog(null,"Erro ao converter código!");}
              } catch (Exception e) {
                JOptionPane.showMessageDialog(null,"Erro ao converter código!");}
            } catch (Exception e) {
               JOptionPane.showMessageDialog(null,"Erro ao converter código!");}
          } catch (Exception e) {
             JOptionPane.showMessageDialog(null,Erro ao converter código!");} } } }
  } catch (java.io.FileNotFoundException e) {
    System.out.println("Arquivo não encontrado.");
    JOptionPane.showMessageDialog(null,"Erro ao converter código! ");
    return;
}                        }
```

Quadro 20 – Conversão de código

Primeiramente é verificado se existem arquivo(s) a ser(em) convertido(s) (trecho 01). Caso nenhum arquivo tenha sido informado, é apresentada uma mensagem, pedindo para que o usuário selecione um arquivo. Posteriormente é verificado se um *template* foi especificado (trecho 02). Também neste caso, caso não tenha sido especificado, é apresentada uma mensagem pedindo ao usuário para que informe um *template*. Tendo ao menos um arquivo e um *template*, a ferramenta percorre a lista de arquivos a serem convertidos, criando um *stream* de dados do arquivo informado (trecho 03). Logo após é efetuada a chamada ao método `CompilationUnit` responsável por percorrer toda a estrutura do arquivo e realizar as análises léxica e sintática do arquivo. Caso não sejam detectados erros léxicos ou sintáticos, são chamados: o método `positionJPanels`, para posicionar os *panels* dentro do formulário, e o método `positionComponents` responsável por posicionar os componentes. Entende-se por posicionar *panels* e componentes, o cálculo para definir as posições `top` e `left` dos mesmos. Na seqüência, o *template* informado é setado e o método `geraVelocity` é chamado para gerar o código de saída baseado no *template* informado. Por último, o método `generatedFile` é

chamado para gravar a saída gerada pelo Velocity em um arquivo, que possui o mesmo nome da classe `.java`.

3.6.3 Operacionalidade da implementação

Nesta seção é apresentado um estudo de caso para demonstrar a funcionalidade da ferramenta SwingToHtml (figura 16). A ferramenta é utilizada para gerar código da camada de interface de uma aplicação web a partir de uma aplicação desenvolvida em Java Swing. O resultado são páginas HTML. Outra funcionalidade da ferramenta é a conversão de eventos Java para JavaScript.

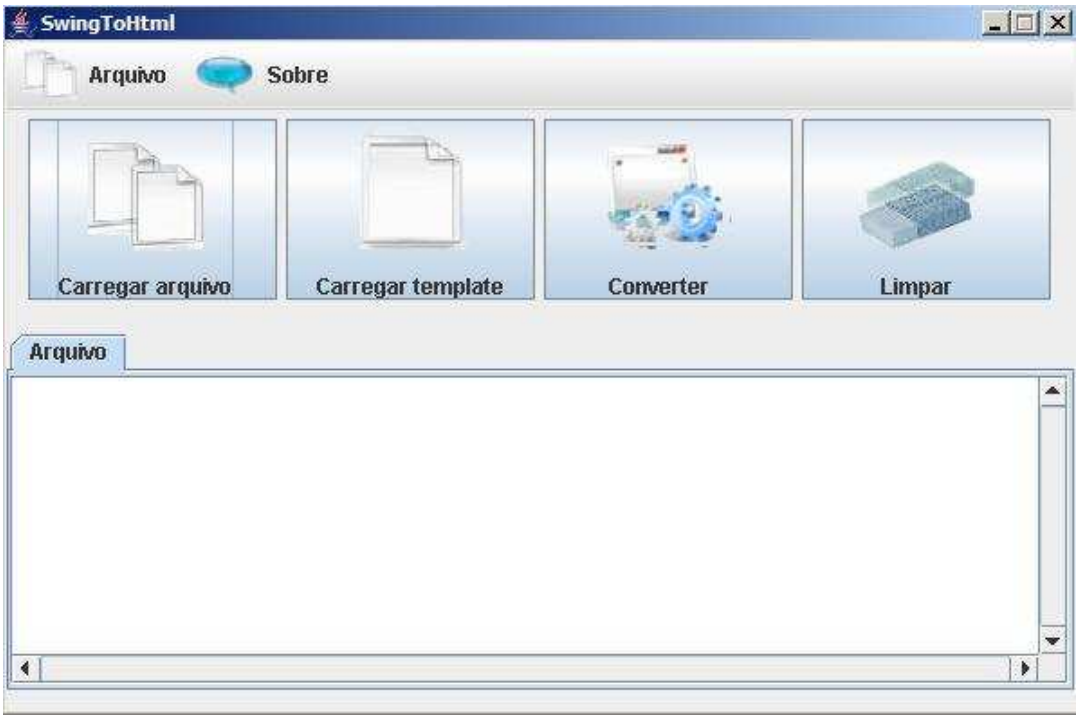


Figura 16 – Tela da ferramenta SwingToHtml

Ao pressionar o botão **Carregar arquivo**, é possível selecionar os arquivos `.java` que serão convertidos (figura 17). Os arquivos selecionados são listados na aba **Arquivo** (figura 18).

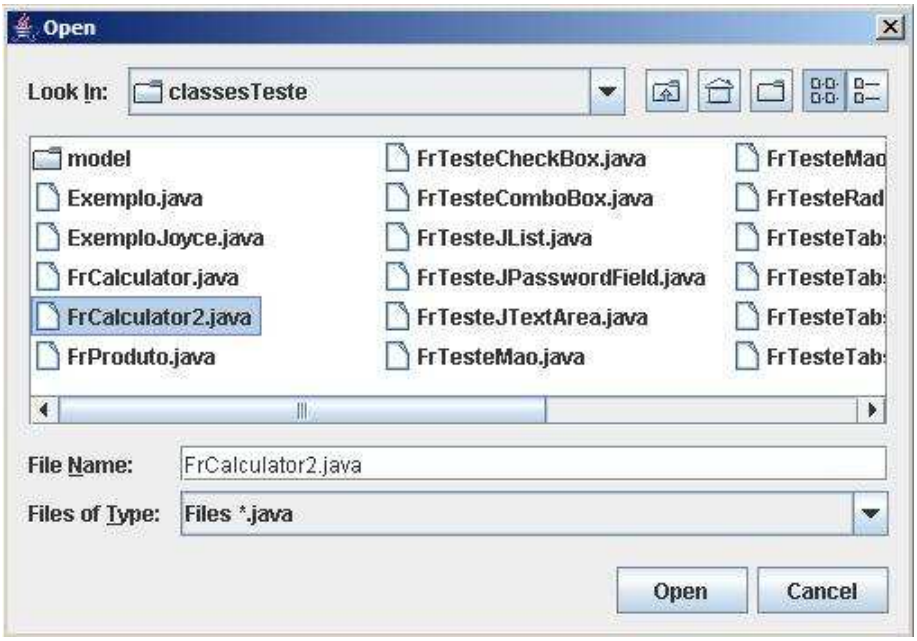


Figura 17 – Seleção de arquivo `.java`

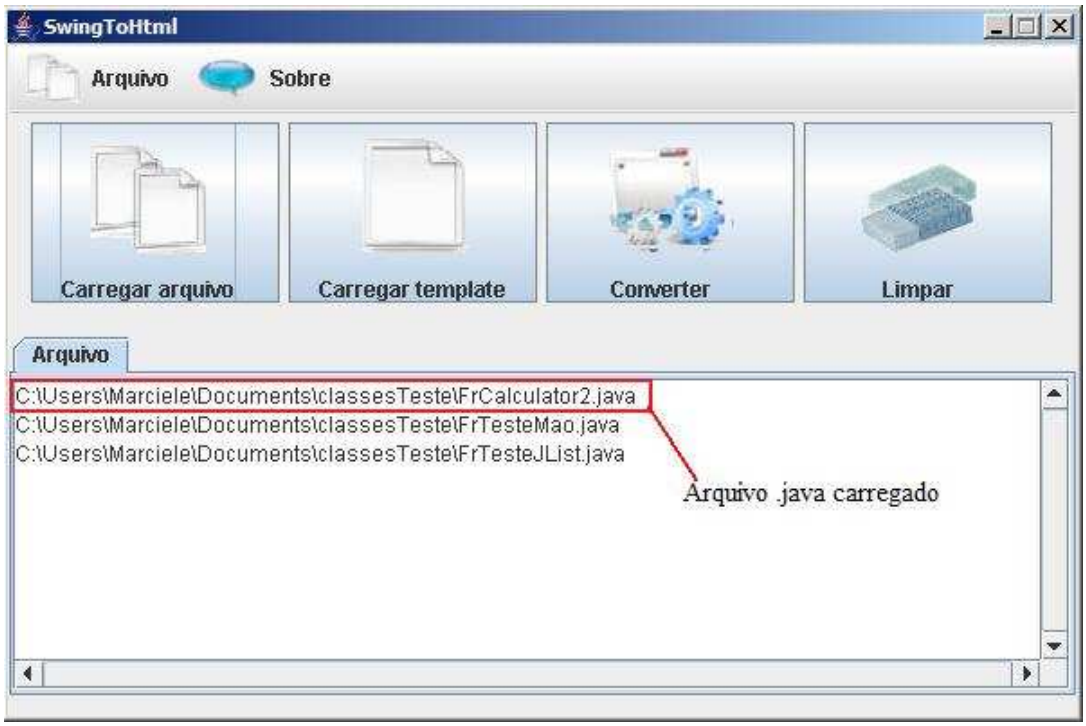


Figura 18 – Arquivos `.java` selecionados

Antes de efetuar a conversão, o usuário deve informar qual o *template* modelo deverá ser usado. Para tanto, deve pressionar o botão **Carregar template** que abrirá a janela apresentada na figura 19.

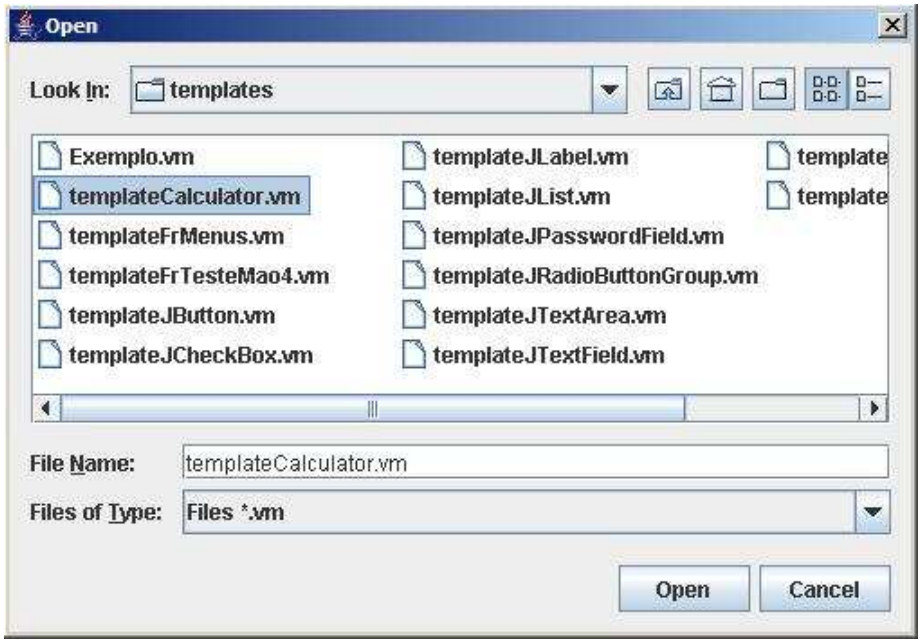


Figura 19 – Seleção de *template*

Depois de selecionados os arquivos `.java` e o *template*, o usuário pode então pressionar o botão **Converter**. Existem duas situações possíveis: o arquivo `.java` pode ser convertido com sucesso (figura 20), ou pode ocorrer algum erro durante o processo de conversão. Quando convertido com sucesso, os arquivos são gerados no diretório principal da ferramenta.

Figura 20 – Conversão de arquivos

Na figura 21 são apresentadas informações sobre a ferramenta.

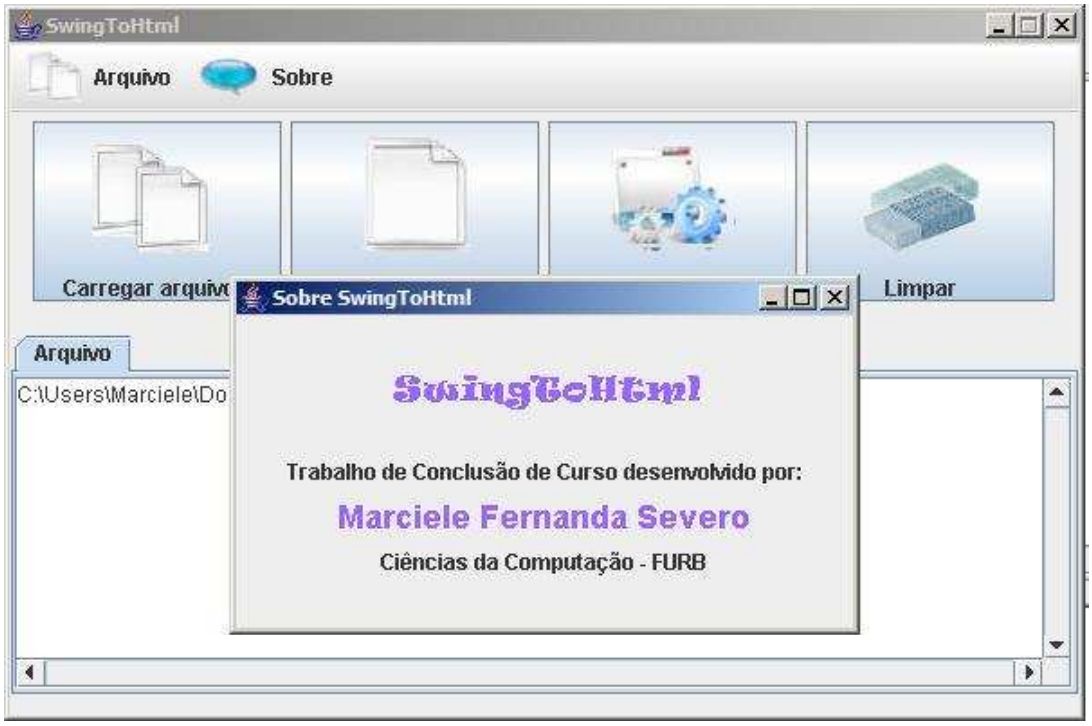


Figura 21 – Tela Sobre a ferramenta SwingToHtml

Por fim, a partir de uma interface Java Swing (figura 22), foi gerado o arquivo `.html` correspondente (figura 23).

Figura 22 – Interface Java Swing

Figura 23 – Interface HTML

Para demonstração da conversão de eventos utilizou-se um exemplo de calculadora. Na figura 24 é apresentada a interface Java Swing da calculadora e na figura 25 a interface correspondente em HTML.

Figura 24 – Interface da calculadora em Java Swing

Figura 25 – Interface da Calculadora em HTML

No quadro 21 é apresentado um trecho do código `.java` e no quadro 22 a conversão para JavaScript. O código completo encontra-se no Apêndice A.

```
...

textfield1.addKeyListener ( new KeyListener() { ...
   public void keyReleased(KeyEvent e) {
      val1 = textfield1.getText();
   } ...
});

textfield3.addKeyListener ( new KeyListener() { ...
   public void keyReleased(KeyEvent e) {
      val2 = textfield3.getText();
   } ...
});

jbutton.addMouseListener ( new MouseListener() {
   public void mouseClicked(MouseEvent e) {
      textfield2.setText("+");
         if (op != ""){
            val1 = textfield4.getText();
            textfield1.setText(val1);
            textfield3.setText("");
         }
         op = "+";
   } ...
});
...
```

Quadro 21 – Eventos em `.java`

```
<script>
   val1 = "0"
   val2 = "0"
   op = ""

   function keyReleasedtextfield1(){
      val1 = document.FrCalculator2.textfield1.value
   }

   function keyReleasedtextfield3(){
      val2 = document.FrCalculator2.textfield3.value
   }

   function mouseClickedjbutton(){
      document.FrCalculator2.textfield2.value= '+'
      if ( op != '' ) {
         val1 = document.FrCalculator2.textfield4.value
         document.FrCalculator2.textfield1.value= val1
         document.FrCalculator2.textfield3.value= ''
      }
      op =  '+'
   }
   ...
</script>
```

Quadro 22 – Eventos em JavaScript

Para melhor exemplificar a utilização de *templates*, no quadro 23 é apresentado o *template* utilizado para geração do código da calculadora. Nota-se o uso da diretiva `#parse`, que simplifica o código. Nesse *template*, o código dos componentes `JTextField`, `JLabel` e `JButton` são interpretados a partir de outros *templates*. No quadro 24 é exemplificado o código do *template* `templateJButton.vm` interpretando quando um componente `JButton` é identificado.

```
<HTML>
   <HEAD>
   <TITLE> $!{JFrameGetTitle} </TITLE>
   </HEAD>
<BODY>
$!{funcJavaScript}
   <fieldset
          name= $!{jcontentPanelGetName}
          style="position: absolute;
          font-family: $!{jcontentPanelGetFont};
          font-size: $!{jcontentPanelGetFontSize};
          color: $!{jcontentPanelGetFontColor};
          background-color: $!{jcontentPanelGetBackgroudColor};
          left: $!{jcontentPanelGetPositionLeft};
          top: $!{jcontentPanelGetPositionTop};
          width: $!{jcontentPanelGetWidth};
          height: $!{jcontentPanelGetHeight};">

   <form name= $!{JFrameGetName}>
   #foreach ($ComponentJPanel in $arrayJPanel)
          <fieldset
                 name= $!{ComponentJPanel.getName()}
                 style="position: absolute;
                 font-family: $!{ComponentJPanel.getFontFamily()};
                 font-size: $!{ComponentJPanel.getFontSize()};
                 width: $!{ComponentJPanel.getSizeWidth()};
                 height: $!{ComponentJPanel.getSizeHeight()};
                 border: 0;
                 left: $!{ComponentJPanel.getPositionLeft()};
```

```
                top: $!{ComponentJPanel.getPositionTop()};
                text-align= $!{style}">
          #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
             #if ($!{ComponentJPanel.getLayout()} == "FlowLayout_Center")
                <div style="text-align:center;">
             #end
             #foreach ($Component in $ComponentJPanel.getChildrenComponents())
                #if ($!{Component.getType()} == "JTextField")
                   #parse("templateJTextField.vm")
                #end
                #if ($!{Component.getType()} == "JLabel")
                   #parse("templateJLabel.vm")
                #end
                #if ($!{Component.getType()} == "JButton")
                   #parse("templateJButton.vm")
                #end
             #end
             #if ($!{ComponentJPanel.getLayout()} == "FlowLayout_Center")
                </div>
             #end
          #end
          </fieldset>
   #end
   </form>
   </fieldset>
</BODY>
</HTML>
```

Quadro 23 – *Template* utilizado na conversão da calculadora

```
<button
   name= $!{Component.getName()} id= id$!{Component.getName()}
   style="position: relative;
   font-family: $!{Component.getFontFamily()};
   font-size: $!{Component.getFontSize()};
   width: $!{Component.getSizeWidth()}; height: $!{Component.getSizeHeight()};
   left: $!{Component.getPositionLeft()}; top: $!{Component.getPositionTop()};"
   $!{Component.getFunctionJavaScript().getMethod()}>
    $!{Component.getText()}
</button>
```

Quadro 24 – *Template* `templateJbutton.vm`

Salienta-se que para que a ferramenta converta o código `.java` corretamente, os componentes de interface devem ser adicionados nos *panels* conforme a ordem que aparecem na visualização. É importante frisar que não são convertidos componentes adicionados diretamente no componente `JFrame`, os componentes precisam estar dentro de *panels*. Definiu-se essa restrição porque geralmente os desenvolvedores utilizam *panels* para melhor organizar os componentes de interface. Para tanto, SwingToHtml assume que existe um *panel* principal, sendo os demais *panels* adicionados a esse *panel* principal. Também não são reconhecidos corretamente componentes que possuem pacote em sua declaração (`javax.swing.JPanel`), sendo que para sua correta conversão, basta apenas declarar `JPanel`. Definiu-se desta forma porque geralmente são importados os pacotes que contêm os componentes no início da classe. Os *layouts* convertidos são: `FlowLayout`, centralizado e alinhado à esquerda; `BoxLayout`, alinhado horizontalmente; e quando o *layout* não está definido, por *default*, no código gerado os componentes são alinhados à esquerda. No

Apêndice C, é apresentado um *template* para conversão de todos os componentes apresentados no quadro 13, exceto os componentes de menu, em função do problema relatado anteriormente.

Para os eventos, são convertidas declarações de variáveis, comandos condicionais (`if-else`) e expressões aritméticas. Os eventos convertidos pela ferramenta estão listados no quadro 14.

3.7 RESULTADOS E DISCUSSÃO

A ferramenta proposta não atendeu completamente os objetivos inicialmente estabelecidos, uma vez que não foi gerado código JSP. Na proposta pensou-se em converter a interface Java Swing para HTML, os tratadores de eventos para JavaScript e os métodos por estes chamados para JSP. Quanto aos tratadores de eventos, a ferramenta implementada converte para JavaScript a chamada a todos os métodos relacionados no quadro 14. Não foi possível a conversão completa do código dos métodos dos tratadores de eventos em função da complexidade da linguagem Java.

No quadro 25 é apresentada uma comparação entre a ferramenta SwingToHtml e os trabalhos correlatos descritos, levando em consideração suas principais características.

CARACTERÍSTICAS	**AjaxSwing**	**DelphiToWeb**	**Converte Forms**	**SwingToHtml**
conversão de componentes de interface	sim	22	15	14
conversão de tratamento de eventos	sim	não	sim	sim
linguagem de entrada	arquivos `.java`	arquivos .DFM	arquivos Oracle Forms	arquivos `.java`
linguagem de saída	HTML, AJAX	HTML, .LZX, XML	`.java`	HMTL, JavaScript
uso de analisadores (léxico, sintático e semântico) para leitura dos arquivos de entrada	-	sim	sim	sim
uso de *templates* para geração de código	-	não	sim	sim

Quadro 25 – Comparação SwingToHtml X trabalhos correlatos

Realizando testes de comparação entre as ferramentas apresentadas no quadro acima, observou-se que o código gerado pela ferramenta DelphiToWeb apresenta uma interface mais

próxima da original quando se trata de espaçamentos entre componentes e até a própria localização dos componentes dentro dos formulários. Isso se deve à forma de criação de componentes de interface na linguagem Delphi, onde cada um possui como propriedade a posição dentro do formulário, o que não ocorre em Java.

Das ferramentas relacionadas acima, a mais semelhante à ferramenta implementada é AjaxSwing. Embora as informações contidas no *site* do produto sejam superficiais (não são apresentados quais componentes são convertidos, não são citados eventos), encontrou-se uma breve descrição sobre suas limitações, podendo citar problemas de desempenho na conversão de alguns eventos, como por exemplo, o não tratamento de movimentos de *mouse*.

4 CONCLUSÕES

Diante das vantagens apresentadas pelos sistemas web e a demanda das empresas por estes sistemas, implementou-se uma ferramenta para a conversão de interfaces de aplicações desenvolvidas em Java para páginas HTML e eventos em JavaScript. A ferramenta apresenta-se como uma opção para agilizar o desenvolvimento de aplicações uma vez que permite a migração de produtos desenvolvidos em Java para HTML, reduzindo assim o tempo de elaboração da interface e alguns tratamentos de eventos.

As informações necessárias para gerar a saída são extraídas de arquivos Java, contendo componentes de interface Swing e tratadores de eventos. São utilizados *templates* para servir de modelo para o código de saída, a fim de agilizar o desenvolvimento, aumentar a produtividade e garantir a qualidade e consistência das páginas geradas. Para desenvolvimento deste trabalho, optou-se pelo motor de *templates* Velocity. Segundo Lozano (2003), o forte do Velocity está na facilidade de reorganizar a estrutura de navegação de uma aplicação web e modificar elementos comuns como cabeçalhos e menus. SwingToHtml gera páginas HTML e JavaScript. Foi necessário o uso da linguagem JavaScript para a criação de alguns componentes visto as limitações dos recursos da linguagem HTML. Alguns componentes, como `JTabbedPane` e `JMenu`, não têm equivalência em HTML. Dessa forma, para criação desses componentes utilizou-se a linguagem JavaScript a fim de manter as funcionalidades presentes em Java Swing, como alterar de uma *tab* para outra e abrir um botão de menu. Para definição de componentes como os citados anteriormente, utilizou-se como referência Souza (2005) e para análise do código de entrada baseou-se no trabalho desenvolvido por Gesser (2007).

A ferramenta atingiu seu maior objetivo, a conversão de interfaces Java Swing, agregando conhecimentos em HTML, JavaScript e utilização de *templates*. A maior dificuldade no desenvolvimento da ferramenta foi o posicionamento dos componentes de interface nas páginas HTML equivalentes, visto que estas informações não são obtidas do código Java e tiveram de ser calculadas. Mas com a ferramenta consegue-se chegar muito próximo do *layout* original.

4.1 EXTENSÕES

Como extensões para este trabalho sugere-se:

a) implementar a conversão de mais componentes já que a ferramenta está limitada à conversão de 14 componentes;
b) aumentar a implementação dos tratadores de eventos, a fim de que se possa ampliar a conversão do código dentro dos tratadores;
c) criar *templates* para gerar código para tratamento de eventos (métodos);
d) melhorar o posicionamento dos componentes de interface, a fim de chegar mais próximo do *layout* original.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVAREZ, M. A. **Introdução a JavaScript:** o que é JavaScript e as possibilidades que nos oferece em relação ao HTML. [S.l.], [2008?]. Disponível em: <http://www.criarweb.com/artigos/156.php> . Acesso em: 25 out. 2008.

COHEN, M. **Tratamento de eventos**. [Porto Alegre], 2008. Disponível em: <http://www.inf.pucrs.br/~flash/lapro2/lapro2_eventos.pdf>. Acesso em: 08 jun. 2008.

CREAMTEC. **Automatic conversion of Swing applications to AJAX websites.** [S.l.], 2008. Disponível em: <http://creamtec.com/products/ajaxswing/index.html>. Acesso em: 26 abr. 2008.

CRIARWEB. **Programação em JavaScript**. [S.l.], 2004. Disponível em: <http://www.criarweb.com/artigos/157.php>. Acesso em: 17 nov. 2008.

DEITEL, H. M.; DEITEL, P. J. **Java:** como programar. 4.ed. Tradução Carlos Arthur Lang Lisboa. Porto alegre: Bookmam, 2003.

GESSER, J. V. **Compilador Java 5.0 para gerar código C++ para plataforma Palm OS**. 2007. 84 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HERRINGTON, J. **Code generation in action**. Greenwich: Manning, 2003.

HTML. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2008. Disponível em: <http://pt.wikipedia.org/wiki/HTML>. Acesso em: 08 jun. 2008.

LOZANO, F. **Software livre para o desenvolvedor Java**. [S.l.], 2003. Disponível em: <http://www.lozano.eti.br/palestras/java-soft-livre-2.pdf>. Acesso em: 01 maio 2008.

MACORATTI, J. C. **Desenvolvendo para desktop ou para web?** [S.l.], [2006]. Disponível em: <http://www.macoratti.net/vbn_dkwb.htm>. Acesso em: 27 maio 2008.

MEIRA, S. **A nova aristocracia.** [Recife], 2004. Disponível em: <http://www.silviomeira.globolog.com.br/archive_2007_03_30_11.html>. Acesso em: 08 jun. 2008.

ROCHA, L. **APE:** plataforma para o desenvolvimento de aplicações web com PHP. [S.l.], 2005. Disponível em: <http://twiki.im.ufba.br/bin/view/Aside/ProjetoConclusaoDeCursoAPEMonografia#4_2_Motores_de_templates>. Acesso em: 01 maio 2008.

SCHVEPE, C. **Gerador de código Java a partir de arquivos do Oracle Forms 6i**. 2006. 76 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVEIRA, P. E. A. **Jakarta Velocity:** acabe a briga com os designers. [S.l.], [2008?]. Disponível em: <http://www.devmedia.com.br/space.asp?id=195867>. Acesso em: 25 out. 2008.

SOARES FILHO, A. **Migrar software reduz custos e economiza tempo**. [S.l.], 2003. Disponível em: <http://webinsider.uol.com.br/index.php/2003/08/20/migrar-software-reduz-custos-e-economiza-tempo/>. Acesso em: 03 maio 2008.

SOMMERVILLE, I. **Engenharia de software**. 6. ed. Tradução Maurício de Andrade. São Paulo: Addison Wesley, 2003.

SOUZA, A. **Ferramenta para conversão de formulários Delphi em páginas HTML**. 2005. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

STEIL, R. **Introdução a programação gráfica em Java com Swing**. [S.l.], [2006a?]. Disponível em: < http://www.guj.com.br/java.tutorial.artigo.38.1.guj>. Acesso em: 08 jun. 2008.

STEIL, R. **Introdução ao Velocity**. [S.l.], [2006b?]. Disponível em: <http://guj.com.br/user.article.get.chain?page=1&article.id=18>. Acesso em: 01 maio 2008.

APÊNDICE A – Código de entrada e saída: Calculadora

Nos quadros 26 e 27 são apresentados, respectivamente, o código da calculadora em Java, entrada da ferramenta SwingToHtml, e em HTML e JavaScript, saída da ferramenta.

```
public class FrCalculator2 extends JFrame {
	private JTextField textfield1 = null;
	private JTextField textfield2 = null;
	private JTextField textfield3 = null;
	private JTextField textfield4 = null;
	private JLabel jlabel = null;
	private String val1 = "0";
	private String val2 = "0";
	private String op = "";

	public FrCalculator2() {
		this.setSize(new Dimension(260, 130));
		this.setTitle("Calculadora");
		JPanel contentPanel = new JPanel();
		contentPanel.setLayout(new FlowLayout(FlowLayout.CENTER,0,0));
		contentPanel.add(getPanel());
		contentPanel.add(getPanel2());
		contentPanel.add(getPanel3());
		this.setContentPane(contentPanel);
		this.setDefaultCloseOperation(FrCalculator.EXIT_ON_CLOSE);
	}

	private JPanel getPanel() {
		JPanel panel1 = new JPanel();
		panel1.setLayout(new FlowLayout(FlowLayout.CENTER,0,0));
		panel1.setPreferredSize(new Dimension(260, 35));
		textfield1 = new JTextField();
		textfield1.setPreferredSize(new Dimension(70, 30));
		textfield1.addKeyListener(new KeyListener() {
			public void keyPressed(KeyEvent e) {}
			public void keyReleased(KeyEvent e) {
				val1 = textfield1.getText();
			}
			public void keyTyped(KeyEvent e) {}
		});
		panel1.add(textfield1);

		textfield2 = new JTextField();
		textfield2.setPreferredSize(new Dimension(20, 30));
		textfield2.setEnabled(false);
		panel1.add(textfield2);

		textfield3 = new JTextField();
		textfield3.setPreferredSize(new Dimension(70, 30));
		textfield3.addKeyListener(new KeyListener() {
			public void keyPressed(KeyEvent e) {}
			public void keyReleased(KeyEvent e) {
				val2 = textfield3.getText();
			}
			public void keyTyped(KeyEvent e) {}
		});
		panel1.add(textfield3);

		jlabel = new JLabel();
		jlabel.setPreferredSize(new Dimension(20, 30));
		jlabel.setText("  =");
		panel1.add(jlabel);

		textfield4 = new JTextField();
```

```
		textfield4.setPreferredSize(new Dimension(70, 30));
		textfield4.setEnabled(false);
		panel1.add(textfield4);

		return panel1;
	}

	private JPanel getPanel2() {
		JPanel panel2 = new JPanel();
		panel2.setLayout(new FlowLayout(FlowLayout.CENTER,0,0));
		panel2.setPreferredSize(new Dimension(260, 35));
		JButton jbutton = new JButton();
		jbutton.setText("+");
		jbutton.setPreferredSize(new Dimension(50, 30));
		jbutton.addMouseListener(new MouseListener() {
			public void mouseClicked(MouseEvent e) {
				textfield2.setText("+");
				if (op != ""){
					val1 = textfield4.getText();
					textfield1.setText(val1);
					textfield3.setText("");
				}
				op = "+";
			}
			public void mouseEntered(MouseEvent e) {}
			public void mouseExited(MouseEvent e) {}
			public void mousePressed(MouseEvent e) {}
			public void mouseReleased(MouseEvent e) {}
		});
		panel2.add(jbutton);

		JButton jbutton2 = new JButton();
		jbutton2.setText("-");
		jbutton2.setPreferredSize(new Dimension(50, 30));
		jbutton2.addMouseListener(new MouseListener() {
			public void mouseClicked(MouseEvent e) {
				textfield2.setText("-");
				if (op !=""){
					val1 = textfield4.getText();
					textfield1.setText(val1);
					textfield3.setText("");
				}
				op = "-";
			}
			public void mouseEntered(MouseEvent e) {}
			public void mouseExited(MouseEvent e) {}
			public void mousePressed(MouseEvent e) {}
			public void mouseReleased(MouseEvent e) {}
		});
		panel2.add(jbutton2);

		JButton jbutton3 = new JButton();
		jbutton3.setText("/");
		jbutton3.setPreferredSize(new Dimension(50, 30));
		jbutton3.addMouseListener(new MouseListener() {

			public void mouseClicked(MouseEvent e) {
				textfield2.setText("/");
				if (op != ""){
					val1 = textfield4.getText();
					textfield1.setText(val1);
					textfield3.setText("");
				}
				op = "/";
			}
			public void mouseEntered(MouseEvent e) {}
			public void mouseExited(MouseEvent e) {}
			public void mousePressed(MouseEvent e) {}
```

```
                public void mouseReleased(MouseEvent e) {}
        });
        panel2.add(jbutton3);

        JButton jbutton4 = new JButton();
        jbutton4.setText("*");
        jbutton4.setPreferredSize(new Dimension(50, 30));
        jbutton4.addMouseListener(new MouseListener() {
                public void mouseClicked(MouseEvent e) {
                        textfield2.setText("*");
                        if (op != ""){
                                val1 = textfield4.getText();
                                textfield1.setText(val1);
                                textfield3.setText("");
                        }
                        op = "*";
                }
                public void mouseEntered(MouseEvent e) {}
                public void mouseExited(MouseEvent e) {}
                public void mousePressed(MouseEvent e) {}
                public void mouseReleased(MouseEvent e) {}
        });
        panel2.add(jbutton4);

        JButton jbutton5 = new JButton();
        jbutton5.setText("=");
        jbutton5.setPreferredSize(new Dimension(50, 30));
        jbutton5.addMouseListener(new MouseListener() {
                public void mouseClicked(MouseEvent e) {
                        double res = 0;
                        if (op == "+") {
                                res = Double.parseDouble(val1) +
                                                Double.parseDouble(val2);
                        } else if (op == "-") {
                                res = Double.parseDouble(val1) -
                                                Double.parseDouble(val2);
                        } else if (op == "/") {
                                res = Double.parseDouble(val1) /
                                                Double.parseDouble(val2);
                        } else if (op == "*") {
                                res = Double.parseDouble(val1) *
                                                Double.parseDouble(val2);
                        }
                        textfield4.setText(String.valueOf(res));
                }
                public void mouseEntered(MouseEvent e) {}
                public void mouseExited(MouseEvent e) {}
                public void mousePressed(MouseEvent e) {}
                public void mouseReleased(MouseEvent e) {}
        });
        panel2.add(jbutton5);

        return panel2;
    }

    private JPanel getPanel3() {
        JPanel panel3 = new JPanel();
        panel3.setLayout(new FlowLayout(FlowLayout.CENTER, 0, 0));
        panel3.setPreferredSize(new Dimension(260, 35));
        JButton jbutton6 = new JButton();
        jbutton6.setText("Limpar");
        jbutton6.setPreferredSize(new Dimension(250, 30));
        jbutton6.addMouseListener(new MouseListener() {
                public void mouseClicked(MouseEvent e) {
                        textfield1.setText("");
                        textfield2.setText("");
                        textfield3.setText("");
                        textfield4.setText("");
```

```
                        val1 = "0";
                        val2 = "0";
                        op="";
                    }
                    public void mouseEntered(MouseEvent e) {}
                    public void mouseExited(MouseEvent e) {}
                    public void mousePressed(MouseEvent e) {}
                    public void mouseReleased(MouseEvent e) {}
            });
            panel3.add(jbutton6);

            return panel3;
      }

      public static void main(String[] args) {
            FrCalculator2 c = new FrCalculator2();
            c.setVisible(true);
            c.setLocationRelativeTo(null);
            c.setDefaultCloseOperation(FrCalculator2.EXIT_ON_CLOSE);
      }
}
```

Quadro 26 – Código Java: Calculadora

```
<HTML>
  <HEAD>
  <TITLE> Calculadora </TITLE>
  </HEAD>
<BODY>
  <script>
    val1 = "0"
    val2 = "0"
    op = ""

    function keyReleasedtextfield1(){
      val1 = document.FrCalculator2.textfield1.value
    }

    function keyReleasedtextfield3(){
      val2 = document.FrCalculator2.textfield3.value
    }

    function mouseClickedjbutton(){
      document.FrCalculator2.textfield2.value= '+'
      if (op != '') {
        val1 = document.FrCalculator2.textfield4.value
        document.FrCalculator2.textfield1.value= val1
        document.FrCalculator2.textfield3.value= ''
      }
      op =  '+'
    }

    function mouseClickedjbutton2(){
        document.FrCalculator2.textfield2.value= '-'
        if ( op != '' ) {
          val1 = document.FrCalculator2.textfield4.value
          document.FrCalculator2.textfield1.value= val1
          document.FrCalculator2.textfield3.value= ''
        }
        op =  '-'
    }

    function mouseClickedjbutton3(){
      document.FrCalculator2.textfield2.value= '/'
      if ( op != '' ) {
        val1 = document.FrCalculator2.textfield4.value
        document.FrCalculator2.textfield1.value= val1
        document.FrCalculator2.textfield3.value= ''
      }
```

```
      op =  '/'
    }
    function mouseClickedjbutton4(){
      document.FrCalculator2.textfield2.value= '*'
      if ( op != '' ) {
        val1 = document.FrCalculator2.textfield4.value
        document.FrCalculator2.textfield1.value= val1
        document.FrCalculator2.textfield3.value= ''
      }
      op =  '*'
    }

    function mouseClickedjbutton5(){
      var res = 0
      if ( op == '+' ) {
        res =  Number(val1) + Number(val2)
      }
      else
        if ( op == '-' ) {
          res =  Number(val1) - Number(val2)
        } else
          if ( op == '/' ) {
            res =  Number(val1) / Number(val2)
          } else  if ( op == '*' ) {
              res =  Number(val1) * Number(val2)
          }
      document.FrCalculator2.textfield4.value= Number(res)
    }

    function mouseClickedjbutton6(){
      document.FrCalculator2.textfield1.value= ''
      document.FrCalculator2.textfield2.value= ''
      document.FrCalculator2.textfield3.value= ''
      document.FrCalculator2.textfield4.value= ''
      val1 = 0
      val2 = 0
      op =  ''
    }
  </script>

  <fieldset
     name= contentPanel
     style="position: absolute;
     font-family: dialog; font-size: 12; color: #000000; background-color: #CCCCCC;
     left: 0; top: 0; width: 260; height: 130;">
     <form name= FrCalculator2>
        <fieldset
           name= panel1
           style="position: absolute;
           font-family: dialog; font-size: 1;
           width: 260; height: 35;
           border: 0; left: 0; top: 0; text-align= center">
           <div style="text-align:center;">
               <input type="edit"
                  name= textfield1
                  style= "position: absolute;
                  font-family: dialog; font-size: 12;
                  width: 70; height: 30;
                  left: 0; top: 0;"
                  onKeyUp=keyReleasedtextfield1()>
               </input>
               <input type="edit"
                  name= textfield2
                  style= "position: absolute;
                  font-family: dialog; font-size: 12;
                  width: 20; height: 30;
                  left: 70; top: 0;"
               </input>
```

```
            <input type="edit"
               name= textfield3
               style= "position: absolute;
               font-family: dialog; font-size: 12;
               width: 70; height: 30;
               left: 90; top: 0;"
               onKeyUp=keyReleasedtextfield3()>
            </input>
            <font
               name=jlabel
               style= "position: absolute;
               font-family: dialog; font-size: 12; font-weight: bold;
               width: 20; height: 30;
               left: 160; top: 0;">
               =
            </font>
            <input type="edit"
               name= textfield4
               style= "position: absolute;
               font-family: dialog; font-size: 12;
               width: 70; height: 30;
               left: 180; top: 0;"
               onKeyUp=keyReleasedtextfield3()>
            </input>
         </div>
      </fieldset>
      <fieldset
         name= panel2
         style="position: absolute;
         font-family: dialog; font-size: 12;
         width: 260; height: 35;
         border: 0; left: 0; top: 35; text-align= center">
         <div style="text-align:center;">
            <button
               name= jbutton id= idjbutton
               style="position:absolute;
               font-family: dialog; font-size: 12;
               width: 50; height: 30;
               left: 0; top: 0;"
               onClick=mouseClickedjbutton()>
               +
            </button>
            <button
               name= jbutton2 id= idjbutton2
               style="position:absolute;
               font-family: dialog; font-size: 12;
               width: 50; height: 30;
               left: 50; top: 0;"
               onClick=mouseClickedjbutton2()>
               -
            </button>
            <button
               name= jbutton3 id= idjbutton3
               style="position:absolute;
               font-family: dialog; font-size: 12;
               width: 50; height: 30;
               left: 100; top: 0;"
               onClick=mouseClickedjbutton3()>
               /
            </button>
            <button
               name= jbutton4 id= idjbutton4
               style="position:absolute;
               font-family: dialog; font-size: 12;
               width: 50; height: 30;
               left: 150; top: 0;"
               onClick=mouseClickedjbutton4()>
               *
```

```
               </button>
               <button
                  name= jbutton5 id= idjbutton5
                  style="position:absolute;
                  font-family: dialog; font-size: 12;
                  width: 50; height: 30;
                  left: 200; top: 0;"
                  onClick=mouseClickedjbutton5()>
                  =
               </button>
            </div>
         </fieldset>
         <fieldset
            name= panel3
            style="position: absolute;
            font-family: dialog; font-size: 1;
            width: 260; height: 35;
            border: 0; left: 0; top: 70; text-align= center">
            <div style="text-align:center;">
               <button
                  name= jbutton6 id= idjbutton6
                  style="position:absolute;
                  font-family: dialog; font-size: 12;
                  width: 250; height: 30;
                  left: 2; top: 0;"
                  onClick=mouseClickedjbutton6()>
                  Limpar
               </button>
            </div>
         </fieldset>
      </form>
   </fieldset>
</BODY>
</HTML>
```

Quadro 27 – Código HTML e JavaScript gerado: Calculadora

APÊNDICE B – Atributos e métodos

Os quadros 28 e 29 apresentam, respectivamente, os atributos e métodos das classes modeladas nas figuras 13 e 14.

classe	atributos
Component	name : String fontFamily : String fontSize : int backGroundColor : String fontColor : String sizeWidth : int sizeHeight : int positionTop : int positionLeft : int layout : String father : String childrenComponents : Stack<Object> text : String type : String style : String functionJavaScript : Function funcjavascript : boolean
ComponentJCombobox	arrayItens : ArrayList<Item>
ComponentJMenu	menusItens : ArrayList<ComponentJMenuItem>
ComponentJMenuBar	menus : ArrayList<ComponentJMenu>
ComponentJPanel	haveComponentJLabel : boolean haveComponentJTextField : boolean haveComponentJButton : boolean countPanelsChilds : int
ComponentJRadioButtonGroup	labelJRadio : ComponentJLabel nameGroup : String
ComponentJTabbedPane	style : String tabsContent : ArrayList<ComponentJPanel> tabs : ArrayList<ComponentTab> contentPaneJTabbedPane : ComponentJPanel
ComponentJTextArea	rows : int cols : int
ComponentTab	onclickText : String
Item	item : String
ASTParser	RecClass recClass boolean isClass String className Properties dirTemplates String whatAssign String newClass String classType String identifier String idAnt ComponentJPanel component ComponentJPanel componentJPanel ComponentJLabel componentJLabel ComponentJTextField componentJTextField ComponentJButton componentJButton ComponentJCombobox componentJComboBox ComponentJRadioButtonGroup componentJRadioButtonGroup ComponentJCheckBox componentJCheckBox ComponentJPasswordField componentJPasswordField ComponentJTextArea componentJTextArea ComponentJLabel labelJRadioButton ComponentJTabbedPane componentJTabbedPane ComponentTab componentTab ComponentJMenuBar componentJMenuBar

classe	atributos
	ComponentJMenu componentJMenu ComponentJMenuItem componentJMenuItem ComponentJList componentJList int countTabs int countPanelsJTabbedPane String layout ContainerView container String methodName Item item String nameGroup String classFatherEvent ArrayList orderJPanels String alteraPanelPai Function func String template ArrayList<Identifier> identifiersJavaScript ArrayList<String> identifiersMethods ArrayList<String> identifiersComponents boolean declarationJavascript String vtypeBoolean String vtypeChar String vtypeByte String vtypeShort String vtypeInt String vtypeFloat String vtypeDouble String vtypeLong String iniMethod boolean assign boolean body String bodyMethod StringWriter writer boolean finishmethody boolean haveIf boolean haveElse Identifier id String typeObject
FrMain	jContentPane : JPanel jPanel : JPanel jPanel1 : JPanel jPanel2 : JPanel jPanel3 : JPanel jJMenuBar : JMenuBar jMenu : JMenu jMenu1 : JMenu jButton : JButton jButton1 : JButton jButton2 : JButton jMenuItem : JMenuItem jMenuItem1 : JMenuItem jMenuItem2 : JMenuItem jButton3 : JButton jTabbedPane : JTabbedPane jPanel4 : JPanel jScrollPane : JScrollPane jTextArea : JTextArea files : ArrayList<String> template : String
RecClass	className : String sizeWidth : int sizeHeight : int title : String container : ContainerView
Identifier	name : String value : String
Function	nameFunction : String bodyFunction : String

classe	atributos
	method : String

Quadro 28 – Atributos e métodos

classe	métodos
Component	Component() getName() setName(String) getSizeWidth() setSizeWidth(int) getSizeHeight() setSizeHeight(int) getPositionTop() setPositionTop(int) getPositionLeft() setPositionLeft(int) getChildrenComponents() setChildrenComponents(Stack<Object>) getLayout() setLayout(String) getFather() setFather(String) getFontFamily() setFontFamily(String) getFontSize() setFontSize(int) getBackGroundColor() setBackGroundColor(String) getFontColor() setFontColor(String) getText() setText(String) getType() setType(String) getStyle() setStyle(String) getFunctionJavaScript() setFunctionJavaScript(Function) isFuncjavascript() setFuncjavascript(boolean)
ComponentJButton	ComponentJButton()
ComponentJCombobox	ComponentJCombobox() getArrayItens() setArrayItens(ArrayList)
ComponentJLabel	ComponentJLabel()
ComponentJMenu	ComponentJMenu() getMenusItens() setMenusItens(ArrayList<ComponentJMenuItem>)
ComponentJMenuBar	ComponentJMenuBar() getMenus() setMenus(ArrayList<ComponentJMenu>)
ComponentJPanel	ComponentJPanel() setHaveComponentJLabel(boolean) setHaveComponentJTextField(boolean) setHaveComponentJButton(boolean) haveComponentJLabel() haveComponentJButton() haveComponentJTextField() getCountPanelsChilds() setCountPanelsChilds(int)
ComponentJRadioButtonGroup	getLabel() setLabelJRadio(ComponentJLabel) getNameGroup() setNameGroup(String)

classe	métodos
ComponentJTabbedPane	getStyle() setStyle(String) getTabsContent() setTabsContent(ArrayList<ComponentJPanel>) getTabs() setTabs(ArrayList<ComponentTab>) getContentPaneJTabbedPane() setContentPaneJTabbedPane(ComponentJPanel)
ComponentJTextArea	getRows() setRows(int) getCols() setCols(int)
ComponentJTextField	ComponentJTextField()
ComponentTab	getOnclickText() setOnclickText(String)
Item	getItem() setItem(String)
ASTParser	setClass() setClassName(Token) setAssignment(Token) setBodyFunctionComponent() setIdentifierName(Token) inicializeVariable() setMethodName(Token) setConstrutorName(Token) setTypeObject(Token) setAssignment2(Token) getClassName(Token) getObjectName(Token) verifyExistIdentifier(String) verifyExistIdentifierMethod(String) verifyExistIdentifierComponents(String) setIntegerElement(Token) positionMenus() setPositionComponentLabelRadioButton() setLongElement(Token) setFloatElement(Token) setCaracterElement(Token) setStringElement(Token) setBooleanElement() setNullElement() positionComponents() position() positionComponentJTabbedPane(ComponentJPanel) setNewClass() positionJPanels() positionTabsContent(ComponentJPanel) positionChildrensJPanel(ComponentJPanel) incElement(Token) decElement(Token) sumElement(Token) subElement(Token) multiElement(Token) divElement(Token) sumEqualsElement(Token) subEqualsElement(Token) multiEqualsElement(Token) divEqualsElement(Token) typeBoolean(Token) typeChar(Token) typeByte(Token) typeShort(Token) typeInt(Token) typeFloat(Token) typeDouble(Token) typeLong(Token) finishBodyMethod(Token)

classe	métodos
	bodyMethod(Token) hasBody() methodKeyListener() verifyNameComponent() methodActionListener() methodFocusListener() methodMouseListener() bodyFunctionEvents() positionJPanelsJTabbedPane() updateGroupJRadio() geraVelocity() verifyFunctions() getTemplate() setTemplate(String) getWriter() setWriter(StringWriter) getRecClass() getIf(Token) elseIf(Token) getEquals(Token) initIf(Token) finishIf(Token) assigmentValue(Token)
FrMain	getJPanel() getJPanel1() getJPanel2() getJPanel3() getJJMenuBar() getJMenu() getJMenu1() getJButton() openFileSwing() openFileTemplate() getJButton1() currentDir(int) getJButton2() generatedFile(StringWriter,String) getJMenuItem() getJMenuItem1() getJMenuItem2() getJButton3() getJTabbedPane() getJPanel4() getJTextArea() getJScrollPane() main(String[]) FrMain() initialize() getJContentPane()
RecClass	RecClass() getClassName() setClassName(String) getSizeWidth() setSizeWidth(int) getSizeHeight() setSizeHeight(int) getTitle() setTitle(String) getContainer() setContainer(ContainerView)
Identifier	getName() setName(String) getValue() setValue(String)
Function	getNameFunction() setNameFunction(String) getBodyFunction()

classe	métodos
	`setBodyFunction(String)` `getMethod()` `setMethod(String)`

Quadro 29 – Métodos

APÊNDICE C – *Template* com todos os componentes

No quadro 30 é apresentado o template para conversão de todos os componentes apresentados no quadro 13, exceto componentes de menu.

```
<HTML>
<HEAD>
<TITLE> $!{JFrameGetTitle} </TITLE>

</HEAD>
  <BODY>
       <fieldset
         name= $!{jcontentPanelGetName}
         style="position: absolute;
         font-family: $!{jcontentPanelGetFont};
         font-size: $!{jcontentPanelGetFontSize};
         color: $!{jcontentPanelGetFontColor};
         background-color: $!{jcontentPanelGetBackgroudColor};
         left: $!{jcontentPanelGetPositionLeft};
         top: $!{jcontentPanelGetPositionTop};
         width: $!{jcontentPanelGetWidth};
         height: $!{jcontentPanelGetHeight};">

         <STYLE TYPE="text/css" TITLE="">
         #foreach ($ComponentJPanel in $arrayJPanel)
             #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
               #foreach ($Component in $ComponentJPanel.getChildrenComponents())
                 #if ($!{Component.getType()} == "JTabbedPane")
                  $!{Component.getStyle()}
                 #end
               #end
             #end
          #end
          </STYLE>

         <form name= $!{JFrameGetName} >
         <script>
         var visible = 0;
         function chgTab(tabNumber) {
           if (tabNumber==visible) {
             return;
         }
         var contentToHide = document.getElementById("TabContent"+visible);
         var contentToShow = document.getElementById("TabContent"+tabNumber);
         var tabToHide2 = document.getElementById("Tab"+visible);
         var tabToShow2 = document.getElementById("Tab"+tabNumber);
         tabToShow2.className = "tab";
         tabToHide2.className = "otherTab";
         contentToHide.style.display = "none";
         contentToShow.style.display = "";
         visible = tabNumber;
         }
         </script>

       <TABLE
        #foreach ($ComponentJPanel in $arrayJPanel)
          #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
             #foreach ($Component in $ComponentJPanel.getChildrenComponents())
               #if ($!{Component.getType()} == "JTabbedPane")
                  name="$!{Component.getName()}"
                  style="position: absolute;
               #end
```

```
            #end
          #end
        #end
      ">

      <TR>
      <TD>
      <table>
      <tr>
       #foreach ($ComponentJPanel in $arrayJPanel)
         #set($cont = 0)
         #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
            #foreach ($Component in $ComponentJPanel.getChildrenComponents())
              #if ($!{Component.getType()} == "JTabbedPane")
                #foreach ($Component in $arrayTabs)
                   <td id="Tab$cont" onclick="chgTab($cont)"
                     #if ($cont == 0)
                       class="tab">
                     #else
                       class="otherTab">
                     #end
                     <font style="
                      font-family: dialog;
                      font-size: 12;
                      color: #000000;
                     ">$!{Component.getText()}</font></td>
                     #set($cont = $cont + 1)
                #end
               #end
            #end
         #end
       #end
      <tr>
      </table>
      </TD>
      </TR>
      <TR>
      <TD>

      #foreach ($ComponentJPanel in $arrayJPanel)
         #set($cont = 0)
         #if ($!{ComponentJPanel.getChildrenComponents().size()} > 0)
            #foreach ($Component in $ComponentJPanel.getChildrenComponents())
              #if ($!{Component.getType()} == "JTabbedPane")
                #foreach ($Component in $arrayTabsContent)

                  <div id="TabContent$cont" style="display: ;
                   border: 1px solid     #FFFFFF;
                   width: $!{Component.getSizeWidth()};
                   Height: $!{Component.getSizeHeight()};">
                   #set($cont = $cont + 1)

                  <fieldset name= $!{Component.getName()}
                   style="position:absolute;
                   font-family: $!{Component.getFontFamily()};
                   font-size: $!{Component.getFontSize()};
                   width:$!{Component.getSizeWidth()};
                   Height:$!{Component.getSizeHeight()};
                   border:0;left:$!{Component.getPositionLeft()};
                   top: $!{Component.getPositionTop()};">
                   #set($count = 0)

                   #foreach($ComponentJPanel in $Component.getChildrenComponents())
                       <fieldset name= $!{ComponentJPanel.getName()}
                        style="position:absolute;font-family:
                        $!{ComponentJPanel.getFontFamily()};
                        font-size: $!{ComponentJPanel.getFontSize()};
                         width:$!{ComponentJPanel.getSizeWidth()};
```

```
                        Height:$!{ComponentJPanel.getSizeHeight()};
                        border: 0;
                        left:$!{ComponentJPanel.getPositionLeft()};
                        top: $!{ComponentJPanel.getPositionTop()};
                        text-align= $!{ComponentJPanel.getLayout()};">

                         #foreach($Component in
$ComponentJPanel.getChildrenComponents())
                       #if ($!{Component.getType()} == "JLabel")
                         #set($count = $count + $!{Component.getPositionTop()})
                            #parse("templateJLabel.vm")
                       #end

                      #if ($!{Component.getType()} == "JTextField")
                           #parse("templateJTextField.vm")
                     #end

                     #if ($!{Component.getType()} == "JButton")
                           #parse("templateJButton.vm")
                     #end

                     #if ($!{Component.getType()} == "JRadioButton")
                          #parse("templateJRadioButton.vm")
                     #end

                     #if ($!{Component.getType()} == "JComboBox")
                          #parse("templateJComboBox.vm")
                     #end

                     #if ($!{Component.getType()} == "JCheckBox")
                           #parse("templateJCheckBox.vm")
                     #end

                     #if ($!{Component.getType()} == "JList")
                           #parse("templateJList.vm")
                     #end

                     #if ($!{Component.getType()} == "JTextArea")
                          #parse("templateJTextArea.vm")
                     #end
               #end
              #set($count = $count + 10)
             </fieldset>
              #end
               </fieldset>  </div>
            #end
                 #end
              #end
           #end
        #end

          </TD>
        </TR>
        </TABLE>
         </form>
        </fieldset>
   </BODY>
</HTML>
```

Quadro 30 – *Template* com todos os componentes

ANEXO A – Gramática do Java 5.0

No quadro 31 é apresentada a gramática da linguagem Java 5.0 utilizada no desenvolvimento da ferramenta.

```
CompilationUnit::= (PackageDeclaration)? (ImportDeclaration)* (TypeDeclaration)*
PackageDeclaration::= package Name ;
ImportDeclaration::= import (static)? Name (. *)? ;
Modifiers::= (public | static | protected | private | final | abstract | native |
    synchronized | transient | volatile | strictfp | annotation)*
TypeDeclaration::= ; | Modifiers (ClassOrInterfaceDeclaration | EnumDeclaration |
   AnnotationTypeDeclaration)
ClassOrInterfaceDeclaration::= (class | interface) <IDENTIFIER> (TypeParameters)?
   (ExtendsList)? (ImplementsList)? ClassOrInterfaceBody
ExtendsList::= extends ClassOrInterfaceType (, ClassOrInterfaceType)*
ImplementsList::= implements ClassOrInterfaceType (, ClassOrInterfaceType)*
EnumDeclaration::= enum <IDENTIFIER> (ImplementsList)? { (EnumConstant
   (, EnumConstant)*)? (, | (; (ClassOrInterfaceBodyDeclaration)*) )? }
EnumConstant::= <IDENTIFIER> (Arguments)? (ClassOrInterfaceBody)?
TypeParameters::= < TypeParameter (, TypeParameter)* >
TypeParameter::= <IDENTIFIER> (TypeBound)?
TypeBound::= extends ClassOrInterfaceType (& ClassOrInterfaceType)*
ClassOrInterfaceBody::= { (ClassOrInterfaceBodyDeclaration)* }
ClassOrInterfaceBodyDeclaration::= (Initializer | Modifiers
   (ClassOrInterfaceDeclaration | EnumDeclaration | AnnotationTypeDeclaration |
    ConstructorDeclaration | FieldDeclaration | MethodDeclaration) | ;)
FieldDeclaration::= Type VariableDeclarator (, VariableDeclarator)* ;
VariableDeclarator::= VariableDeclaratorId (= VariableInitializer)?
VariableDeclaratorId::= <IDENTIFIER> ([ ])*
VariableInitializer::= (ArrayInitializer | Expression)
ArrayInitializer::= { (VariableInitializer (, VariableInitializer)*)? (,)? }
MethodDeclaration::= (TypeParameters)? ResultType <IDENTIFIER> FormalParameters
   ([ ])* (throws NameList)? (Block | ;)
FormalParameters::= ( (FormalParameter (, FormalParameter)*)? )
FormalParameter::= Modifiers Type (...)? VariableDeclaratorId
ConstructorDeclaration::= (TypeParameters)? <IDENTIFIER> FormalParameters
   (throws NameList)? { (ExplicitConstructorInvocation)? Statements }
ExplicitConstructorInvocation::= (this Arguments ; | (PrimaryExpression .)? super
   Arguments ;
Statements::= (BlockStatement)*
Initializer::= (static)? Block
Type::= (ReferenceType | PrimitiveType)
ReferenceType::= (PrimitiveType ([ ])+ | ClassOrInterfaceType ([ ])* )
ClassOrInterfaceType::= <IDENTIFIER> (TypeArguments)? (. <IDENTIFIER>
   (TypeArguments)? )*
TypeArguments::= < TypeArgument (, TypeArgument)* >
TypeArgument::= (ReferenceType | Wildcard)
Wildcard::= ? (extends ReferenceType | super ReferenceType)?
PrimitiveType::= boolean | char | byte | short | int | long | float | double
ResultType::= void | Type
Name::= <IDENTIFIER> (. <IDENTIFIER>)*
NameList::= Name (, Name)*
Expression::= ConditionalExpression (AssignmentOperator Expression)?
AssignmentOperator::= = | *= | /= | %= | += | -= | <<= | >>= | >>>= | &= | ^= | |=
ConditionalExpression::= ConditionalOrExpression (? Expression : Expression)?
ConditionalOrExpression::= ConditionalAndExpression (|| ConditionalAndExpression)*
ConditionalAndExpression::= InclusiveOrExpression (&& InclusiveOrExpression)*
InclusiveOrExpression::= ExclusiveOrExpression (| ExclusiveOrExpression)*
ExclusiveOrExpression::= AndExpression (^ AndExpression)*
AndExpression::= EqualityExpression (& EqualityExpression)*
EqualityExpression::= InstanceOfExpression ((== | !=) InstanceOfExpression)*
InstanceOfExpression::= RelationalExpression (instanceof Type)?
RelationalExpression::= ShiftExpression ( (< | > | <= | >=) ShiftExpression)*
ShiftExpression::= AdditiveExpression ( (<< | RSIGNEDSHIFT | RUNSIGNEDSHIFT)
```

```
   AdditiveExpression)*
AdditiveExpression::= MultiplicativeExpression ( (+ | -) MultiplicativeExpression)*
MultiplicativeExpression::= UnaryExpression ( (* | / | %) UnaryExpression)*
UnaryExpression::= ( (+ | -) UnaryExpression | PreIncrementExpression |
   PreDecrementExpression | UnaryExpressionNotPlusMinus)
PreIncrementExpression::= ++ PrimaryExpression
PreDecrementExpression::= -- PrimaryExpression
UnaryExpressionNotPlusMinus::= ( (~ | !) UnaryExpression | CastExpression |
   PostfixExpression )
CastLookahead::= ( PrimitiveType | ( Type [ ] | ( Type ) (~ | ! | ( | <IDENTIFIER>
   | this | super | new | Literal)
PostfixExpression::= PrimaryExpression (++ | --)?
CastExpression::= ( Type ) UnaryExpression | ( Type ) UnaryExpressionNotPlusMinus
PrimaryExpression::= PrimaryPrefix (PrimarySuffix)*
PrimaryPrefix::= Literal | this | super . <IDENTIFIER> (Arguments)? | (
   Expression ) | AllocationExpression | ResultType . class | <IDENTIFIER>
   (Arguments)?
PrimarySuffix::= . (this | super | AllocationExpression | (TypeArguments)?
   <IDENTIFIER> (Arguments)? | [ Expression ]
Literal::= <INTEGER_LITERAL> | <LONG_LITERAL> | <FLOATING_POINT_LITERAL> |
   <CHARACTER_LITERAL> | <STRING_LITERAL> | BooleanLiteral | NullLiteral
BooleanLiteral::= true | false
NullLiteral::= null
Arguments::= ( (ArgumentList)? )
ArgumentList::= Expression (, Expression)*
AllocationExpression::= (new PrimitiveType ArrayDimsAndInits | new
   ClassOrInterfaceType (TypeArguments)? (ArrayDimsAndInits | Arguments
   (ClassOrInterfaceBody)? ) )
ArrayDimsAndInits::= ( [ Expression ])+ ([ ])* | ([ ])+ ArrayInitializer
Statement::= LabeledStatement | AssertStatement | Block | EmptyStatement |
    StatementExpression | SwitchStatement | IfStatement | WhileStatement |
    DoStatement | ForStatement | BreakStatement | ContinueStatement |
    ReturnStatement | ThrowStatement | SynchronizedStatement | TryStatement
AssertStatement::= assert Expression (: Expression)? ;
LabeledStatement::= <IDENTIFIER> : Statement
Block::= { Statements }
BlockStatement::= Modifiers ClassOrInterfaceDeclaration |
   VariableDeclarationExpression ; | Statement
VariableDeclarationExpression::= Modifiers Type VariableDeclarator
   (, VariableDeclarator )*
EmptyStatement::= ;
StatementExpression::= (PreIncrementExpression | PreDecrementExpression |
   PrimaryExpression (++ | -- | AssignmentOperator Expression)? ) ;
SwitchStatement::= switch ( Expression ) { (SwitchEntry)* }
SwitchEntry::= (case Expression | default) : Statements
IfStatement::= if ( Expression ) Statement (else Statement)?
WhileStatement::= while ( Expression ) Statement
DoStatement::= do Statement while ( Expression ) ;
ForStatement::= for ( (VariableDeclarationExpression : Expression | (ForInit)? ;
   (Expression)? ; (ForUpdate)? ) ) Statement
ForInit::= VariableDeclarationExpression | ExpressionList
ExpressionList::= Expression (, Expression)*
ForUpdate::= ExpressionList
BreakStatement::= break (<IDENTIFIER>)? ;
ContinueStatement::= continue (<IDENTIFIER>)? ;
ReturnStatement::= return (Expression)? ;
ThrowStatement::= throw Expression ;
SynchronizedStatement::= synchronized ( Expression ) Block
TryStatement::= try Block (catch ( FormalParameter ) Block)* (finally Block)?
RUNSIGNEDSHIFT::= > > >
RSIGNEDSHIFT::= > >
Annotation::= NormalAnnotation | SingleMemberAnnotation | MarkerAnnotation
NormalAnnotation::= @ Name ( (MemberValuePairs)? )
MarkerAnnotation::= @ Name
SingleMemberAnnotation::= @ Name ( MemberValue )
MemberValuePairs::= MemberValuePair (, MemberValuePair)*
MemberValuePair::= <IDENTIFIER> = MemberValue
MemberValue::= Annotation | MemberValueArrayInitializer | ConditionalExpression
```

```
MemberValueArrayInitializer::= { MemberValue (, MemberValue)* (,)? }
AnnotationTypeDeclaration::= @ interface <IDENTIFIER> AnnotationTypeBody
AnnotationTypeBody::= { (AnnotationBodyDeclaration)* }
AnnotationBodyDeclaration::= Modifiers ( (AnnotationTypeMemberDeclaration |
   ClassOrInterfaceDeclaration | EnumDeclaration | AnnotationTypeDeclaration |
   FieldDeclaration) | ; )
AnnotationTypeMemberDeclaration::= Type <IDENTIFIER> ( ) (DefaultValue)? ;
DefaultValue::= default MemberValue
```

Fonte: adaptado de Gesser (2007, p. 77).

Quadro 31 – Gramática do Java 5.0