UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA CONVERSÃO DE INTERFACES GRÁFICAS DESENVOLVIDAS EM DELPHI PARA A BIBLIOTECA GTK+

JOSIMAR ZIMERMANN

BLUMENAU
2011

2011/2-12

JOSIMAR ZIMERMANN

FERRAMENTA PARA CONVERSÃO DE INTERFACES GRÁFICAS DESENVOLVIDAS EM DELPHI PARA A BIBLIOTECA GTK+

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 Ciência da Computação — Bacharelado.

Profª. Joyce Martins - Orientadora

BLUMENAU
2011

2011/2-12

FERRAMENTA PARA CONVERSÃO DE INTERFACES GRÁFICAS DESENVOLVIDAS EM DELPHI PARA A BIBLIOTECA GTK+

Por

JOSIMAR ZIMERMANN

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

Presidente:	Profª. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Aurélio Faustino Hope, Mestre – FURB
Membro:	Prof. José Roque Voltolini da Silva – FURB

Blumenau, 14 de dezembro de 2011.

RESUMO

O presente trabalho descreve a especificação e implementação de uma ferramenta para conversão de interfaces gráficas construídas em Delphi para um formato legível à biblioteca gráfica GTK+. Utiliza analisadores léxico, sintático e semântico para a interpretação da entrada da ferramenta. O conteúdo lido do arquivo de entrada é carregado para a memória e convertido para o formato XML legível à `Libglade`, que define interfaces gráficas para a biblioteca GTK+.

Palavras-chave: Delphi. GTK+. GUI. Glade.

ABSTRACT

The current work describes the specification and implementation of a tool for conversion of graphical users interfaces built in Delphi to a readable format for the GTK+ library. Uses lexical, syntactic and semantic analyzers for interpretation the input of the tool. The content readed from the input file is loaded into memory and converted to XML format readable for the `Libglade`, wich defines user interface for the GTK+.

Key-words: Delphi. GTK+. GUI. Glade.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

BNF – *Backus Naur Form*

GALS – Gerador de Analisadores Léxicos e Sintáticos

GDK – *Gimp Drawing Kit*

GTK+ - *Gimp Toolkit*

GUI – *Graphical User Interface*

RF – Requisito Funcional

RNF – Requisito Não Funcional

UML – *Unified Modeling Language*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Em face do crescimento no uso do Linux, os desenvolvedores de software notaram a necessidade de adequar seu produto com o objetivo de atender os usuários de diferentes sistemas operacionais. Esta adequação pode ser realizada através do uso de linguagens de programação e bibliotecas portáveis. Contudo, ainda existe uma grande quantidade de software desenvolvido em ambientes de programação que geram código executável específico para um sistema operacional.

Em especial, o ambiente de desenvolvimento integrado Delphi é amplamente utilizado na construção e manutenção de software (ANTUNES, 2008). Esta ferramenta é conhecida por oferecer ao construtor do software um ambiente para fácil e rápida criação de interfaces gráficas com o usuário. Porém, as interfaces gráficas desenhadas com o auxílio do Delphi são interpretadas apenas pelo sistema operacional Microsoft Windows.

Por isso, muitas vezes opta-se pela migração de aplicativos escritos em Delphi para uma linguagem de programação portável como o Java. A etapa de migração inclui redesenhar as interfaces gráficas confeccionadas com o auxílio do Delphi, para uma equivalente na linguagem de programação para a qual se está migrando. Trata-se de um processo que exige um grande esforço e demanda tempo considerável no processo da migração. Ainda assim, o que se obtém é um software migrado para uma nova linguagem de programação e a interface gráfica do aplicativo continuará inerente à linguagem adotada.

Tendo em vista os fatores supracitados, propôs-se o desenvolvimento de uma ferramenta para auxiliar na tradução de interfaces gráficas criadas no Delphi para o formato da `Libglade` (THE GLADE PROJECT, 2009), legível à biblioteca gráfica GTK+[1] (GALE et al., 2010). Além da portabilidade oferecida pela GTK+, a interface gráfica especificada no formato da `Libglade` poderá ser utilizada por qualquer linguagem de programação que implementa a mesma.

[1] GIMP *Toolkit* (GTK+).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para auxiliar na conversão de interfaces gráficas desenvolvidas no ambiente de desenvolvimento integrado Delphi, para um formato legível à biblioteca gráfica GTK+.

Os objetivos específicos do trabalho são:

a) identificar os componentes visuais mais comumente utilizados na construção de interfaces gráficas em Delphi e verificar os componentes similares oferecidos pela biblioteca GTK+;
b) extrair informações dos componentes visuais que compõem uma interface gráfica desenvolvida em Delphi, através do uso de técnicas de análise léxica e sintática;
c) gerar um arquivo `Libglade` para criar a definição de uma interface gráfica para a GTK+ semelhante à desenvolvida no Delphi;
d) traduzir a assinatura dos eventos dos componentes gráficos do Delphi para os componentes gráficos da biblioteca GTK+;
e) exibir a interface gráfica convertida para a biblioteca GTK+.

1.2 ESTRUTURA DO TRABALHO

O trabalho está estruturado em quatro capítulos. O segundo capítulo apresenta o embasamento teórico para desenvolvimento do trabalho. Esse capítulo faz uma breve abordagem da notação XML e sua utilização na criação de padrões e comportamentos, fala sobre o desenvolvimento de interface gráfica do usuário com enfoque na ferramenta Delphi e finaliza apresentando a biblioteca gráfica GTK+ e os recursos que a compõem, além dos trabalhos correlatos à ferramenta desenvolvida.

O terceiro capítulo apresenta os detalhes envolvidos no desenvolvimento do trabalho. Inicialmente é relatado o estudo realizado nas ferramentas Delphi e Glade e o resultado obtido. Segue com a especificação dos requisitos, casos de uso e diagramas de classes que constituíram a base para desenvolvimento da ferramenta DelphiToGTK+. O capítulo finaliza apresentando detalhes técnicos relevantes da concepção do software.

Por fim, o quarto capítulo apresenta as conclusões alcançadas com o desenvolvimento

do trabalho, incluindo problemas e limitações, bem como sugestões para extensão da ferramenta e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados aspectos teóricos, conceitos, técnicas e ferramentas que formam a base para desenvolvimento do trabalho proposto. Primeiramente é abordada brevemente a *eXtensible Markup Language* (XML), linguagem de marcação utilizada no desenvolvimento do trabalho e presente nos recursos explorados. Na sequência fala-se sobre geradores de interfaces gráficas do usuário, seguido pelo tópico sobre *Graphical User Interface* (GUI). Depois, é apresentado um exemplo de uma interface gráfica do usuário construída com auxílio do Delphi, bem como a estrutura do arquivo que contém a definição da interface construída, o `.dfm`, entrada da ferramenta desenvolvida. O presente capítulo também fala sobre o desenvolvimento de aplicativos portáveis, mais especificamente sobre a criação de interfaces gráficas legíveis a diferentes linguagens de programação, aprofundando no tema GTK+, biblioteca para desenvolvimento de interfaces gráficas portáveis. A ferramenta Glade, utilizada no desenvolvimento da interface do trabalho proposto, também é mencionada, bem como a biblioteca `Libglade`, usada para processamento das interfaces criadas no Glade. Por fim, são apresentados os trabalhos correlatos similares ao trabalho proposto.

2.1 DEFINIÇÃO DE PADRÕES E COMPORTAMENTO COM XML

XML é um padrão para documentos de marcação que define uma sintaxe genérica usada para marcação de dados com *tags*, de forma simples e legível. É um formato flexível e pode ser customizado para atender diferentes domínios como *sites* da web, dados eletrônicos para integração, vetores gráficos, genealogia, serialização de objetos, chamadas a procedimentos remotos, sistemas de correio por voz, entre outros (HAROLD; MEANS, 2002, p. 3).

Uma das vantagens na utilização de documentos XML é a sua portabilidade entre as diferentes linguagens de programação e sistemas operacionais. Segundo Harold e Means (2002, p. 3), há uma grande variedade de bibliotecas livres para diferentes linguagens de programação que são capazes de ler e escrever documentos XML. Existem ferramentas desenvolvidas exclusivamente para leitura e escrita de documentos XML, enquanto outras são

customizadas para processar um documento XML com um formato específico a um domínio de aplicação. Contudo, em todos os casos a mesma sintaxe é usada, evitando que o formato fique restrito a um domínio, impedindo-o de ser processado por outras ferramentas (HAROLD; MEANS, 2002, p. 3).

A ferramenta Glade, usada para criação de interfaces gráficas de aplicações GTK+, utiliza o padrão XML para definição das interfaces gráficas. A adoção desta técnica possibilita a criação de um mecanismo de carregamento de interfaces gráficas de forma dinâmica na aplicação, fornecendo a possibilidade de modificações na interface definida no documento XML sem a necessidade de reconstrução do código fonte da aplicação.

2.2 GERADORES DE INTERFACES GRÁFICAS COM O USUÁRIO

Segundo Herrington (2003, p. 3), geradores de código são programas criados para escrever outros programas. Geradores de interfaces gráficas são igualmente geradores de código. Contudo, o resultado não é necessariamente um programa ou aplicação, mas sim uma interface gráfica para o usuário final do produto, o software.

Um gerador de interface gráfica do usuário recebe sua entrada de um arquivo de definição de interface gráfica e de uma definição de *Application Programming Interface* (API). Alguns programas especialistas na criação de interfaces gráficas utilizam um arquivo com um formato distinto para construir interfaces gráficas, ao passo que outros utilizam formatos consagrados como o XML. A adoção desta abordagem no processo de desenvolvimento de software possibilita a criação de uma camada de abstração entre as regras de negócio para a interface e sua implementação (HERRINGTON, 2003, p. 100).

Para Herrington (2003, p. 101-102), alguns benefícios chaves na utilização de geradores de interfaces gráficas são:

a) consistência: geração em massa de formulários criam formulários consistentes e que apresentam padrões de usabilidade;
b) flexibilidade: usando ferramentas para desenhar interfaces, pode-se rapidamente efetuar modificações requeridas para esta interface. Modificações nas regras de segurança podem destacar esta flexibilidade, pois ao modificar e melhorar a API de segurança da aplicação pode-se alterar os modelos para recriar as interfaces que estarão em conformidade com as novas regras de segurança;

c) portabilidade: tão boa quanto as ferramentas hoje disponíveis para criação de interfaces gráficas, esta técnica permite vincular a lógica de negócio da interface aos detalhes da implementação. Proporcionando um alto nível de representação dos requisitos de negócio da interface gráfica na aplicação, permite reproduzir código executável da aplicação para diferentes tecnologias de interface gráfica.

2.3 GUI

Uma GUI "é um tipo de interface do utilizador que permite a interação com dispositivos digitais através de elementos gráficos como ícones e outros indicadores visuais [...]" (INTERFACE..., 2010). Segundo Preece, Rogers e Sharp (2005, p. 19), a necessidade do desenvolvimento de interfaces para interação com dispositivos digitais, em especial os computadores, passou a surgir "no final dos anos 70 e início dos [anos] 80 [com o] advento dos monitores e estações de trabalho pessoais".

A construção de interfaces gráficas é uma etapa comumente presente no processo de desenvolvimento de software aplicativo. Através do uso de componentes gráficos, os desenvolvedores de software procuram criar interfaces que tornam o uso do seu produto uma tarefa simples e intuitiva. Um exemplo claro é apresentado na Figura 1, onde a GUI de um software reprodutor de mídia tenta imitar o painel de funções de um aparelho de som.

Fonte: NullSoft (2010).

Figura 1 - Exemplo de software reprodutor de mídia com interface gráfica do usuário

Para auxiliar os desenvolvedores de software na confecção das interfaces dos seus produtos, têm-se as bibliotecas para criação de interfaces gráficas com o usuário. Estas

bibliotecas costumam fornecer aos seus utilizadores um conjunto de componentes gráficos e funções para criação de interfaces gráficas.

Um exemplo bastante conhecido é a *Visual Component Library* (VCL). Criada para ser disponibilizada junto ao Delphi, esta biblioteca oferece ao desenvolvedor uma camada de abstração para as funções gráficas do sistema operacional Microsoft Windows (VISUAL..., 2010). Outros exemplos são as bibliotecas Qt[2] e GTK+, ambas multi-plataforma. São utilizadas no desenvolvimento dos ambientes de *desktop* KDE[3] e GNOME[4], respectivamente.

2.4 DESENVOLVIMENTO DE INTERFACES GRÁFICAS NO DELPHI

O Delphi é uma ferramenta *Rapid Application Development* (RAD) construída sobre a linguagem de programação Object Pascal e largamente utilizada para o desenvolvimento de aplicações com interface gráfica de usuário na plataforma Microsoft Windows (EMBARCADERO..., 2010).

Esta ferramenta oferece um ambiente prático e de fácil utilização para a criação de interfaces gráficas. O desenho das interfaces gráficas no Delphi é baseado em formulários, ou janelas, nos quais o desenvolvedor utiliza o recurso *drag-and-drop* (arrastar-e-soltar) para adicionar os componentes que irão compor a interface. É possível ainda definir as propriedades e comportamentos destes componentes.

O Delphi utiliza a VCL, que fornece uma grande quantidade de componentes visuais para auxiliar na criação de interfaces gráficas. Esta biblioteca foi escrita em Object Pascal e é utilizada para desenvolvimento de aplicações para o sistema operacional Microsoft Windows. Alguns dos componentes gráficos que integram a VCL são: botões, caixas de seleção, caixa de checagem, listas, janelas, painéis, barras de ferramenta, barra de *status*, barras de progresso, barras de rolagem, entre outros.

As definições de um formulário ou janela desenvolvido no Delphi ficam armazenadas em um arquivo com a extensão `.dfm`. Este arquivo possui uma formatação distinta para identificar os componentes que compõem a interface, armazenando informações sobre o seu posicionamento no formulário, seu tamanho em *pixels*, comportamento e funções associadas a

[2] *Framework* multi-plataforma para desenvolvimento de aplicativos com interface gráfica (QT, 2010).
[3] Ambiente gráfico *desktop* e plataforma de desenvolvimento para o sistema operacional Linux (KDE, 2010).
[4] GNU *Network Object Model Environment* (GNOME, 2010).

seus eventos. A Figura 2 exibe uma interface gráfica simples produzida no Delphi e o Quadro 1 apresenta o arquivo `.dfm` correspondente à interface.

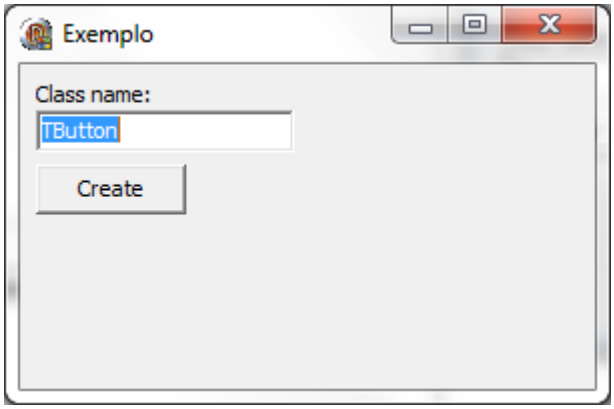

Figura 2 - Uma interface gráfica simples produzida no ambiente Delphi

```
object Form1: TForm1
  Left = 0
  Top = 0
  Caption = 'Exemplo'
  ClientHeight = 208
  ClientWidth = 318
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 8
    Top = 8
    Width = 58
    Height = 13
    Caption = 'Class name:'
  end
  object Edit1: TEdit
    Left = 8
    Top = 23
    Width = 121
    Height = 21
    TabOrder = 0
    Text = 'TButton'
  end
  object Button1: TButton
    Left = 8
    Top = 50
    Width = 75
    Height = 25
    Caption = 'Create'
    TabOrder = 1
  end
end
```

Quadro 1 - Conteúdo do arquivo `.dfm` correspondente à interface apresentada

2.5 GTK+

A GTK+ foi originalmente desenvolvida para auxiliar na criação de um novo aplicativo para editoração de imagens, chamado GNU *Image Manipulator Program* (GIMP). O *toolkit* foi criado por Peter Mattis, Spencer Kimball e Josh MacDonald no ano de 1997, na Universidade da Califórnia (KRAUSE, 2007, p. 2). Harlow (1999, p. 3) afirma que a GTK+ é uma biblioteca para desenvolvimento de aplicações com interface gráfica do usuário,

extensivamente utilizada na confecção de aplicações Linux com GUI, com destaque para os ambientes de *desktop* GNOME e Xfce[5]. Apesar da sua origem no sistema operacional Linux, a biblioteca tem sido expandida para suportar outros sistemas operacionais, tais como: Microsoft Windows, BeOS, Solaris, Mac OS X e outros (KRAUSE, 2007, p. 2).

A GTK+ é um sistema baseado em sinais e funções de *call-back*. Um sinal é uma notificação para a aplicação na qual o usuário efetuou alguma ação. É possível executar uma função para responder a este sinal através da GTK+. Estas funções são conhecidas como funções de *call-back* (KRAUSE, 2007, p. 27).

Uma aplicação GTK+ é iniciada por uma chamada à função `gtk_main()`. A partir deste ponto o programa entra no laço principal e fica em estado de espera até receber um sinal, como o clique do *mouse* sobre um botão, por exemplo. Os sinais são emitidos pelo sistema gerenciador de janelas do sistema operacional, responsável pelo controle dos componentes gráficos da aplicação, e capturados pela GDK, que por usa vez repassa para a GTK+. A GTK+ procura uma função de *call-back* associada ao sinal ou evento e efetua a sua chamada (LIRA, 2010).

A GTK+ é constituída por bibliotecas, cada uma fornecendo ao desenvolvedor uma classe de funcionalidades específicas. As seis bibliotecas que a compõem são apresentadas nos tópicos subsequentes.

2.5.1 `GLib`

A biblioteca `GLib` é uma coleção de funções que são usadas em larga escala pela GTK+. Listas encadeadas, árvores, manipuladores de exceções, gerenciadores de memória e temporizadores são parte dos recursos oferecidos por esta biblioteca. A GTK+ requer a `GLib` e baseia-se nela para a implementação de portabilidade e funcionalidade. A `GLib` pode ser usada sem a GTK+ para desenvolvimento de aplicações que não necessitam de interação com o usuário através de uma interface gráfica (HARLOW, 1999, p. 7).

Um dos principais benefícios oferecidos pela utilização da `GLib` é que esta biblioteca provê uma interface multi-plataforma que possibilita o desenvolvimento de aplicativos portáveis entre os diferentes sistemas operacionais suportados pela GTK+, com pequenas ou nenhuma alteração no código fonte (KRAUSE, 2007, p. 5).

[5] Ambiente gráfico *desktop* para sistemas baseados em UNIX e similares (XFCE, 2010).

2.5.2 GLib Object System ou GObject

GObject foi originalmente distribuída como parte da biblioteca GTK+ 1 na forma de uma classe denominada GtkObject. A partir da versão 2.0 da GTK+, passou a ser distribuída como uma biblioteca, junto à GLib. A GObject constitui a base para a hierarquia de componentes gráficos da GTK+, bem como para muitos objetos implementados nas suas bibliotecas de apoio (KRAUSE, 2007, p. 6).

Esta biblioteca foi desenvolvida para permitir o acesso fácil aos objetos da GTK+, escritos em C, a partir de outras linguagens de programação. Tal habilidade facilitou o desenvolvimento da larga variedade de *bindings* disponíveis para outras linguagens de programação, embora a própria biblioteca seja escrita em C (KRAUSE, 2007, p. 6). Contudo, para Krause (2007, p. 6), este aspecto constitui uma dificuldade, visto que cada linguagem de programação fornece uma abordagem diferente para tipos de dados.

2.5.3 GIMP Drawing Kit (GDK)

Esta biblioteca gráfica, originalmente desenvolvida para o X Windows System (KRAUSE, 2007, p. 7), constitui uma camada de abstração entre a GTK+ e o sistema gerenciador de janelas ou interfaces gráficas do sistema operacional, efetuando chamadas às funções de manipulação de janelas do próprio sistema operacional (HARLOW, 1999, p. 209).

A GDK é responsável por apresentar desenhos, gráficos, cursores e fontes de todas as aplicações GTK+. Também fornece suporte para o recurso *drag-and-drop* (arrastar-e-soltar) e manipulação de eventos de janelas (KRAUSE, 2007, p. 7).

A habilidade presente nos componentes gráficos da GTK+ de desenhar-se na tela é fornecida por esta biblioteca. Para tanto, cada componente possui uma associação com o objeto GdkWindow, com exceção de alguns poucos componentes. O objeto GdkWindow é essencialmente uma área retangular localizada na tela na qual o componente será desenhado. Também permite aos componentes detectar os eventos disparados pelo sistema gerenciador de janelas do sistema operacional (KRAUSE, 2007, p. 7).

2.5.4 GdkPixbuf

A GdkPixbuf é uma pequena biblioteca que fornece funções para manipulação de imagens do lado cliente (KRAUSE, 2007, p. 7), oferecendo uma grande variedade de funções para manipulação e exibição de imagens (GALE et al., 2010). Foi criada como substituta da biblioteca Imlib. Utilizando funções disponibilizadas por esta biblioteca, é possível carregar imagens a partir de arquivos ou de um conjunto de dados de imagem (KRAUSE, 2007, p. 7).

A GdkPixbuf guarda referência para as imagens por ela carregadas. A utilização desta técnica permite que uma mesma imagem possa ser exibida em várias localizações, enquanto apenas uma instância da imagem foi carregada para a memória. A imagem somente será destruída quando não houver mais referência a ela (KRAUSE, 2007, p. 7).

2.5.5 Pango

Ao passo que a GDK manipula a apresentação de imagens e janelas, esta biblioteca controla a saída de texto e fonte internacionalizados, tarefa esta realizada em conjunto com a biblioteca Cairo ou Xft, dependendo da versão da GTK+ (KRAUSE, 2007, p. 8).

Todo texto dentro da biblioteca Pango é representado na codificação UTF-8. Esta codificação é utilizada devido à sua compatibilidade com softwares de 8-bits, muito comuns nas plataformas UNIX (KRAUSE, 2007, p. 8).

2.5.6 Accessibility Toolkit (ATK)

Para Krause (2007, p. 9), ao desenhar uma aplicação é importante levar em consideração deficiências das quais alguns de seus usuários podem ser portadores. Portanto, a ATK fornece aos componentes gráficos da GTK+ construções integradas para criação de interfaces gráficas para usuários com problemas de acessibilidade. Algumas funções que a ATK oferece são: leitor de tela, temas de alto contraste visual, modificador de comportamento de teclado, entre outros (KRAUSE, 2007, p. 9).

2.6 GLADE

Embora o *toolkit* GTK+ ofereça ao desenvolvedor de software um conjunto completo de funcionalidades para criação de interfaces gráficas, realizar esta tarefa manualmente – escrever as instruções para criação da interface gráfica com todos os seus componentes, incluindo seu comportamento e disposição na tela – é um processo moroso e desgastante, e muitas vezes o resultado é longe do esperado. Ademais, caso seja necessário alterar qualquer aspecto da interface – mover, alterar a dimensão ou comportamento de um componente; adicionar um novo componente; remover um componente – o novo código precisará ser recompilado.

O Glade surgiu da necessidade da criação de uma ferramenta para facilitar a criação de interfaces gráficas para a biblioteca GTK+. É uma ferramenta da espécie RAD, usada no desenvolvimento rápido e fácil de interfaces gráficas para o *toolkit* GTK+ e para o ambiente *desktop* GNOME (THE GLADE PROJECT, 2009). As interfaces gráficas desenhadas com o Glade são salvas em arquivos no formato XML que descrevem a estrutura dos componentes que integram a interface, suas propriedades, comportamento e eventos associados. Esta abordagem permite ao desenvolvedor manter as definições da interface gráfica separadamente do código fonte da aplicação. Qualquer alteração na interface pode ser efetuada sem a necessidade de alterar o código fonte da aplicação (KRAUSE, 2007, p. 359).

Os arquivos gerados pelo Glade recebem a extensão `.glade`. Estes arquivos podem ser carregados dinamicamente para aplicações GTK+ utilizando uma biblioteca capaz de efetuar a leitura do conteúdo do arquivo e transformá-lo numa interface gráfica GTK+. Isto permite que uma mesma interface gráfica possa ser carregada para aplicações GTK+ nas diferentes linguagens de programação que possuem *binding* para a GTK+ (THE GLADE PROJECT, 2010). Um exemplo de arquivo `.glade` é apresentado no Quadro 2, ao passo que a Figura 3 exibe a interface gráfica correspondente.

```
<?xml version="1.0"?>
<glade-interface>
  <!-- interface-requires gtk+ 2.16 -->
  <!-- interface-naming-policy project-wide -->
  <widget class="GtkWindow" id="window1">
    <property name="visible">True</property>
    <property name="title" translatable="yes">Exemplo</property>
    <property name="window_position">center</property>
    <signal name="delete_event" handler="window1_delete_event_cb"/>
    <child>
      <widget class="GtkFixed" id="fixed1">
        <property name="visible">True</property>
        <child>
          <widget class="GtkLabel" id="label1">
            <property name="width_request">63</property>
            <property name="height_request">20</property>
            <property name="visible">True</property>
            <property name="label" translatable="yes">Class name:</property>
          </widget>
          <packing>
            <property name="x">6</property>
            <property name="y">6</property>
          </packing>
        </child>
        <child>
          <widget class="GtkEntry" id="entry1">
            <property name="width_request">125</property>
            <property name="height_request">23</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="text" translatable="yes">TButton</property>
          </widget>
          <packing>
            <property name="x">6</property>
            <property name="y">23</property>
          </packing>
        </child>
        <child>
          <widget class="GtkButton" id="button1">
            <property name="label" translatable="yes">Create</property>
            <property name="width_request">80</property>
            <property name="height_request">27</property>
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="receives_default">True</property>
          </widget>
          <packing>
            <property name="x">7</property>
            <property name="y">52</property>
          </packing>
        </child>
      </widget>
    </child>
  </widget>
</glade-interface>
```

Quadro 2 - Arquivo gerado pelo Glade com as definições da interface gráfica

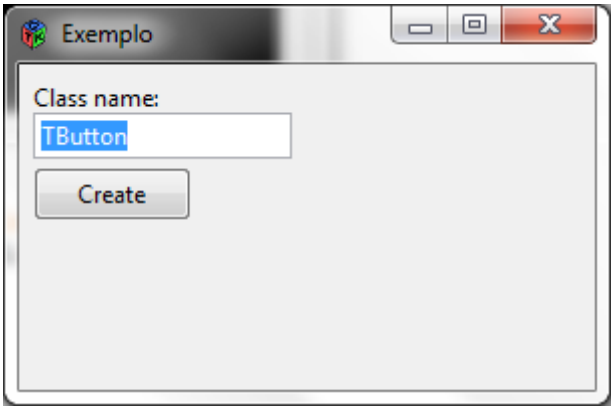

Figura 3 - Interface gráfica simples para GTK+ produzida pelas definições apresentadas no Quadro 2

O processamento das interfaces gráficas desenhadas no Glade realiza-se através da utilização de uma biblioteca de leitura de definição de interface. A primeira biblioteca desenvolvida para esta finalidade denomina-se *Graphical Interface Description Loader* API (`Libglade`), distribuída como parte componente da biblioteca GTK+.

A `Libglade` fornece uma classe denominada `GladeXML`, usada para criar e manter a uma interface gráfica do usuário carregada a partir de um arquivo XML. Também é disponibilizada uma função que permitir associar os sinais dos componentes gráficos definidos no Glade às funções de *call-back* codificadas na aplicação (KRAUSE, 2007, p. 372).

Uma vez que a interface gráfica é carregada para a aplicação, a `Libglade` permite acessar cada componente gráfico presente na interface através do seu identificador único, definido durante a criação da interface (KRAUSE, 2007, p. 375). Desta forma, é possível modificar aspectos inerentes a um componente gráfico em tempo de execução. Por exemplo, em uma aplicação cujo acesso às funcionalidades é controlado por regras de permissão, pode-se usar a `Libglade` para habilitar ou desabilitar os componentes gráficos que fornecem acesso à funcionalidade em questão.

Os eventos associados aos componentes gráficos da interface podem ser vinculados às funções de *call-back* da aplicação através da utilização da função `glade_xml_signal_autoconnect`. Esta função procura na aplicação uma função cujo nome é similar ao nome definido para o manipulador do sinal do componente gráfico (KRAUSE, 2007, p. 375).

2.7 TRABALHOS CORRELATOS

Durante a elaboração deste documento foram identificados alguns trabalhos correlatos que possuem características semelhantes ao trabalho proposto. Foram selecionados três projetos: DelphiToWeb (SOUZA, 2005), a extensão da ferramenta Delphi2Java-II desenvolvida por Silveira (2006) e ScriptCase (NETMAKE, 2010), os quais são apresentados mais detalhadamente nas próximas seções.

2.7.1 DelphiToWeb

Desenvolvida no ano de 2005, a ferramenta DelphiToWeb converte formulários desenhados no Delphi para páginas *HyperText Markup Language* (HTML) (SOUZA, 2005). A ferramenta consiste numa primeira etapa de interpretação do arquivo `.dfm` com as suas definições do formulário Delphi e depois na sua conversão para uma interface em outros formato, neste caso páginas HTML.

A ferramenta converte uma grande quantidade de componentes gráficos Delphi, sendo que foram considerados os componentes mais utilizados e com equivalentes em HTML. Os componentes convertidos pela ferramenta são: `TForm`, `TMainMenu`, `TMenuItem`, `TToolBar`, `TToolButton`, `TLabel`, `TEdit`, `TButton`, `TSpeedButton`, `TBitBtn`, `TComboBox`, `TMemo`, `TRichEdit`, `TListBox`, `TGroupBox`, `TRadioGroup`, `TRadioButton`, `TPanel`, `TCheckBox`, `TPopupMenu`, `TPageControl` e `TTabSheet` (SOUZA, 2005, p. 35-36).

Para o desenvolvimento foram utilizados analisadores léxico, sintático e semântico para efetuar a interpretação dos arquivos `.dfm` que contém as definições da interface gráfica desenhada no Delphi (SOUZA, 2005, p. 37).

2.7.2 Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados

Silveira (2006, p. 13) diz que a ferramenta Delphi2Java-II é utilizada para migração de aplicações desenvolvidas em Delphi para a plataforma Java, efetuando a leitura de formulários Delphi e gerando classes Java, preservando a estrutura originalmente desenhada.

No entanto, uma limitação de Delphi2Java-II é a quantidade restrita de componentes que são convertidos. Sendo assim, Silveira (2006, p. 13) propôs a melhoria da ferramenta por permitir também a conversão de componentes visuais para acesso a banco de dados, além da conversão da forma de conexão ao banco utilizando a *Java Data Base Connectivity* (JDBC).

A extensão contempla a conversão dos seguintes componentes de visualização de dados: `TDBCheckBox`, `TDBComboBox`, `TDBEdit`, `TDBGrid`, `TDBMemo`, `TDBRadioGroup`, `TDBText` (SILVEIRA, 2006, p. 41-42). Além dos componentes de visualização de dados, responsáveis por apresentar os dados do banco de dados ao usuário, os seguintes componentes de acesso a dados também são convertidos pela ferramenta: `TDatabase`, `TDataSource`, `TTable` e `TQuery` (SILVEIRA, 2006, p. 43).

O trabalho foi desenvolvido sobre o padrão de projeto *Model-View-Controller* (MVC), a fim de dividir as camadas de saída geradas pela ferramenta, entre componentes de visualização, componentes de visualização de dados e componentes de acesso a banco de dados. A análise da entrada, arquivo `.dfm`, foi efetuada pela utilização de analisadores léxico, sintático e semântico.

2.7.3 ScriptCase

O ScriptCase é um gerador de aplicações web baseadas em banco de dados *Structured Query Language* (SQL). As aplicações geradas pela ferramenta utilizam as linguagens de programação PHP e JavaScript, as mesmas linguagens utilizadas no desenvolvimento do software (NETMAKE, 2010). As aplicações resultantes do ScriptCase rodam sobre qualquer servidor capaz de interpretar o PHP. Portanto, segundo NetMake (2010), os aplicativos gerados pelo ScriptCase podem ser executados em uma grande quantidade de sistemas operacionais, tais como: Linux, Windows, IBM I5 e outros sistemas UNIX.

A construção de aplicações com o ScriptCase é efetuada através de um navegador com acesso ao servidor no qual a ferramenta está instalada. A criação é baseada em formulários web que efetuarão a manipulação dos dados no banco de dados. A ferramenta fornece recursos como (SCRIPTCASE, 2010): validação de campos (data, moeda, número, entre outros); validação de chaves (primárias, únicas, estrangeiras); possibilidade de geração de relatórios baseados em consultas SQL definidas pelo usuário; execução de consultas complexas com `subselects`, `joins` e `stored procedures`; suporte a vários bancos de dados conhecidos (Oracle, MS-SQL Server, DB2, MySQL, Informix, PostGreSQL, Access,

Interbase e Firebird) e criação de aplicações que se interligam com o Delphi e o Visual Basic, através da instanciação do *Component Object Model* (COM).

O software é desenvolvido pela empresa brasileira NetMake e atualmente encontra-se na sua versão 5.2, sendo utilizado por diversos clientes no território nacional (NETMAKE, 2010).

2.7.4 Funcionalidades dos trabalhos correlatos

O apresenta Quadro 3 recursos e funcionalidades presentes nos trabalhos correlatos, estabelecendo uma compração entre as ferramentas apresentadas. Características não aplicáveis à ferramenta são representadas pelo símbolo hífen.

característica	**DelphiToWeb**	**Extensão da ferramenta Delphi2Java-II**	**ScriptCase**
linguagem de programação	Java	Java	PHP e Javascript
flexibilidade para conversão	Não	Não	-
tradução para tecnologia portável	Sim	Sim	-
suporte para acesso a banco de dados	Não	Sim	Sim
interface traduzida pode ser migrada para diferentes linguagens de programação	Não	Não	-

Quadro 3 - Características dos trabalhos correlatos

Sobre as características elencadas no Quadro 3, têm-se:

a) linguagem de programação: relaciona a linguagem de programação usada no desenvolvimento da ferramenta;
b) flexibilidade para conversão: indica se a ferramenta oferece mecanismo ou recurso que permite alterar o comportamento da mesma no processo de conversão;
c) tradução para tecnologia portável: indica se a ferramenta é capaz de converter a entrada para uma linguagem de programação ou tecnologia portável entre diferentes sistemas operacionais;
d) suporte para acesso a banco de dados: indica a capacidade da ferramenta traduzir ou fornecer suporte a recursos para acesso a sistemas de banco de dados;
e) interface traduzida pode ser migrada para diferentes linguagens de programação: indica se o resultado obtido com utilização da ferramenta permite compatilizar a

interface gráfica de origem para diferentes linguagens de programação.

3 DESENVOLVIMENTO DA FERRAMENTA

Este capítulo descreve de forma detalhada as etapas envolvidas no desenvolvimento da ferramenta DelphiToGTK+. Inicialmente são apresentados os principais requisitos que devem ser atendidos pela ferramenta. Em seguida, tem-se as equivalências e similaridades entre os componentes visuais da VCL, biblioteca de componentes gráficos do Delphi e os componentes gráficos disponibilizados pela `Libglade`, biblioteca para especificação de interfaces gráficas para aplicações GTK+. Para além das equivalências e similaridades, são apresentadas as estratégias adotadas para tradução dos componentes da VCL que não possuem equivalência na `Libglade`.

As equivalências identificadas na etapa supracitada foram mapeadas utilizando padrões XML. Tal mapeamento inclui, além dos próprios componentes, suas propriedades e eventos associados, também denominados sinais na GTK+. O resultado deste processo são arquivos XML, denominados mapas neste domínio de aplicação.

Uma vez identificadas as equivalências entre as bibliotecas envolvidas, faz-se necessária a leitura e a interpretação das interfaces gráficas desenvolvidas no Delphi. Tal processo é realizado com auxílio da ferramenta GALS (GESSER, 2003), capaz de efetuar a validação de uma cadeia de caracteres segundo uma *Backus-Naur Form* (BNF) pré-definida. Para o domínio de aplicação do trabalho, foi especificada uma BNF para leitura de arquivos segundo o formato `.dfm`.

Através da aplicação dos conceitos da orientação a objetos, foram especificados os diagramas de casos de uso e classes, a fim de descrever os métodos e o processamento executados pela ferramenta para leitura do arquivo `.dfm` e sua transformação para a GTK+ segundo as definições dos mapas de tradução em formato XML.

A interface gráfica da ferramenta propriamente dita foi desenhada utilizando o Glade, através do qual é possível criar interfaces gráficas para aplicações GTK+ utilizando o formato descrito pela biblioteca `Libglade`.

Ademais, o presente capítulo também apresenta os resultados obtidos com o desenvolvimento da ferramenta.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Na sequência são apresentados os requisitos, atendidos pela ferramenta. No Quadro 4 são apresentados os Requisitos Funcionais (RF) e sua rastreabilidade em relação aos casos de uso. O Quadro 5 lista os Requisitos Não Funcionais (RNF) da ferramenta.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: Permitir ao usuário selecionar o diretório de localização dos mapas de tradução.	UC01
RF02: Permitir ao usuário selecionar o diretório de entrada onde se localizam os arquivos que deseja converter.	UC02
RF03: Permitir ao usuário selecionar o diretório de saída dos arquivos convertidos.	UC03
RF04: Realizar a conversão dos arquivos `.dfm` selecionados para arquivos `.glade` correspondentes.	UC04
RF05: Permitir ao usuário solicitar a visualização da interface gráfica convertida para GTK+.	UC05

Quadro 4 - Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Ser codificada em Pascal/Object Pascal para o compilador Free Pascal.
RNF02: Ser implementada no ambiente de desenvolvimento Lazarus 0.9.28.2.
RNF03: Ter a interface gráfica desenhada para a biblioteca GTK+ utilizando o Glade.
RNF04: Realizar a leitura e a interpretação do arquivo de entrada (`.dfm`) utilizando técnicas de compiladores, como análises léxica, sintática e semântica.
RNF05: Gerar código Pascal/Object Pascal para carregar as interfaces traduzidas para a GTK+.

Quadro 5 - Requisitos não funcionais

3.2 COMPONENTES CONVERTIDOS

Esta seção apresenta os componentes gráficos mais comumente utilizados e disponibilizados no Delphi que foram traduzidos para a biblioteca GTK+ utilizando os recursos disponibilizados pela biblioteca `Libglade`.

Durante o processo de estudo e análise dos componentes, foram identificadas três diferentes categorias de tradução: simples, condicionada e especial. Cada uma das categorias citadas é detalhada nos tópicos subsequentes.

3.2.1 Componentes de tradução simples

São categorizados como componentes de tradução simples os componentes gráficos do Delphi que possuem um similar na biblioteca `Libglade`. Foram identificados vinte componentes nesta condição: `TForm`, `TLabel`, `TEdit`, `TMemo`, `TButton`, `TCheckBox`, `TRadioButton`, `TComboBox`, `TImage`, `TScrollBox`, `TStaticText`, `TPageControl`, `TRichEdit`, `TProgressBar`, `TMonthCalendar`, `TStatusBar`, `TToolBar`, `TCoolBar`, `TSpinEdit` e `TCalendar`. O Quadro 6 apresenta uma breve descrição dos componentes mencionados e o nome do componente similar disponibilizado pela biblioteca `Libglade`.

componente Delphi	descrição	componente `Libglade`
`TForm`	Janela ou formulário onde são inseridos outros componentes.	`GtkWindow`
`TLabel`	Etiqueta de texto.	`GtkLabel`
`TEdit`	Caixa de texto simples com uma linha única.	`GtkEntry`
`TMemo`	Caixa de texto com múltiplas linhas.	`GtkTextView`
`TButton`	Botão simples.	`GtkButton`
`TCheckBox`	Caixa de seleção para marcar ou desmarcar opções.	`GtkCheckButton`
`TRadioButton`	Botão do tipo rádio para selecionar apenas uma entre várias opções.	`GtkRadioButton`
`TComboBox`	Caixa de texto com opções para seleção.	`GtkComboBox`
`TImage`	Componente usado para exibir uma imagem.	`GtkImage`
`TScrollBox`	Caixa de composição provida de barras de rolagem vertical e horizontal.	`GtkScrolledWindow`
`TStaticText`	Componente semelhante ao `TLabel`.	`GtkLabel`
`TPageControl`	Controle de abas, sendo que cada aba é uma composição com seus próprios componentes.	`GtkNotebook`
`TRichEdit`	Componente semelhante ao `TMemo`.	`GtkTextView`
`TProgressBar`	Barra de progresso.	`GtkProgressBar`
`TMonthCalendar`	Calendário que permite selecionar uma data.	`GtkCalendar`
`TStatusBar`	Barra de *status*.	`GtkStatusBar`
`TToolBar`	Barra de ferramentas para armazenar um conjunto de botões.	`GtkToolBar`
`TCoolBar`	Barra de ferramentas com subdivisões que podem ser movidas ou redimensionadas.	`GtkHandleBox`
`TSpinEdit`	Caixa para entrada de valores numéricos inteiros, acompanhado por dois botões que permitem incrementar e decrementar o valor.	`GtkSpinButton`
`TCalendar`	Calendário que não permite alternar mês ou ano.	`GtkCalendar`

Quadro 6 - Equivalência dos componentes de tradução simples

3.2.2 Componentes de tradução condicionada

São categorizados como componentes condicionados aqueles cuja tradução é condicionada ao valor de uma ou mais propriedades nele presentes. O componente `TScrollBar` (barra de rolagem), por exemplo, pode ter sua propriedade `orientation` configurada como `vertical` ou `horizontal`, definindo a orientação do componente. Contudo, a `Libglade` disponibiliza dois componentes distintos para as duas orientações citadas, sendo um deles para orientação vertical e outro para orientação horizontal.

No Quadro 7 estão relacionados os três componentes desta categoria, incluindo uma breve descrição sobre sua finalidade, as propriedades condicionantes e as possíveis traduções para a `Libglade` de acordo com o valor da propriedade.

componente Delphi	descrição	propriedades condicionantes	componente `Libglade`
`TMenuItem`	Entrada de menu que pode ser vinculada à barra de menu da aplicação ou à um menu de contexto.	`Bitmap.Data`	`GtkImageMenuItem`
		`AutoCheck`	`GtkCheckMenuItem`
		`RadioItem`	
		`AutoCheck`	`GtkRadioMenuItem`
		`RadioItem`	
		`Caption`	`GtkSeparatorMenuItem`
			`GtkMenuItem`
`TScrollBar`	Barra para rolagem de tela.	`Kind`	`GtkHScrollbar`
		`Kind`	`GtkVScrollbar`
`TTrackBar`	Componente usado para selecionar uma posição dentro de uma escala de valores.	`Orientation`	`GtkHScale`
		`Orientation`	`GtkVScale`

Quadro 7 - Equivalência dos componentes de tradução condicionada às propriedades

3.2.3 Componentes de tradução especial

Uma terceira categoria de componentes foi identificada como de tradução especial. São componentes do Delphi cuja tradução para a biblioteca `Libglade` não pode ser padronizada devido às suas especificidades. Alguns destes componentes, ao traduzi-los para a `Libglade`, resultam em dois ou mais componentes para simular a aparência original. Um exemplo é o componente `TGroupBox`: uma caixa retangular decorada com uma borda e uma etiqueta de texto, normalmente localizada na parte superior esquerda. A sua tradução para a `Libglade` consiste na junção de três componentes: `GtkFrame`, `GtkAlignment` e `GtkLabel`.

Outro exemplo é o componente `TPanel`, comumente usado para auxiliar na disposição

e alinhamento de outros componentes inseridos na interface gráfica. A tradução deste para a `Libglade` é condicionada pela quantidade dos componentes nele armazenados e seu alinhamento. As possíveis traduções são: `GtkVBox`, `GtkHBox` e `GtkFixed`.

Todos os componentes não categorizados como simples ou condicionados são categorizados como especiais. A sua tradução não apresenta padrões legíveis e, portanto, recebem tratamento especial pela ferramenta. A ferramenta DelphiToGTK+ implementa a tradução de dois componentes da categoria especial, sendo eles o `TPanel` e o `TGroupBox`, cujo traduções são realizadas pelas classes `TPanelTranslator` e `TGroupBoxTranslator`, respectivamente.

3.3 ESPECIFICAÇÃO DA SAÍDA

A ferramenta DelphiToGTK+ processa arquivos de definição de interface gráfica do usuário construídas no Delphi e o resultado são arquivos de definição de interface segundo as especificações da biblioteca `Libglade`. Tais arquivos comumente recebem a extensão `.glade`, visto que podem ser construídos com auxílio da ferramenta Glade. Contudo, o seu conteúdo é formatado segundo os padrões da XML.

Cada componente gráfico inserido na interface do usuário recebe uma etiqueta XML denominada `widget`. Nesta etiqueta também estão presentes os atributos que informam o identificador do componente dentro da interface e o nome da classe correspondente a este componente. Cada `widget` presente no arquivo `.glade` pode conter propriedades que determinam aspectos visuais e comportamentais do elemento gráfico. Tais propriedades são definidas pela etiqueta `property`, que contém um nome definido no atributo `name`. O valor da propriedade é delimitado pelo início e fim da etiqueta.

Durante o processo de confecção de uma interface gráfica `Libglade` também é possível informar quais dos sinais emitidos pelos elementos gráficos em decorrência de ações aplicadas pelo usuário serão capturados e processados pela aplicação GTK+. Os sinais da GTK+ são similares aos eventos da biblioteca VCL. Para cada componente disponível na `Libglade` há um conjunto de sinais que podem ser interceptados pela aplicação através da implementação de funções de *call-back*. A estas funções são enviadas informações pertinentes ao sinal disparado que podem ser interpretadas pela aplicação para definir o fluxo de

comandos desejado. No escopo do arquivo `.glade` os sinais são identificados pela etiqueta `signal`, sendo o seu nome definido no atributo `name`. O nome da função *call-back* responsável pela manipulação do sinal é informada no atributo `handler`.

Dentre os componentes gráficos disponíveis na GTK+, há aqueles que são capazes de armazenar dentro da sua área outros componentes. Os componentes com esta capacidade são descendentes da classe `GtkContainer`. Cada elemento contido em um descendente da `GtkContainer` estará delimitado pela etiqueta `child` que poderá conter uma etiqueta `packing`, onde define-se o comportamento de redimensionamento do elemento em relação ao *container* no qual está contido.

O Quadro 8 apresenta o conteúdo de um arquivo `.glade` que contém todos os elementos mencionados nesta seção.

```
<?xml version="1.0"?>
<glade-interface>
  <!-- interface-requires gtk+ 2.16 -->
  <!-- interface-naming-policy project-wide -->
  <widget class="GtkWindow" id="window1">
    <property name="visible">True</property>
    <property name="title" translatable="yes">Exemplo</property>
    <property name="window_position">center</property>
    <signal name="delete_event" handler="window1_delete_event_cb"/>
    <child>
      <widget class="GtkVBox" id="vbox1">
        <property name="visible">True</property>
        <property name="orientation">vertical</property>
        <child>
          <widget class="GtkLabel" id="label1">
            <property name="visible">True</property>
            <property name="label" translatable="yes">label</property>
          </widget>
          <packing>
            <property name="expand">False</property>
            <property name="position">0</property>
          </packing>
        </child>
        <child>
          <widget class="GtkEntry" id="entry1">
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">&#x25CF;</property>
          </widget>
          <packing>
            <property name="expand">False</property>
            <property name="position">1</property>
          </packing>
        </child>
        <child>
          <placeholder/>
        </child>
      </widget>
    </child>
  </widget>
</glade-interface>
```

Quadro 8 - Especificação de uma interface gráfica GTK+

A interface gráfica, gerada como resultado da especificação do Quadro 8, é apresentada na Figura 4.

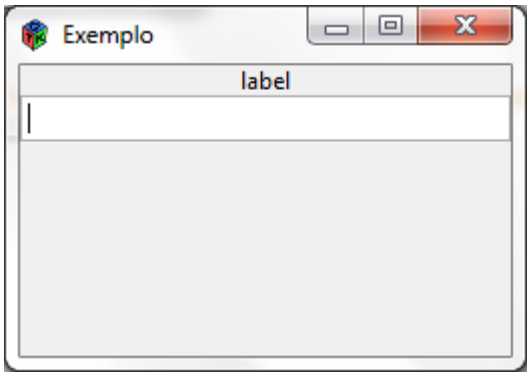

Figura 4 - Interface gráfica gerada como resultado das especificações do Quadro 8

3.4 ANÁLISE DA ENTRADA DA FERRAMENTA

Como mencionado em tópicos anteriores, a entrada da ferramenta DelphiToGTK+ são arquivos de extensão `.dfm` gerados pelo Delphi e que possui as definições de interface gráfica construída sobre a biblioteca VCL. Para análise do conteúdo do arquivo `.dfm` utilizou-se a solução proposta por Silveira (2006) na extensão da ferramenta Delphi2Java-II. Esta solução consiste na elaboração de uma BNF (Anexo A) capaz de interpretar o conteúdo do arquivo `.dfm`. Com base na BNF elaborada, utilizou-se o GALS (GESSER, 2003) para gerar os analisadores léxico, sintático e semântico.

Os analisadores mencionados, especificados segundo o paradigma de programação orientado a objetos, foram adaptados para atender as necessidades impostas pelo presente trabalho, bem como para adaptar-se aos padrões de codificação da linguagem Object Pascal, sobre a qual a ferramenta DelphiToGTK+ foi desenvolvida. A Figura 5 apresenta o diagrama de classes dos analisadores léxico, sintático e semântico, bem como seus relacionamentos, além de outras classes pertinentes, todas geradas pelo GALS. A classe `TSemantic` teve sua especificação substancialmente alterada para atender as necessidades da ferramenta.

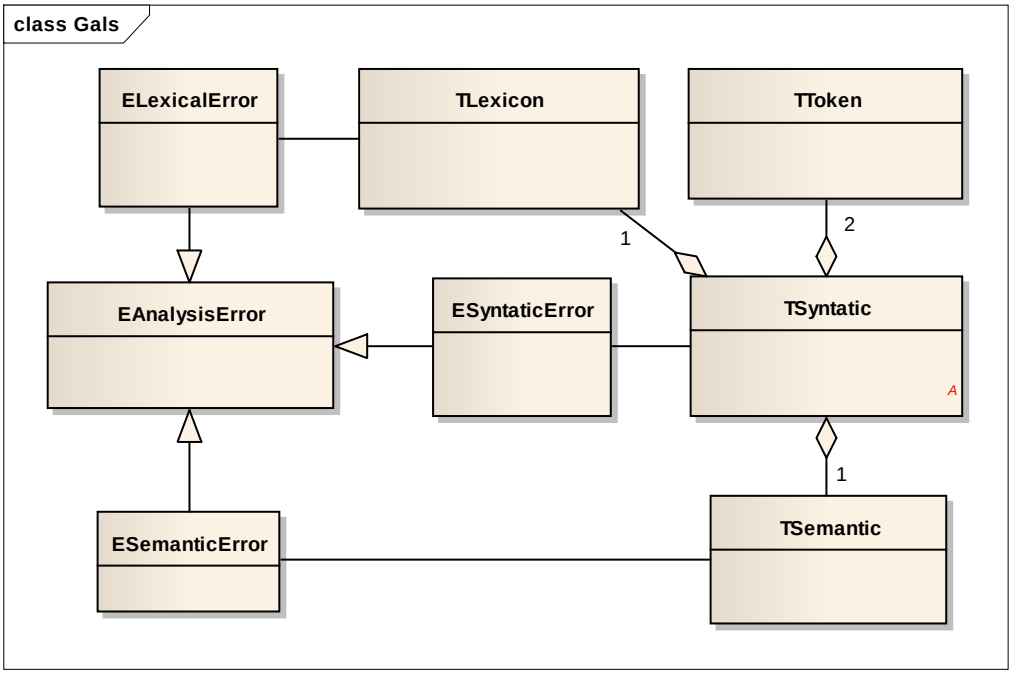

Figura 5 - Diagrama de classes dos analisadores léxico, sintático e semântico

3.5 ESPECIFICAÇÃO DA FERRAMENTA

A especificação da ferramenta DelphiToGTK+ foi elaborada utilizando o software Enterprise Architect, utilizando a técnica de orientação a objetos a fim de modelar a solução para o presente problema. O resultado da especificação são os diagramas de casos de uso, classes e sequência, apresentados nos tópicos subsequentes.

3.5.1 Casos de uso

A ferramenta DelphiToGTK+ foi modelada para atender as necessidades especificadas nos cinco casos de uso apresentados na Figura 6.

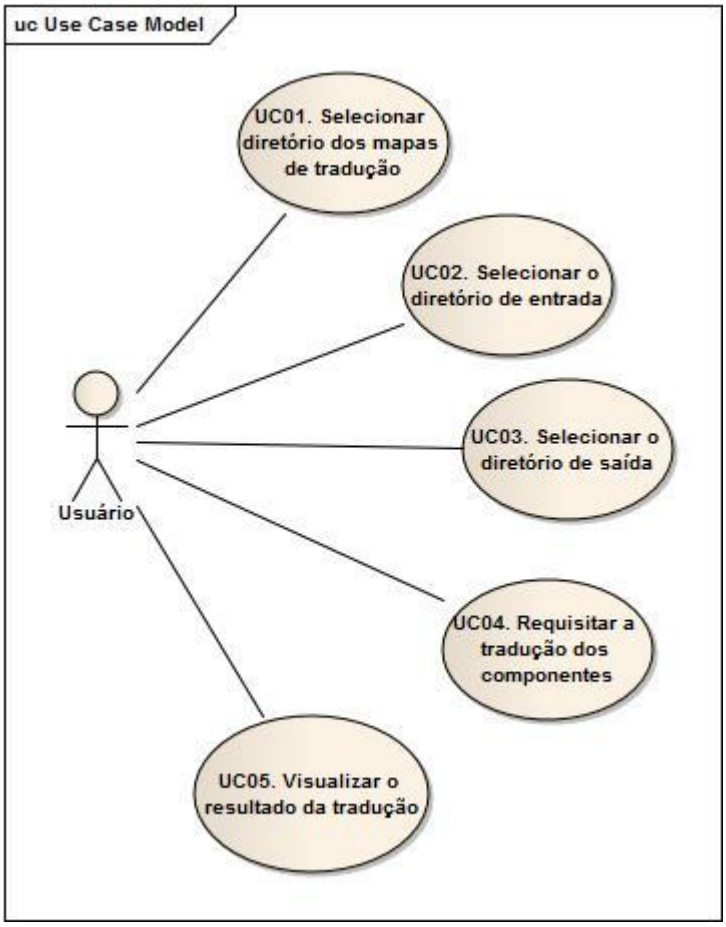

Figura 6 - Diagrama de casos de uso

O caso de uso identificado como UC04 somente pode ser executado após os casos de uso UC01, UC02 e UC03. O caso de uso denominado UC05, por sua vez, somente poderá ser acionado após a execução do caso de uso UC04. O único ator presente no diagrama de casos de uso, o `Usuário`, trata-se de um programador com conhecimentos no desenvolvimento de aplicações *desktop* com interface gráfica do usuário, com foco na criação de software em Delphi.

No caso de uso UC01, o `Usuário` seleciona o diretório onde estão localizados os mapas de tradução, arquivos com formato XML que especificam como os componentes gráficos do Delphi e suas propriedades serão traduzidos para a GTK+ segundo as definições da `Libglade`. No diretório selecionado, procurar-se-á por três arquivos de mapa:

a) `components.xml`: arquivo com o mapeamento da tradução dos componentes;
b) `properties.xml`: arquivo com o mapeamento da tradução das propriedades visuais e comportamentais dos componentes;
c) `events.xml`: arquivo com o mapeamento dos eventos dos componentes.

Para que o caso de uso UC01 (Quadro 9) possa ser realizado com sucesso é imprescindível a existência dos três arquivos supracitados no diretório selecionado pelo ator `Usuário`. Na ausência de quaisquer destes arquivos, uma mensagem de erro notificará o usuário, orientando-o a selecionar um diretório que contenha os arquivos de mapeamento necessários. Também se faz necessário que os arquivos para mapeamento dos componentes e suas demais propriedades estejam devidamente formatados de forma que a ferramenta seja capaz de interpretá-los. A não conformidade no formato de quaisquer dos arquivos de mapeamento resultará em uma mensagem de erro ao usuário da ferramenta, impedindo-o de prosseguir o processo de tradução.

UC01 – Selecionar diretório dos mapas de tradução: permite informar o diretório onde estão os arquivos XML com as definições de mapeamento dos componentes, suas propriedades (visuais e comportamentais) e eventos.	
Requisitos atendidos	RF01.
Pré-condições	Não possui.
Cenário principal	1) O usuário informa diretório onde estão localizados os mapas de tradução. 2) A aplicação verifica a existência dos mapas no diretório informado. 3) Os mapas são interpretados e carregados para a aplicação.
Exceção 01	Mapa(s) não encontrado(s): No passo 2, caso ao menos um dos mapas necessários não seja localizado, o usuário é notificado e o processo interrompido.
Exceção 02	Formato inválido: No passo 3, se os mapas selecionados não possuem formato legível à ferramenta, o processo não será finalizado, o usuário será notificado sobre o formato inadequado do(s) mapa(s) e o processo interrompido.
Pós-condições	Mapas carregados.

Quadro 9 - Caso de uso UC01

O segundo caso do uso da ferramenta, denominado UC02, diz respeito à possibilidade do usuário selecionar um diretório que contém os arquivos de extensão `.dfm` para os quais deseja-se realizar a tradução para a biblioteca GTK+. Uma vez selecionado, a aplicação efetua a leitura do conteúdo do diretório a fim de localizar arquivos passíveis de tradução, ou seja, arquivos de definição de interface gráfica produzidos com auxílio do Delphi. Caso nenhum arquivo para tradução seja encontrado no diretório informado, a aplicação notificará o usuário, orientando-o a selecionar um diretório válido para tradução. O Quadro 10 apresenta em detalhes o cenário do caso de uso UC02.

UC02 – Selecionar o diretório de entrada: permite ao usuário selecionar o diretório onde estão armazenados os arquivos de extensão `.dfm` que deseja traduzir.	
Requisitos atendidos	RF02.
Pré-condições	Não possui.
Cenário principal	1) O usuário informa o diretório de localização dos arquivos para tradução. 2) A aplicação verifica se o diretório informado possui arquivos passíveis de tradução.
Exceção 01	Não há arquivo(s) para tradução: No passo 2, se nenhum arquivo para tradução for encontrado, o usuário será notificado e o processo interrompido.
Pós-condições	Diretório de entrada definido.

Quadro 10 - Caso de uso UC02

A possibilidade de seleção do diretório de saída dos arquivos traduzidos é comtemplada pelo caso de uso UC03, cujo cenário detalhado é exibido no quadro 11. Neste cenário, o ator informa o diretório onde deseja que os arquivos traduzidos sejam armazenados. Os arquivos processados recebem o mesmo nome do arquivo de entrada correspondente, diferindo apenas na sua extensão, uma vez que os arquivos de definição de interface gráfica da biblioteca `Libglade` são identificados pela extensão `.glade`.

UC03 – Selecionar o diretório de saída: permite ao usuário informar à aplicação o diretório onde deseja que os arquivos traduzidos sejam armazenados.	
Requisitos atendidos	RF03.
Pré-condições	Não possui.
Cenário principal	1) O usuário seleciona o diretório de saída dos arquivos traduzidos. 2) A aplicação verifica a existência do diretório selecionado.
Exceção 01	Diretório inexistente: No passo 2, caso o diretório informado não seja encontrado, o usuário será notificado e o processo interrompido.
Pós-condições	Diretório de saída definido.

Quadro 11 - Caso de uso UC03

Após a execução bem sucedida dos casos de uso UC01, UC02 e UC03, o ator `Usuário` estará apto a realizar o caso de uso UC04 (Quadro 12), que consiste no processo de tradução propriamente dito. Neste cenário, o usuário solicita à ferramenta DelphiToGTK+ a realização da tradução dos arquivos `.dfm` encontrados no diretório de entrada, informado no caso de uso UC02. Para cada arquivo encontrado no diretório, aplicar-se-á técnicas de análise léxica, sintática e semântica, de forma a interpretar o seu conteúdo, identificando os componentes gráficos e suas propriedades. O resultado desta interpretação é um conjunto de dados carregados para a memória em uma estrutura de dados modelada para o domínio do problema. A tradução efetuar-se-á com auxílio dos mapas carregados no caso de uso UC01. Os arquivos traduzidos serão armazenados no diretório de saída, informado no caso de uso UC03, recebendo o mesmo nome do arquivo `.dfm` de origem, contudo com a extensão `.glade`.

UC04 – Requisitar a tradução dos componentes: usuário aciona o processo de tradução dos arquivos informados.	
Requisitos atendidos	RF04, RNF04.
Pré-condições	Mapas carregados. Diretório de entrada definido. Diretório de saída definido.
Cenário principal	1) O usuário aciona o processo de tradução. 2) A aplicação realiza a análise léxica do arquivo de entrada. 3) A aplicação realiza a análise sintática do arquivo de entrada. 4) A aplicação realizada a análise semântica do arquivo de entrada. 5) O arquivo interpretado e processado é carregado para a memória. 6) A aplicação realiza a tradução dos componentes especificados no arquivo de entrada. 7) O arquivo traduzido é armazenado no diretório de saída.
Exceção 01	Erro léxico: No passo 2, o arquivo processado não possui um formato `.dfm` legível para tradução, portanto, o processo é interrompido.
Exceção 02	Erro sintático: No passo 3, o arquivo processado não possui sintaxe válida, portanto, a tradução é interrompida.
Exceção 03	Erro na tradução do componente: No passo 6, caso ocorra erro no processo de tradução do componente e/ou suas propriedades, o processo é interrompido. O erro pode ocorrer por conta de incompatibilidade entre o tradutor configurado no mapa e o componente.
Pós-condições	Interface(s) gráfica(s) traduzida(s).

Quadro 12 - Caso de uso UC04

Depois de realizada a tradução dos arquivos de definição de interface gráfica, o usuário poderá visualizar o resultado da tradução, conforme especificado pelo cenário do caso de uso UC05 (Quadro 13). A aplicação utiliza um modelo (Apêndice A) de código escrito na linguagem Pascal e adapta-o de forma a capacitá-lo a carregar as definições de uma interface gráfica GTK+ segundo as definições da biblioteca `Libglade`, para a qual os arquivos `.dfm` foram traduzidos. O código é compilado pela ferramenta utilizando a versão 2.4.2 do compilador Free Pascal, disponibilizado junto a ferramenta DelphiToGTK+, assim como as bibliotecas *runtime* da GTK+, referenciadas pelo modelo de código.

UC05 – Visualizar o resultado da tradução: permite ao usuário visualizar o formulário Delphi traduzido para uma interface gráfica similar na GTK+.	
Requisitos atendidos	RF05, RNF03, RNF05.
Pré-condições	Interface gráfica traduzida.
Cenário principal	1) O usuário solicita a visualização da interface gráfica traduzida. 2) A aplicação adapta um modelo de código Pascal para carregar a interface gráfica traduzida. 3) A aplicação compila o código Pascal. 4) A aplicação executa o resultado da compilação.
Exceção 01	Erro na compilação: No passo 3, pode ocorrer erro na compilação do código e o processo é abortado.
Pós-condições	Não possui.

Quadro 13 - Caso de uso UC05

3.5.2 Diagramas de classes

O diagrama de classe fornece uma visualização clara dos relacionamentos entre as classes da ferramenta bem como suas responsabilidades. Para facilitar a compreensão, as classes foram classificadas segundo as suas responsabilidades em pacotes. Um pacote engloba um conjunto de classes que se comunicam através da troca de mensagens com o intuito de alcançar um objetivo em comum. Alguns pacotes são independentes, ao passo que outros fazem referência a outros pacotes e necessitam deles para executar suas ações. A Figura 7 apresenta a estrutura de pacotes da ferramenta DelphiToGTK+.

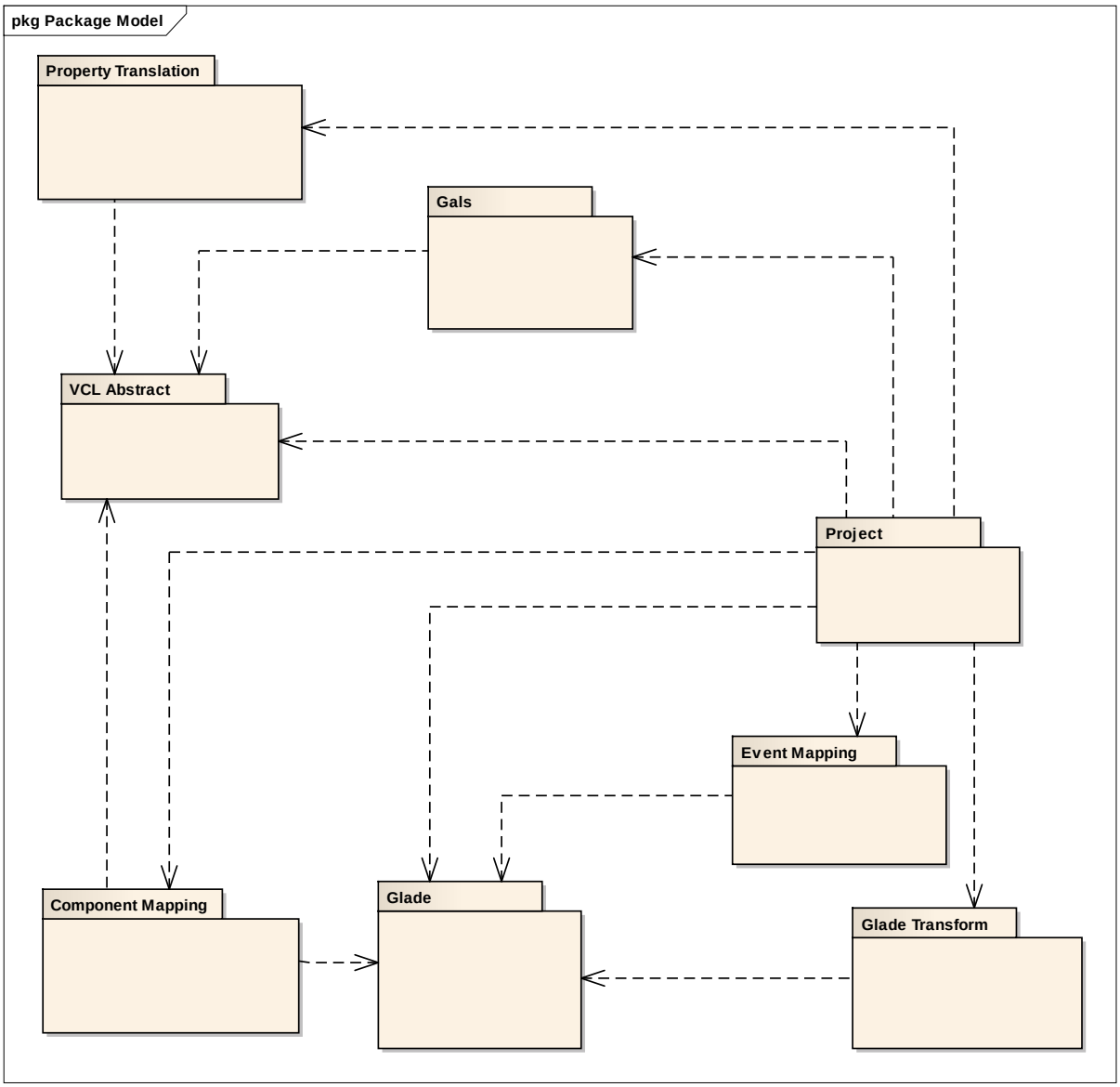

Figura 7 - Diagrama de pacotes

Nas seções subsequentes cada um dos pacotes apresentados é abordado em detalhes e também são apresentados os diagramas de classes, destacando as classes que possuem

responsabilidades mais relevantes na ferramenta. Apenas o pacote `GALS` não é abordado por tratar-se de uma tecnologia já conhecida do público acadêmico e bastante explorada em outros trabalhos científicos na área da computação.

3.5.2.1 Pacote `VCL Abstract`

O pacote `VCL Abstract` engloba um conjunto de classes responsável pelo mapeamento das informações coletadas pelos analisadores léxico, sintático e semântico do pacote `Gals`. Tais classes fornecem uma abstração para os elementos da biblioteca VCL, armazenando as informações relevantes à realização da tradução para a biblioteca `Libglade`. As classes presentes no pacote `VCL Abstract`, com seus relacionamentos, são apresentadas na Figura 8.

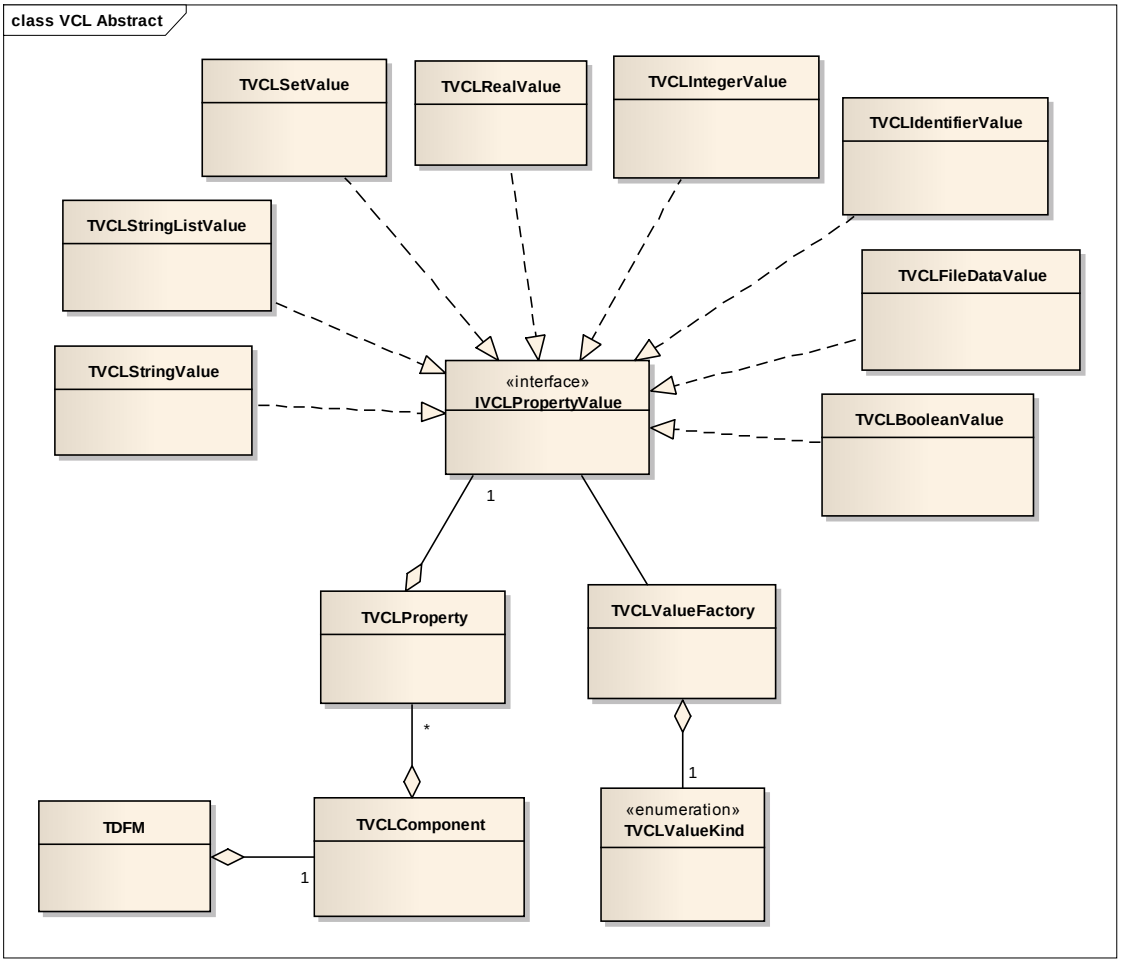

Figura 8 - Diagrama de classes do pacote `VCL Abstract`

A interface `IVCLPropertyValue` fornece um nível de abstração para os valores das propriedades dos componentes da VCL. A especificação da ferramenta prevê a possibilidade

de oito diferentes tipos de dados como valores de propriedades. Para cada um dos tipos previsto há uma classe que implementa a interface `IVCLPropertyValue`, de forma a interpretar o valor proveniente do arquivo `.dfm` de forma apropriada ao tipo em questão. O quadro 14 apresenta as classes mencionadas e os tipos por elas processados.

classe	descrição
`TVCLBooleanValue`	Valor do tipo lógico: `true` ou `false`.
`TVCLFileDataValue`	Cadeia de hexadecimais, representando o conteúdo de um arquivo.
`TVCLIdentifierValue`	Identificador para um elemento de enumeração ou nome de objeto.
`TVCLIntegerValue`	Valor inteiro sinalizado.
`TVCLRealValue`	Valor de ponto flutuante.
`TVCLSetValue`	Lista de identificadores que compõe um conjunto.
`TVCLStringListValue`	Lista de `strings`.
`TVCLStringValue`	Valor do tipo `string`.

Quadro 14 - Classes para interpretação de valores de propriedades

A interface `IVCLPropertyValue` fornece às classes que a implementam métodos para informar e recuperar o valor por ela armazenado. Este nível de abstração permite recuperar e enviar valores para estas classes sem necessidade de conhecer o tipo de dado por ela armazenado.

O pacote `VCL Abstract` fornece também a classe `TVCLValueFactory`, utilizada para criar objetos do tipo `IVCLPropertyValue`. Esta classe armazena um atributo do tipo enumeração denominado `TVCLValueKind`, no qual estão declarados oito elementos, cada qual representando um tipo de dado específico, conforme detalha o quadro 15.

elemento	classe representada
`vkBoolean`	`TVCLBooleanValue`
`vkFileData`	`TVCLFileDataValue`
`vkIdentifier`	`TVCLIdentifierValue`
`vkInteger`	`TVCLIntegerValue`
`vkReal`	`TVCLRealValue`
`vkSet`	`TVCLSetValue`
`vkStringList`	`TVCLStringListValue`
`vkString`	`TVCLStringValue`

Quadro 15 - Elementos declarados na enumeração `TVCLValueKind`

À classe `TVCLValueFactory` é enviado o valor de uma propriedade de componente da VCL e também informa-se o tipo de dado armazenado por esta propriedade, alterando-se o valor do atributo `FKind`, que armazena um elemento da enumeração `TVCLValueKind`.

A classe `TVCLProperty` contém os atributos `FName` e `FValue`, que armazenam o nome e valor da propriedade respectivamente. O atributo `FValue` é uma instância de uma classe descendente da interface `IVCLPropertyValue`, o que permite que os valores nele armazenados possam ser instanciados com auxílio da classe `TVCLValueFactory`.

A classe `TVCLComponent` contém todas as informações pertinentes a um componente gráfico da VCL. Esta classe é composta por uma lista de propriedades (objetos do tipo

`TVCLPropertyValue`) e uma lista de componentes filhos (objetos do tipo `TVCLComponent`), caso o componente armazene outros componentes. Ademais, possui os atributos `FName` e `FVCLClassName`, que armazenam o nome e a classe do componente respectivamente.

Por fim, a classe `TDFM` representa o conteúdo de um arquivo `.dfm`. Contém apenas um atributo do tipo `TVCLComponent`, que é uma referência para o componente que representa um formulário Delphi. Todos os outros componentes que compõem a interface gráfica estarão contidos no formulário ou dentro de componentes filhos, recursivamente.

3.5.2.2 Pacote `Glade`

Ao passo que o pacote `VCL Abstract` fornece uma camada de abstração para o conteúdo de um arquivo `.dfm`, o pacote `Glade` fornece uma camada de abstração às informações armazenadas num arquivo de definição de interface gráfica segundo as especificações da `Libglade`. As classes que o compõem são apresentadas na Figura 9.

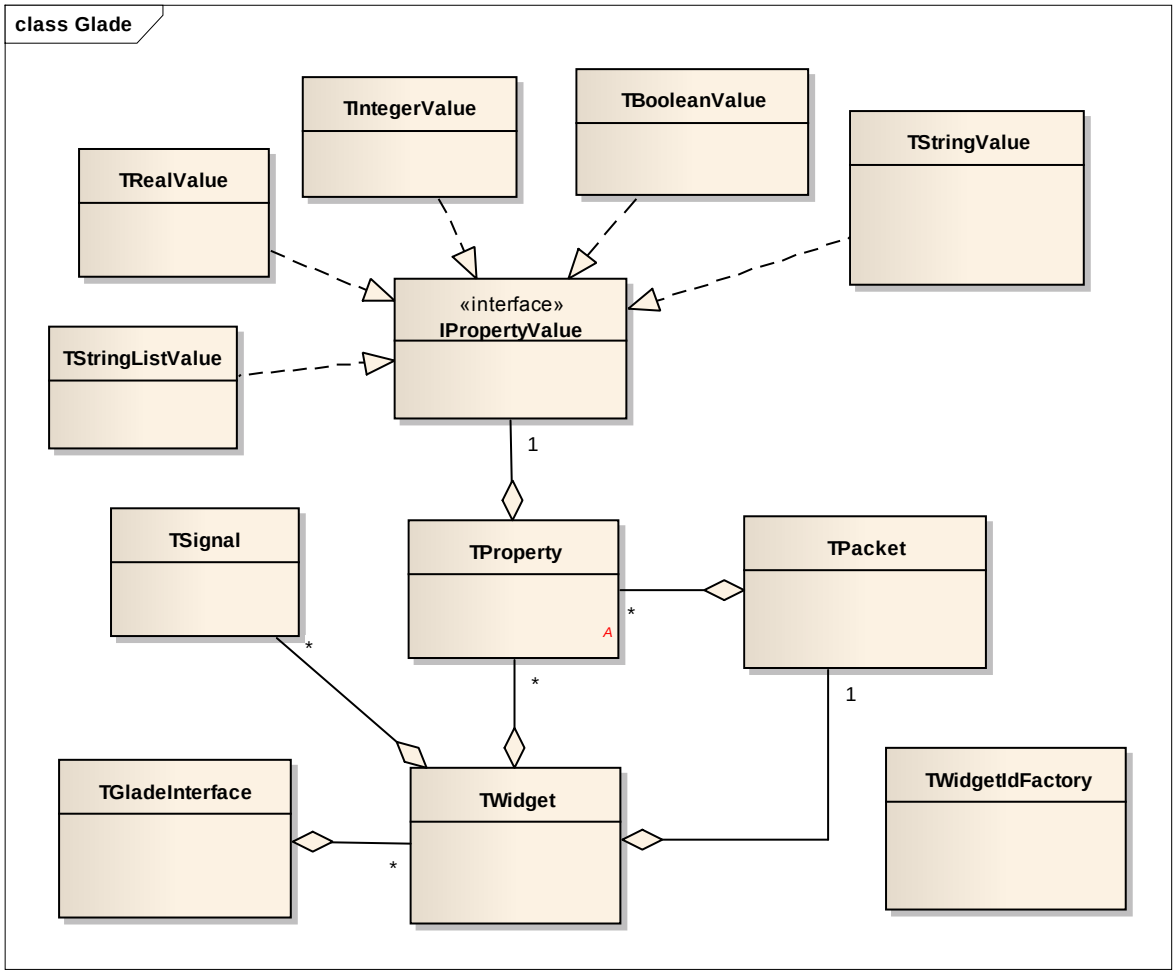

Figura 9 - Diagrama de classes do pacote `Glade`

Tal como na biblioteca VCL, as propriedades presentes nos elementos da GTK+ podem armazenar valores de diferentes tipos de dados. Portanto, para comportar os possíveis tipos de dados foi adotada uma solução similar à utilizada no pacote `VCL Abstract` para armazenamento de valores de propriedades. Uma interface denominada `IPropertyValue` fornece um nível abstração para armazenamento de valores nas propriedades, estas representadas pela classe `TProperty`. O Quadro 16 apresenta os cinco tipos de dados possíveis para propriedades dos elementos da GTK+ e a respectiva classe que implementa a interface `IPropertyValue`.

classe	descrição
`TBooleanValue`	Valor do tipo lógico: `true` e `false`.
`TIntegerValue`	Valor inteiro.
`TRealValue`	Valor monetário e de ponto flutuante.
`TStringListValue`	Lista de `strings`.
`TStringValue`	Valor do tipo `string`.

Quadro 16 - Classes que implementam a interface `IPropertyValue`

A classe `TSignal` representa um sinal da GTK+. Esta classe possui dois atributos que armazenam o nome do sinal e o nome da função de *call-back* responsável por interceptar o sinal.

Na classe `TPacket` estão armazenadas as propriedades referentes ao empacotamento do elemento gráfico em relação ao componente que o contém dentro da interface.

Todas as informações relevante a um elemento gráfico da GTK+ estão encapsulados na classe `TWidget`. Nesta encontram-se atributos que armazenam o seu identificador, classe, propriedades, sinais, empacotamento e elementos contidos.

A classe `TGladeInterface` representa todo o conteúdo da interface gráfica. Contém apenas uma lista dos elementos gráficos de nível superior na interface gráfica. Comumente estará presente apenas um elemento do tipo janela, dentro do qual estarão armazenados todos os outros elementos que compõem a interface gráfica.

3.5.2.3 Pacote `Component Mapping`

Através da interpretação de um arquivo XML com as definições sobre a tradução dos componentes gráficos da biblioteca VCL para a biblioteca GTK+, este pacote fornece um conjunto de classes que realizam um mapeamento dos componentes de forma a padronizar o processo de tradução. As classes que o compõem e seus relacionamentos são apresentados na Figura 10.

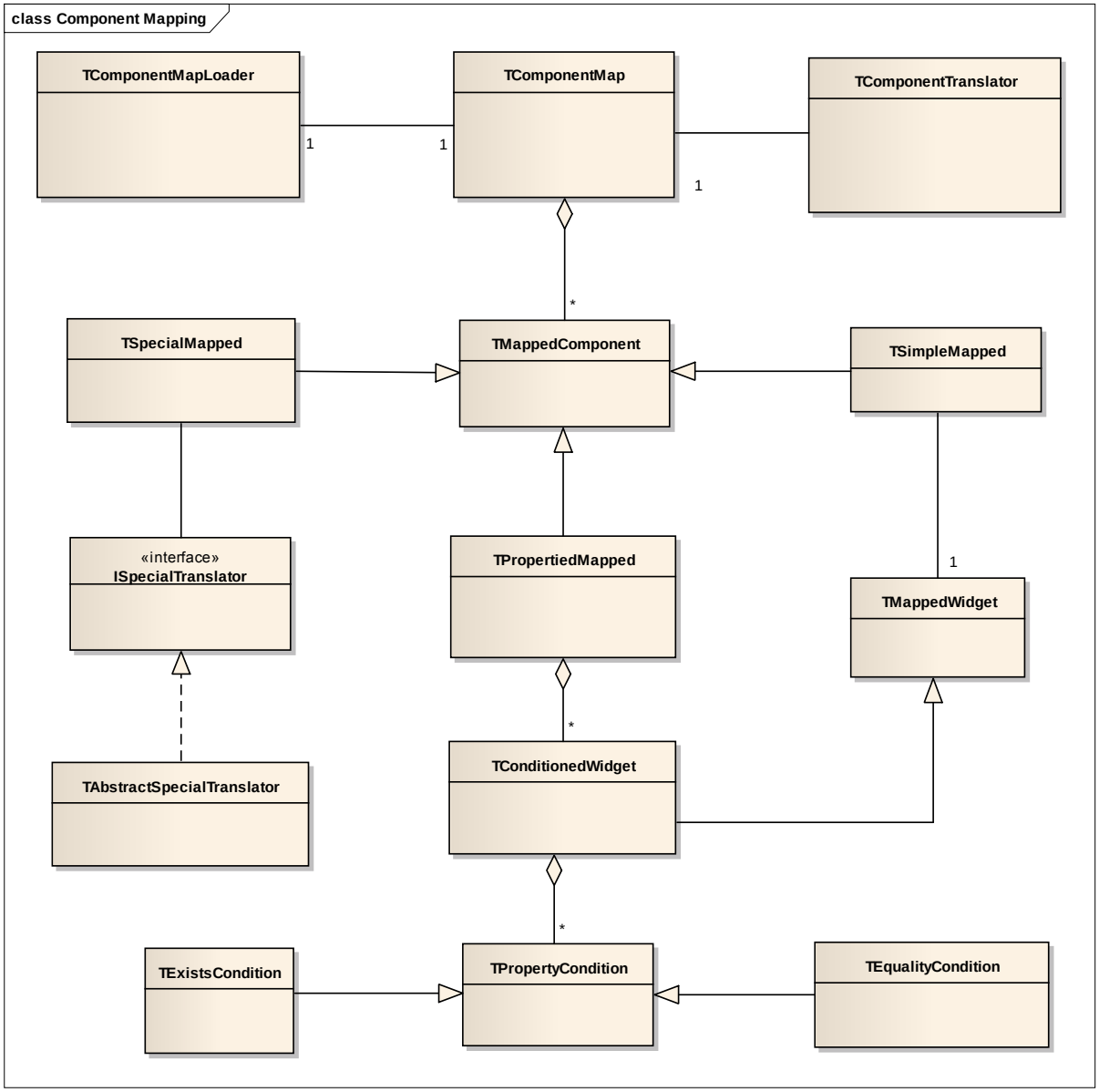

Figura 10 - Diagrama de classes do pacote `Component Mapping`

Um componente pode ser mapeado de três maneiras diferentes de acordo com a categoria de tradução: simples, condicionado e especial. Para cada uma das categorias foi implementada uma classe descendente de `TMappedComponent`: `TSimpleMapped`, `TPropertiedMapped` e `TSpecialMapped`. Enquanto a classe base `TMappedComponent` contém um atributo que indica o nome da classe do componente na biblioteca VCL, as classes especialistas armazenam informações pertinentes à categoria de tradução.

Visto que a classe `TSimpleMapped` representa um componente mapeado cuja categoria de tradução é simples, ela armazena apenas um atributo do tipo `TMappedWidget`, que faz referência ao elemento GTK+ para o qual o componente VCL será traduzido. Na classe `TMappedWidget` está armazenado o nome da classe do elemento GTK+ para o qual o componente VCL será traduzido.

Responsável pelo mapeamento de um componente VCL classificado como de tradução condicionada às propriedades, a classe `TPropertiedMapped` armazena informações referentes aos condicionamentos das propriedades para realização da tradução. Duas condições foram previstas: existência e igualdade. A condição de existência indica que determinada propriedade deve estar presente no componente para que a tradução seja realizada. A condição de igualdade indica que determinada propriedade deve conter um valor previamente informado no mapeamento para a realização da tradução. As condições são representadas pelas classes `TExistsCondition` e `TEqualityCondition`, descendentes de `TPropertyCondition`. A classe `TConditionedWidget` contém uma lista das condições que devem ser atendidas para sua tradução, ao passo que a classe `TPropertiedMapped` armazena uma lista de objetos `TConditionedWidget` das possíveis traduções.

Os componentes da categoria de tradução especial são representados pela `TSpecialMapped`. Esta contém apenas uma referência para um objeto que implementa a interface `ISpecialTranslator`. Portanto, para um componente de tradução especial faz-se necessário o desenvolvimento de uma classe que implementa `ISpecialTranslator` e será responsável pelo processamento e tradução do componente.

Todo o mapeamento de componentes é armazenado em um objeto do tipo `TComponentMap`. Esta classe referencia a classe `TComponentMapLoader`, destinada à interpretação de um arquivo XML que contém o mapeamento da tradução dos componentes e carregamento destas informações para as classes de mapeamento.

Por fim, a classe `TComponentTranslator` é responsável por realizar a tradução dos componentes categorizados como simples e condicionados. Para tanto, utiliza o mapa carregado para a classe `TComponentMap`. Para os componentes de tradução especial esta classe efetua chamada à classe específica de tradução definida para o componente através da abstração fornecida pela interface `ISpecialTranslator`.

3.5.2.4 Pacote `Property Translation`

Tal como o pacote `Component Mapping`, o pacote `Property Translation` realiza a interpretação de um arquivo de mapeamento de tradução no formato XML. Ao passo que as classes do pacote apresentado no tópico anterior são responsáveis pelo mapeamento e tradução dos componentes da VCL, este, por sua vez, encapsula classes que realizam o

mapeamento das propriedades de componentes VCL para propriedades equivalentes na biblioteca `Libglade`. O diagrama de classes do pacote é apresentado na Figura 11.

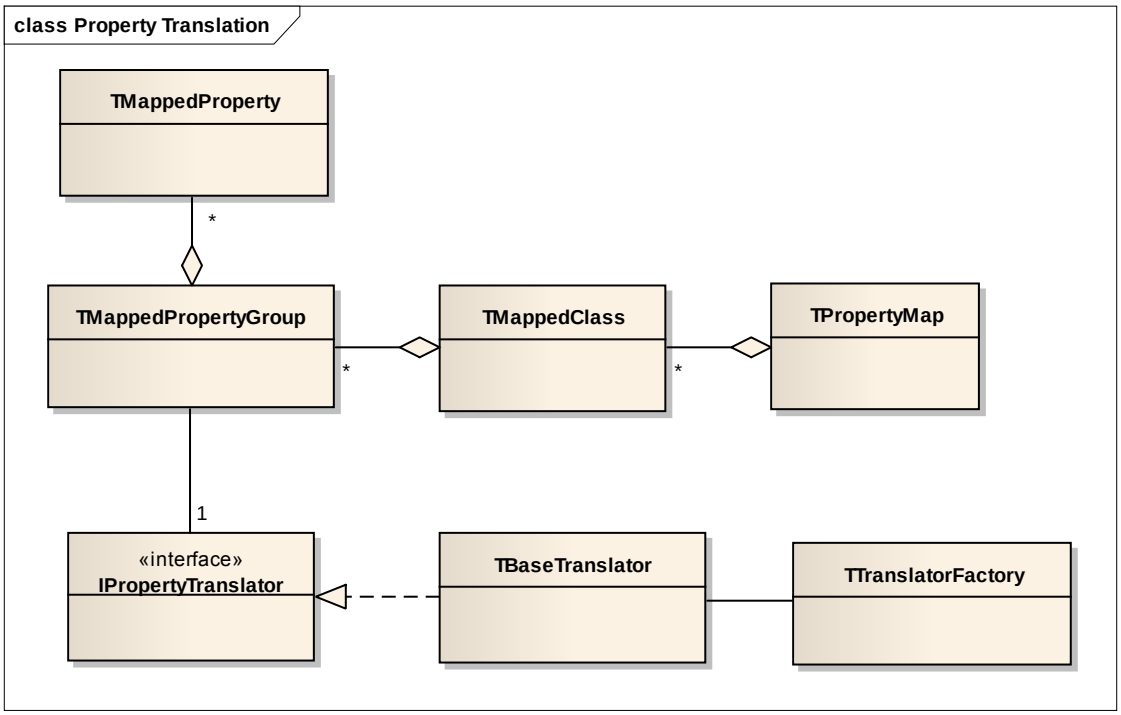

Figura 11 - Diagrama de classes do pacote `Property Translation`

Toda propriedade de componente VCL cuja tradução é plausível para a biblioteca `Libglade` será mapeada em uma instância da classe `TMappedProperty`. Esta classe armazena uma referência para um objeto do tipo `TVCLProperty` que será traduzido e o valor padrão que será assumido para a propriedade VCL, caso esteja ausente no componente, é armazenado no atributo `FDefaultValue`, do tipo `IVCLPropertyValue`.

Uma propriedade VCL, na sua tradução para a `Libglade`, pode tornar-se um conjunto de propriedades; ou um conjunto de propriedades VCL pode tornar-se uma única propriedade na `Libglade` na sua tradução. Portanto, a classe `TMappedPropertyGroup` encapsula um conjunto de propriedades VCL e também uma referência para uma classe tradutora de propriedades. As classes tradutoras devem estender a classe `TBaseTranslator`, que constitui uma base para os tradutores de propriedades, uma vez que esta implementa a interface `IPropertyTranslator`, onde estão declarados os métodos que uma classe tradutora de propriedades deve implementar.

A classe `TMappedClass` representa um componente VCL que contém propriedades mapeadas e traduzíveis para a biblioteca `Libglade`. A classe armazena uma lista de conjuntos de propriedades mapeadas e o nome da classe VCL correspondente ao componente por ela representado. Todos os componentes VCL que possuem propriedades mapeadas para a

Libglade são encapsulados na classe TPropertyMap, que fornece um acesso centralizado à todas as propriedades VCL mapeadas, além de métodos que auxiliam no processo de tradução.

A última classe apresentada neste pacote denomina-se TTranslatorFactory. Esta classe é uma fábrica de tradutores de propriedades. Para cada conjunto de propriedades VCL mapeadas existe um tradutor devidamente nomeado. Através da interpretação do nome do tradutor, esta classe é capaz de instanciar um objeto descendente da classe TBaseTranslator.

3.5.2.5 Pacote Event Mapping

O terceiro pacote (Figura 12) responsável por mapeamento de aspectos da VCL denomina-se Event Mapping, e seu objetivo é fornecer um conjunto de classes com capacidade para mapear as equivalências entre os eventos de componentes da VCL para sinais de elementos gráficos da Libglade.

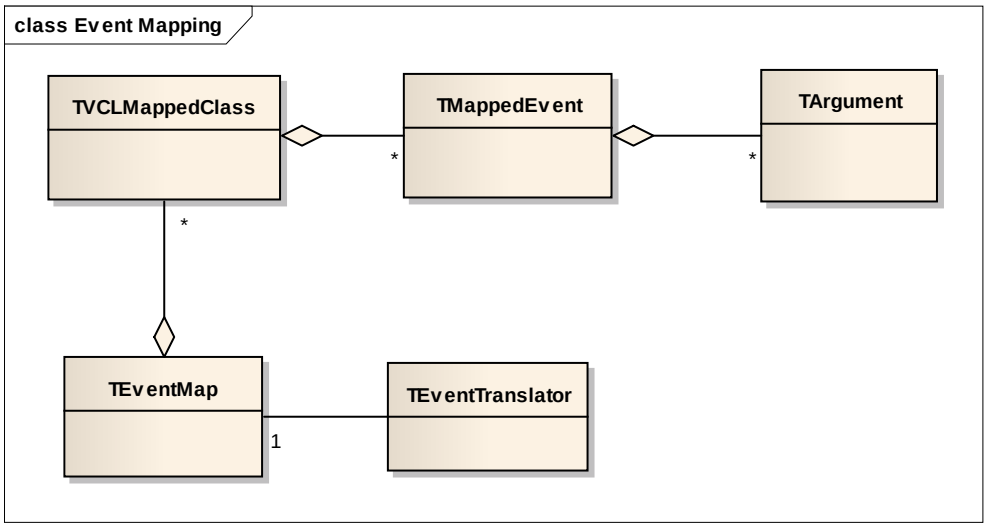

Figura 12 - Diagrama de classes do pacote Event Mapping

O mapeamento do evento é encapsulado na classe TMappedEvent, na qual estão armazenados o nome do evento VCL, o nome do sinal equivalente na Libglade e a lista de argumentos que são enviados à função de *call-back* responsável por interceptar o sinal da Libglade. A lista de argumentos constitui-se de uma lista de objetos do tipo TArgument. Esta classe, por sua vez, é composta de dois atributos: o nome do argumento e o nome do tipo do argumento.

A classe TVCLMappedClass representa um componente VCL cujo eventos estão mapeados para a biblioteca Libglade. Nela estão armazenados o nome da classe VCL

correspondente ao componente em questão e uma lista de objetos do tipo `TMappedEvent`, que indicam os eventos do componente que estão mapeados.

Todo o mapeamento carregado é armazenado em um objeto do tipo `TEventMap`. A classe `TEventMap` contém uma lista de componentes VCL para os quais há eventos mapeados para a `Libglade`. Consultando o mapeamento definido na classe `TEventMap`, a classe `TEventTranslator` é capaz de realizar a tradução dos eventos VCL para sinais da `Libglade`.

3.5.2.6 Pacote `Glade Transform`

Como explanado anteriormente, a biblioteca `Libglade` possui a capacidade de carregar uma interface gráfica GTK+ a partir de uma definição de interface segundo a notação XML. Desta forma, o pacote `Glade Transform` (Figura 13) foi concebido com a finalidade de interpretar os objetos do pacote `Glade` e transformá-los em uma definição de interface gráfica legível à `Libglade`.

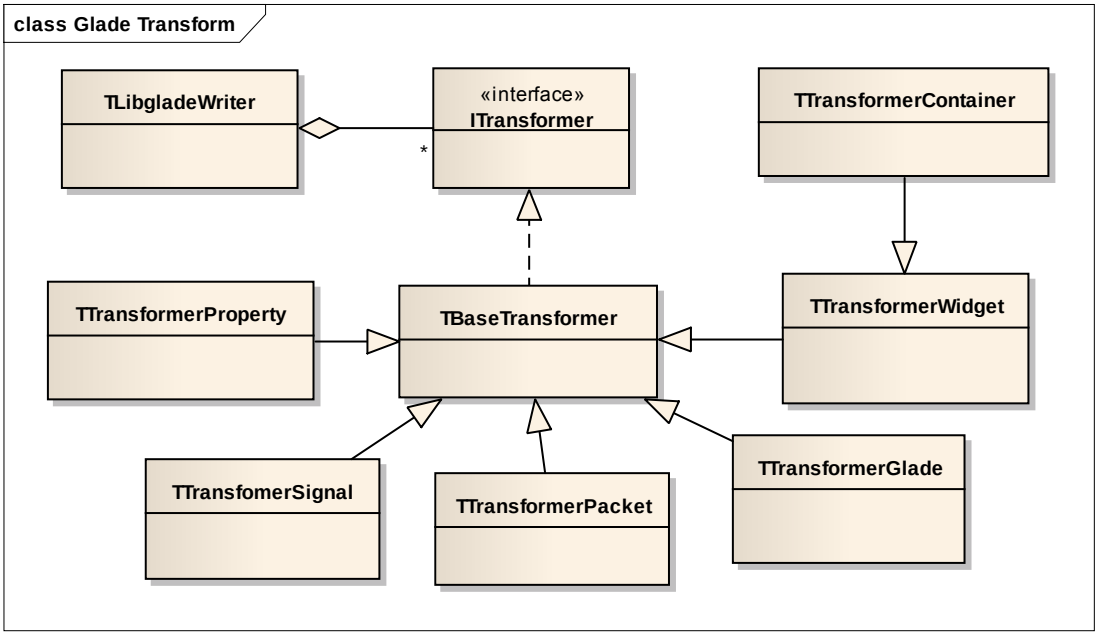

Figura 13 - Diagrama de classes do pacote `Glade Transform`

Para cada um dos objetos do pacote `Glade` que compõem uma interface gráfica `Libglade` há uma classe capaz de transformá-lo em um fragmento de documento XML. Estas classes são descendentes da `TBaseTransformer`, que constitui a base para as classes transformadoras de objetos do pacote `Glade` em fragmentos do documento XML que define a interface gráfica. A classe `TBaseTransformer` implementa a interface `ITransformer` que fornece o conjunto de métodos necessário à realização da transformação. As classes

transformadoras e suas classes correspondentes no pacote `Glade` são apresentadas no Quadro 17.

classe transformadora	classe transformada
`TTransformerProperty`	`TProperty`
`TTransformerSignal`	`TSignal`
`TTransformerPacket`	`TPacket`
`TTransformerGlade`	`TGladeInterface`
`TTransformerWidget`	`TWidget`
`TTransformerContainer`	`TWidget`

Quadro 17 - Classes transformadoras e seus respectivos transformados

A única classe transformadora que não descente diretamente da classe `TBaseTranslator` é a classe `TTransformerContainer`. Tal como sua classe ancestral, a `TTransformerWidget`, esta classe realiza a transformação de objetos `TWidget`. Contudo, esta classe possui a capacidade adicional de verificar objetos do tipo `TWidget` contidos pelo objetos em transformação e acionar o seu transformador. Em outras palavras, esta classe transforma elementos gráficos da GTK+ que contém outros elementos armazenados.

Todo o processo de transformação é desencadeado pela classe `TTransformerGlade`. Verifica-se os objetos contidos no objeto `TGladeInterface` que deseja-se transformar e aciona os seu transformadores específicos. Cada objeto passível de transformação é acessado e transformado. O processo é realizado até que todos os objetos tenham sido devidamente transformados. A classe `TLibgladeWriter` responsabiliza-se por salvar os dados transformados para um documento XML.

3.5.2.7 Pacote `Project`

O pacote `Project` é responsável pela integração de todos os pacotes já apresentados, fornecendo uma interface de desenvolvimento abstrata e de fácil compreensão para executar tradução de interfaces gráficas Delphi em interfaces gráficas GTK+ segundo as definições da biblioteca `Libglade`. As classes que o compõem são apresentadas na Figura 14.

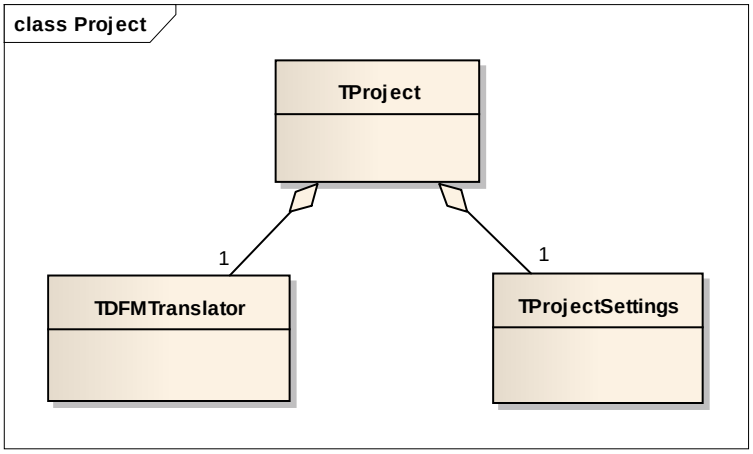

Figura 14 - Diagrama de classes do pacote `Project`

Na classe `TDFMTranslator` estão encapsulados os mapas de componentes, propriedades e eventos. Também estão presentes nesta classe os tradutores de componentes e eventos. Esta classe possui o potencial de interpretar um objeto do tipo `TDFM` e transformá-lo em um objeto do tipo `TGladeInterface`, através da utilização dos mapas e tradutores.

A classe `TProjectSettings` armazena informações pertinentes à preferências definidas para um projeto de tradução. Ela indica os diretórios de entrada e saída e o diretório onde se encontram os arquivos de mapeamento.

As classes `TDFMTranslator` e `TProjectSettings` são encapsuladas pela classe `TProject`, que representa um projeto de tradução. Além destas, a `TProject` possui também instâncias das classes `TLexicon`, `TSyntatic` e `TSemantic`, responsáveis pelo processamento e interpretação de arquivos `.dfm`. Também armazena um objeto do tipo `TLibgladeWriter` para realizar a transformação das interfaces traduzidas em documentos XML legíveis à `Libglade`. As interfaces traduzidas também ficam armazenadas nesta classe; em uma lista de objetos do tipo `TGladeInterface`.

3.5.3 Mapas de tradução

Foram especificados três mapas de tradução que fornecem à ferramenta como os componentes da VCL, suas propriedades e eventos devem ser traduzidos para a biblioteca `Libglade`. Os mapas são arquivos com formatação XML. Os tópicos subsequentes apresentam detalhes sobre a concepção dos mapas.

3.5.3.1 Mapa de componentes

O mapa de componentes é um arquivo XML denominado `componentes.xml`. Nele estão declarados todos os componentes VCL que podem ser traduzidos pela ferramenta DelphiToGTK+ e como a tradução será realizada. O conteúdo do arquivo inicia com um cabeçalho XML que indica a versão da linguagem XML e a codificação do texto. A *tag* principal do arquivo é identificada como `delphitogtk`.

Dentro da *tag* `delphitogtk` estão contidas as *tags* que especificam a tradução dos componentes. Cada componente passível de tradução possuirá uma *tag* denominada `component`, que obrigatoriamente deve conter os atributos `class` e `translation`, que indicam a classe do componente na biblioteca VCL e a categoria de tradução, respectivamente. Os valores permitidos para o atributo `translation` são: `simple`, `propertied` e `special`.

O conteúdo da *tag* `component` difere de acordo com a categoria definida no atributo `translation`. Para componentes da categoria `simple`, a *tag* deve conter apenas uma *tag* denominada `widget` com um atributo `class`, que indica o nome da classe na biblioteca GTK+ equivalente ao componente VCL. O Quadro 18 apresenta um fragmento do arquivo `componentes.xml` no qual é declarado um componente VCL de tradução simples.

```
<component class="TButton" translation="simple">
   <widget class="GtkButton" />
</component>
```

Quadro 18 - Definição de mapeamento de um componente de tradução simples

O trecho do arquivo `componentes.xml` apresentado no Quadro 18 indica que o componente `TButton` da biblioteca VCL deve ser traduzido para a classe `GtkButton` da biblioteca GTK+. Os componentes cuja tradução é condicionada devem conter na *tag* `component`, obrigatoriamente, uma *tag* denominada `widgets`. Nesta *tag* serão informadas as classes da GTK+ para as quais o componente VCL pode ser traduzido. Cada possibilidade de tradução do componente é declarado em uma *tag* identificada pelo nome `widget`, que possui um atributo `class` que indica a classe correspondente ao elemento gráfico na biblioteca GTK+. A *tag* `widget`, por sua vez, conterá uma *tag* chamada `properties`, na qual são declaradas as propriedades que condicionam a tradução do componente VCL. Cada propriedade que condiciona a tradução do componente deve ser declarada em uma *tag* `property`, dentro da *tag* `properties`. Cada *tag* `property` pode conter até quatro atributos, denominados `name`, `type`, `check` e `value`; que indicam, respectivamente, o nome da

propriedade do componente VCL a verificar, o tipo do valor armazenado pela propriedade, a espécie de verificação que será aplicada e o valor padrão que será assumido para a propriedade caso esta esteja ausente no componente VCL. Dos quatro atributos, apenas `value` é opcional. Um exemplo de componente mapeado para tradução condicionada às propriedades é apresentado no Quadro 19.

```
<component class="TTrackBar" translation="propertied">
   <widgets>
      <widget class="GtkHScale">
         <properties>
            <property name="Orientation"
                      type="Identifier"
                      check="equality"
                      value="trHorizontal" />
         </properties>
      </widget>
      <widget class="GtkVScale">
         <properties>
            <property name="Orientation"
                      type="Identifier"
                      check="equality"
                      value="trVertical" />
         </properties>
      </widget>
      <widget class="GtkHScale" />
   </widgets>
</component>
```

Quadro 19 - Mapeamento de um componente de tradução condicionada à propriedade

Os possíveis valores para o atributo `type` da *tag* `property` são especificados no Quadro 20.

valor	descrição
`Boolean`	Valores de tipo lógico: `true` e `false`.
`FileData`	Valores hexadecimais.
`Identifier`	Valores que representam identificadores.
`Integer`	Valores numéricos inteiros.
`Real`	Valores monetários e de ponto flutuante.
`Set`	Conjuntos.
`StringList`	Listas de `strings`.
`String`	Valores do tipo `string`.

Quadro 20 - Valores válidos para o atributo `type` da *tag* `property`

Para o atributo `check`, dois valores são válidos: `equality` e `exists`. `Equality` indica que o componente deve conter a propriedade indicada com o valor igual ao informado no atributo `value`. `Exists` indica que a propriedade deve estar presente no componente para que a tradução seja realizada.

Para componentes da categoria de tradução condicionada à propriedade, é possível informar uma *tag* `widget` sem quaisquer condicionamentos de propriedades. Desta forma, caso o componente não atenda os condicionamentos de qualquer das *tags* `widget` compreendidas, esta *tag* sem condicionamentos será assumida para tradução.

A categoria de tradução especial é especificada apenas com uma *tag* `component` com os atributos `class`, `translation` e `translator`. Tal como nas especificações de tradução para as categorias simples e condicionada a propriedades, o atributo `class` armazena o nome da classe correspondente ao componente VCL. O atributo `translation` deve conter o valor `special`. O atributo `translator`, por sua vez, deve conter o nome da classe implementada na ferramenta DelphiToGTK+ capaz de realizar a tradução do componente VCL em questão. Esta classe deve estender a classe `TAbstractSpecialTranslator`. Um exemplo de mapeamento especial é apresentado no Quadro 21.

```
<component class="TSpinButton" translation="special"
           translator="TSpinButtonTranslator" />
```

Quadro 21 - Exemplo de mapeamento da categoria especial

A *tag* `widget`, presente no mapeamento de componentes das categorias `simple` e `propertied`, podem conter uma *tag* opcional denominada `required-properties`. Nesta *tag* são informadas as propriedades que deverão obrigatoriamente ser criadas para o elemento GTK+ resultante da tradução do componente VCL. Cada propriedade é identificada pela *tag* `property`, que contém, obrigatoriamente, os atributos `name` e `value`, sendo o primeiro o nome da propriedade e o segundo o seu valor. No Quadro 22 é apresentado um modelo de componente mapeado cuja tradução especifica propriedades requeridas.

```
<component class="TCheckBox" translation="simple">
   <widget class="GtkCheckButton">
      <required-properties>
         <property name="draw_indicator" value="True" />
      </required-properties>
   </widget>
</component>
```

Quadro 22 - Exemplo de mapeamento que contém atributos requeridos

3.5.3.2 Mapa de propriedades

Identificado pelo nome `properties.xml`, este arquivo contém as especificações sobre como as propriedades dos componentes da VCL serão traduzidas para a `Libglade`. A *tag* raiz do arquivo denomina-se `delphitogtk`. Cada componente, cujas propriedades podem ser traduzidas para a `Libglade`, deve ser representado por uma denominada `class` que possui um atributo `name`, no qual é informado o nome da classe correspondente ao componente na biblioteca VCL.

A *tag* `class` deve conter obrigatoriamente apenas uma *tag* `groups`, dentro da qual serão especificados os conjuntos de propriedades passíveis de tradução. Isto se faz necessário,

pois um conjunto de propriedades da VCL pode ser traduzido em uma única propriedade para a `Libglade`. Um conjunto de propriedades é contido por uma *tag* `group`, declarada dentro da *tag* `groups`. A *tag* `group`, por sua vez, deve conter obrigatoriamente uma *tag* `properties`, onde serão informadas as propriedades da VCL que compõem o conjunto de propriedades. Na *tag* `group` deve-se informar o atributo `translator`. Neste deve-se indicar o identificador da classe responsável pela tradução do conjunto de propriedades contidas pela *tag* `group`. As classes tradutoras de propriedades descendem da classe `TBaseTranslator` que implementa a interface `IPropertyTranslator` e estão implementadas na unidade `upropertytranslation` do código fonte da ferramenta.

Dentro da *tag* `properties` são declaradas as propriedades do componente VCL que compõem o grupo de tradução. Cada propriedade é especificada em uma *tag* denominada `property` que deve conter os atributos `name`, `required`, `type` e `default`. No atributo `name` é informado o nome da propriedade VCL e no atributo `default` permite armazenar um valor que será assumido para a propriedade caso a mesma não esteja presente no componente. O atributo `type` segue a mesma regra do atributo homônimo da *tag* `property` na especificação do mapeamento de componentes explanada na seção anterior. O atributo `required` indica se esta propriedade é necessária para realizar a tradução do grupo, sendo seus possíveis valores `true` ou `false`. Todos os atributos são obrigatórios, contudo, permite-se não informar qualquer valor para o atributo `default`.

O Quadro 23 apresenta fragmento do arquivo `properties.xml` que mostra o mapeamento dos atributos de um componente VCL.

```
<class name="TTrackBar">
  <groups>
    <group translator="Alignment">
      <properties>
        <property name="Position" required="true" type="Integer" default="0" />
        <property name="Min" required="true" type="Integer" default="0" />
        <property name="Max" required="true" type="Integer" default="10" />
        <property name="PageSize" required="true" type="Integer" default="2" />
        <property name="Frequency" required="true" type="Integer" default="1" />
      </properties>
    </group>
  </groups>
</class>
```

Quadro 23 - Exemplo de mapeamento de propriedades

3.5.3.3 Mapa de eventos

O mapa de eventos, especificado no arquivo `events.xml`, objetiva fornecer a

ferramenta as informações necessárias para criar a assinatura dos métodos de *call-back* que processarão os sinais da GTK+ equivalentes aos eventos dos componentes VCL. Tal como os mapas apresentados nas duas seções antecedentes, este também inicia com a *tag* `delphitogtk`, dentro da qual estão armazenadas todas as informações referentes à tradução dos eventos.

Cada componente VCL cujos eventos devem ser traduzidos para a `Libglade` deve possuir uma *tag* `component` dentro da *tag* raiz do arquivo, onde o atributo `class` indica o nome da classe na biblioteca VCL que representa o componente. Cada evento passível de tradução é representado pela *tag* `event`, dentro da *tag* `component`. Esta *tag* deve possuir dois atributos denominados `vcl` e `gtk`. No primeiro é informado o nome do evento na biblioteca VCL e o segundo o nome do sinal da GTK+ similar ao evento VCL.

A *tag* `event` deve possuir uma *tag* `arguments`, onde serão informados os parâmetros que serão enviados à função de *call-back* responsável pelo tratamento do sinal da GTK+. Cada parâmetro é especificado em uma *tag* denominada `argument` que possui os atributos `name` e `type`. O atributo `name` armazena o nome do parâmetro, enquanto `type` guarda o nome do tipo do parâmetro. O Quadro 24 apresenta trecho do arquivo `events.xml` onde os eventos de um componente estão mapeados.

```
<component class="TMenuItem">
   <event vcl="OnClick" gtk="button-press-event">
      <arguments>
         <argument name="Widget" type="PGtkWidget" />
         <argument name="Event" type="PGdkEvent" />
         <argument name="UserData" type="GPointer" />
      </arguments>
   </event>
</component>
```

Quadro 24 - Exemplo de mapeamento de eventos de um componente

3.6 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas para a implementação da ferramenta DelphiToGTK+, bem como detalhes da sua implementação.

3.6.1 Técnicas e ferramentas utilizadas

A ferramenta foi codificada sobre a linguagem de programação Pascal/Object Pascal, utilizando o ambiente de desenvolvimento Lazarus, na sua versão 0.9.28.2 *beta*. Este, por sua vez, faz uso da versão 2.2.4 do compilador Free Pascal. A interface gráfica do DelphiToGTK+ foi construída sobre a biblioteca gráfica multi-plataforma GTK+, com auxílio da ferramenta Glade, versão 3.6.7. Visto que todas as ferramentas envolvidas no desenvolvimento da ferramenta fornecem suporte multi-plataforma, torna-se possível que o software possa ser executado em diferentes sistemas operacionais, bastando apenas que o mesmo seja recompilado para a plataforma para a qual se deseja portá-lo.

Para criação dos arquivos `.dfm` utilizados nos testes da ferramenta foi usado o ambiente de desenvolvimento Borland Developer Studio 2006.

3.6.2 Implementação da ferramenta

Esta seção trata dos aspectos técnicos e práticos envolvidos na construção da ferramenta. São apresentas as classes e rotinas de maior relevância computacional dentro do processo de tradução de interfaces gráficas especificadas em arquivos `.dfm`. Também são relatadas as dificuldades encontradas ao longo do desenvolvimento do trabalho, bem como estratégias adotadas para sua resolução.

3.6.2.1 Customização das classes geradas pelo GALS

A partir da BNF definida por Silveira (2006), foi usado o GALS para gerar as classes e constantes que constituem os analisadores léxico, sintático e semântico. O GALS é capaz de gerar código fonte para três diferentes linguagens de programação: Java, C++ e Delphi. Como a ferramenta foi implementada sobre a linguagem Pascal/Object Pascal, optou-se pela geração dos códigos em Delphi. No código gerado, foi detectada uma falha de chamada recursiva presente no método `NextToken` da classe `TLexicon`. O trecho de código onde se encontra a chamada recursiva que apresentou falha é destacado no Quadro 25.

```
function TLexicon.NextToken: TToken; {...}
begin
      {...}
      if TokenId = 0 then
         Result := Self.NextToken
      else begin
         Lexeme := Copy(FInput, Start, EndPos - Start);
         TokenId := LookupToken(TokenId, Lexeme);
         Result := TToken.Create(TokenId, Start, Lexeme);
      end;
{...}
end;
```

Quadro 25 - Trecho do código do método `NextToken` da classe `TLexicon`

A linha de código destacada no Quadro 25 realiza a chamada recursiva mencionada. No código gerado pelo GALS a chamada recursiva a `NextToken` era declarada sem a precedência da palavra chave `Self`, o que ocasionava um erro em tempo de execução. O erro disparado durante os testes é apresentado na Figura 15.

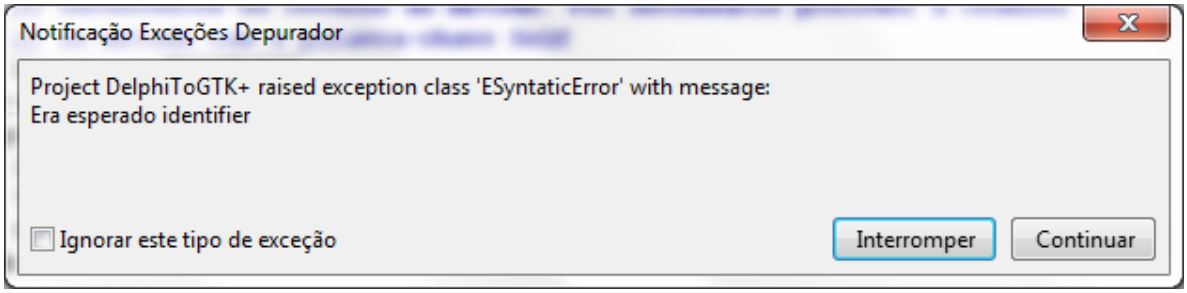

Figura 15 - Erro disparado em decorrência de falha na chamada recursiva a `NextToken`

Das classes geradas pelo GALS a que sofreu maiores alterações foi a `TSemantic`. A esta classe foram adicionados três novos atributos e o método `ExecuteAction` foi implementado para interpretar o conteúdo processado pelos analisadores léxico e sintático a partir de um arquivo `.dfm`.

O atributo `FDFM` da classe `TSemantic` armazena um objeto do tipo `TDFM` que é alimentado com as informações pertinentes à interface gráfica VCL durante o processo de interpretação do arquivo `.dfm`. O atributo `FComponents`, que representa uma pilha de objetos do tipo `TVCLComponent` e `FIdentifier` são auxiliares ao método `ExecuteAction`.

Sempre que encontrar uma ação semântica, o analisador sintático (classe `TSyntatic`) realizará uma chamada à `ExecuteAction` da classe `TSemantic`, enviando como parâmetros o número da ação e um objeto do tipo `TToken`, que representa o último elemento encontrado no arquivo `.dfm` antes da ação semântica. Para atender as necessidades da ferramenta, foram criadas vinte e duas ações semânticas. Para cada ação, o método `ExecuteAction` adota uma estratégia distinta e alimenta gradualmente as classes do pacote `VCL Abstract` através do atributo `FDFM`. As ações semânticas são:

a) ação semântica #1: executada após um identificador de componente VCL ser

encontrado. É criado um novo objeto do tipo `TVCLComponent`, cujo identificador está armazenado em `Token`. Se existir algum elemento na pilha de componentes `FStack`, significa que o componente identificado por esta ação está contido no componente anterior, ou seja, o último componente empilhado em `FStack`. O próprio componente aqui encontrado, por sua vez, também é empilhado em `FStack`;

b) ação semântica #2: executada após um identificador de classe de componente VCL ser encontrado. O componente criado na ação #1 é resgatado da pilha `FStack` e o identificador da classe que o representa é recuperado do argumento `Token`. Porém, se for o primeiro componente encontrado, significa que é da classe `TForm`, que representa um formulário Delphi, já que todos os componentes de uma interface gráfica Delphi estão contidos dentro de um formulário;

c) ação semântica #3: pode ser executada em duas situações diferentes, ou após o reconhecimento do identificador de uma propriedade ou após o reconhecimento do valor de uma propriedade. O segundo caso ocorre somente quando o valor da propriedade representa um identificador ou um conjunto. Para conhecer em qual das duas situações está sendo executada esta ação, utiliza-se o atributo `Reading` da classe `TVCLValueFactory`. Se o valor desse atributo é `True`, significa que a ação foi identificada após ser encontrado o valor da propriedade e, portanto, o valor armazenado no argumento `Token` é enviado para a classe `TVCLValueFactory`; caso contrário, o identificador da propriedade fica armazenado no atributo `FIdentifier`;

d) ações semânticas #4 e #5: as ações #4 e #5 são disparadas somente quando encontrado um identificador composto (de propriedade VCL). Um identificador composto é formado por dois ou mais identificadores separadores pelo caractere ponto. A ação #4 indica que foi encontrado o caractere ponto, ao passo que a ação #5 indica que foi encontrado um identificador complementar. Tal como a ação #3, as ações #4 e #5 podem indicar o reconhecimento do valor de uma propriedade ou do seu identificador, de acordo com o estado do atributo `Reading` da classe `TVCLValueFactory`. Para o primeiro caso, o valor armazenado no argumento `Token` é enviado à classe `TVCLValueFactory`, ao passo que para o segundo caso o valor é concatenado ao atributo `FIdentifier`;

e) ação semântica #6: nessa ação semântica o último elemento encontrado é o

símbolo de igualdade, indicando que o nome de uma propriedade VCL foi completamente processado. Uma vez que se tem definido o nome da propriedade, cria-se um novo objeto do tipo `TVCLProperty` e adiciona-o ao último componente armazenado na pilha de componentes `FStack`. Neste momento, o analisador semântico indica à classe `TVCLValueFactory` que iniciar-se-á a leitura do valor da propriedade criada, definindo o valor do atributo `Reading` para `True`;

f) ações semânticas #7 e 8: ocorrem após o reconhecimento dos símbolos de adição e subtração, respectivamente. Estes símbolos precedem valores de propriedades dos tipos inteiro e de ponto flutuante. Contudo, neste ponto, não se sabe o tipo do valor que está sendo processado;

g) ações semânticas #9, #10, #11, #13 e #14: estas ações são disparadas quando do reconhecimento de um valor de uma propriedade. As ações #9, #10 e #11 indicam o reconhecimento de valores inteiro, de ponto flutuante e *string*, respectivamente; ao passo que as ações #13 e #14 indicam o reconhecimento dos valores `False` e `True`, respectivamente. Nestas ações o último elemento encontrado é enviado à classe `TVCLValueFactory` e o analisador semântico passa a conhecer o tipo do valor da propriedade e indica isso à `TVCLValueFactory`, através da propriedade `Kind`;

h) ação semântica #12: esta ação é disparada antes de iniciar o reconhecimento de um valor de propriedade do tipo identificador. Esta informação é enviada à classe `TVCLValueFactory`, atribuindo o valor `vkIdentifier` ao atributo `Kind`;

i) ações semânticas #15 e #16: a ação #15 indica que um símbolo abre-colchete foi reconhecido, ao passo que a ação #16 indica que um símbolo fecha-colchete foi encontrado. Isso indica que está sendo efetuada a leitura de um valor de propriedade que representa um conjunto de identificadores. Esta informação é indicada à classe `TVCLValueFactory` alterando o valor da propriedade `Kind` para `vkSet`;

j) ações semânticas #17 e #18: as ações #17 e #18 são disparadas logo após o reconhecimento de um símbolo de abre-parêntese ou fecha-parêntese. Para os dois casos o analisador semântico indica à `TVCLValueFactory` que está sendo efetuada a leitura de uma lista de *strings*, enviando o valor `vkStringList` ao atributo `Kind`;

k) ações semânticas #19 e #20: indicam a leitura de um valor de propriedade que representa uma cadeira de caracteres hexadecimais. Tais valores são delimitados

pelos símbolos de abre-chaves e fecha-chaves. O analisador semântico indica à classe `TVCLValueFactory` que está efetuando o reconhecimento de uma cadeia de hexadecimais, enviando o valor `vkFileData` ao atributo `Kind`;

l) ação semântica #21: a ação #21 é disparada logo após finalizar o reconhecimento do valor da última propriedade criada por ocasião da ação #6. A esta propriedade é atribuído o valor criado pela classe `TVCLValueFactory`, com base nas informações resgatadas nas ações #3, #4, #5, #7, #8, #9, #10, #11,#12, #13, #14, #15, #16, #17, #18, #19 e #20;

m) ação semântica #22: é a última ação executada após o reconhecimento total de um componente. O componente adicionado à pilha `FStack` por ocasião da ação semântica #1 é dela removido.

3.6.2.2 Inicialização dos mapas de tradução

Uma vez que a interface gráfica definida em um arquivo `.dfm` foi propriamente processada e carregada para o conjunto de classes do pacote `VCL Abstract`, os mapas de tradução são carregados para a aplicação. Como já mencionado, a ferramenta utiliza três diferentes mapas: mapa de componentes, mapa de propriedades e mapa de eventos. Para cada mapa há um pacote de classes específico, com um conjunto de classes capazes de processar o documento XML e transformá-lo em um formato legível à ferramenta.

O compilador Free Pascal fornece um conjunto de classes pré-compiladas para processamento de documentos que utilizam a notação XML. Para carregar um documento XML foi utilizada a classe `TXMLDocument`, definida na unidade `DOM`, distribuída junto ao Free Pascal. O carregamento de um documento XML para a classe `TXMLDocument` é realizado através da chamada à função `ReadXMLFile`, declarada na unidade `XMLRead` do Free Pascal. Esta função recebe dois parâmetros, sendo o primeiro o objeto `TXMLDocument` para o qual o documento XML será carregado e o segundo o caminho completo de localização do documento XML que se deseja carregar. O Quadro 26 mostra o trecho de código do método `Load` da classe `TComponentMapLoader`, destacando o comando que efetua o carregamento do documento XML.

```
procedure TComponentMapLoader.Load(const XMLPath: string);
var
   Node: TDOMNode;
   Translation: string;
begin
   ReadXMLFile(FXMLMap, XMLPath);
   try
      Node := FXMLMap.DocumentElement.FirstChild;
      while Node <> nil do begin
         Translation := Node.Attributes.GetNamedItem('translation').NodeValue;
         if Translation = 'simple' then
            LoadSimpleMapped(Node)
         else
         if Translation = 'propertied' then
            LoadPropertiedMapped(Node)
         else
         if Translation = 'composed' then
            LoadComposedMapped(Node)
         else
         if Translation = 'special' then
            LoadSpecialMapped(Node);
         Node := Node.NextSibling;
      end;
   finally
      FXMLMap.Free;
   end;
end;
```

Quadro 26 - Código do método `Load` da classe `TComponentMapLoader`

Através destes recursos para interpretação de arquivos XML efetua-se o carregamento dos mapas de tradução de componentes, propriedades e eventos para as classes dos pacotes `Component Mapping`, `Property Translation` e `Event Mapping`.

3.6.2.3 Inicialização do tradutor de interfaces

Tendo sido os mapas devidamente carregados dos arquivos XML, torna-se possível realizar a tradução de uma interface gráfica Delphi para a biblioteca `Libglade`. A tradução completa da interface gráfica é gerenciada pela classe `TDFMTranslator`, que encapsula os mapas. Esta classe realiza a tradução a partir de um objeto do tipo `TDFM` devidamente carregado pelo analisador semântico com base em informações interpretadas do arquivo `.dfm` de origem.

Para instanciar a classe `TDFMTranslator` deve-se passar ao construtor o caminho do diretório onde estão localizados os arquivos de mapeamento. O construtor da `TDFMTranslator` procurará pelos arquivos `componentes.xml`, `properties.xml` e `events.xml` dentro do diretório indicado. Se os três arquivos estão presentes no diretório, então os mapas serão inicializados através da chamada ao método `LoadMaps`.

Tendo sido a classe `TDFMTranslator` instanciada, a tradução pode ser realizada através da chamada ao método `Translate` que recebe como parâmetro um objeto do tipo `TDFM` que foi carregado pelo analisador semântico por ocasião da interpretação do arquivo `.dfm`. O resultado da execução deste método é um objeto do tipo `TGladeInterface` da interface gráfica `Libglade` equivalente a interface gráfica VCL definida no arquivo `.dfm` de origem.

Tendo sido acionado, o método `Translate` cria um novo objeto da classe `TGladeInterface`, no qual será armazenado o objeto `TWidget` equivalente ao formulário Delphi traduzido. A tradução dos componentes é realizada de forma recursiva pela chamada à `PerformTranslation`, a partir do método `Translate`, enviando como parâmetro o objeto `TVCLComponent` referente ao formulário Delphi. O Quadro 27 apresenta o código do método `Translate`.

```
function TDFMTranslator.Translate(const DFM: TDFM): TGladeInterface;
begin
   Result := TGladeInterface.Create;
   Result.Widgets.Add(PerformTranslation(DFM.Form));
end;
```

Quadro 27 - Código fonte do método `Translate` da classe `TDFMTranslator`

3.6.2.4 Tradução do componente

No método `PerformTranslate` ocorre todo o processo de tradução dos componentes e suas propriedades. Este método é também responsável por realizar o posicionamento dos componentes dentro da interface gráfica. Os próximos parágrafos detalham as ações efetuadas pelas linhas de código presentes no corpo do método `PerformTranslate`.

O método `PerformTranslate` inicia com uma chamada ao método `Translate` da classe `TComponentTranslator`, enviando como argumento o objeto `TVCLComponent` que representa o componente VCL a traduzir. Como resultado desta chamada, um objeto do tipo `TWidget`, representando o componente traduzido para a `Libglade`, é criado. O método `Translate` da classe `TComponentTranslator`, quando acionado, procura no mapa de componentes o objeto do tipo `TMappedComponent` referente ao mapeamento do componente que deseja-se traduzir. Esta tarefa é realizada com auxílio do método `Find` da classe `TComponentMap`, cuja instância está armazenada no atributo `FMap`, enviando como parâmetro o nome da classe que representa o componente na biblioteca VCL. Caso o componente possua

mapeamento, um objeto descendente de `TMappedComponent` será retornado, caso contrário, uma referência nula, representada pela palavra-chave `nil` na linguagem Pascal, é retornada.

Na sequência da consulta ao mapa, o método `Translate` verifica o tipo retornado pelo método `Find` para decidir o tipo de tradução que será adotada de acordo com a categoria de tradução configurada para o componente. O contexto do método `Translate` da classe `TComponentTranslator` é apresentado no Quadro 28.

```
function TComponentTranslator.Translate(const VCLComponent: TVCLComponent):
   TWidget;
var
   MappedComponent: TMappedComponent;
begin
   MappedComponent := FMap.Find(VCLComponent.VCLClassName);
   if MappedComponent <> nil then begin
      if MappedComponent is TSimpleMapped then begin
         Result := TranslateSimple(TSimpleMapped(MappedComponent));
      end
      else
      if MappedComponent is TComposedMapped then begin
         Result := TranslateComposed(TComposedMapped(MappedComponent));
      end
      else
      if MappedComponent is TSpecialMapped then begin
         Result := TranslateSpecial(TSpecialMapped(MappedComponent),
                   VCLComponent);
      end;
   end
   else
      Result := nil;
end;
```

Quadro 28 - Conteúdo do método `Translate` da classe `TComponentTranslator`

O método `TranslateSimple`, responsável pela tradução de componentes da categoria simples, recebe como parâmetro um objeto do tipo `TSimpleMapped`, no qual estão armazenadas as informações necessárias para criação de um objeto do tipo `TWidget` equivalente ao componente VCL a traduzir. O Quadro 29 apresenta o conteúdo do método, com destaque para a linha de código responsável pela tradução do componente.

```
function TComponentTranslator.TranslateSimple(const Mapped: TSimpleMapped):
   TWidget;
begin
   Result := TWidget.Create(Mapped.MappedWidget.Name,
             MakeNewId(Mapped.MappedWidget.Name));
   AssignProperties(Mapped.MappedWidget, Result);
end;
```

Quadro 29 - Código fonte do método `TranslateSimple` da classe `TComponentTranslator`

Na linha destacada no Quadro 29, cria-se uma nova instância da classe `TWidget` que representa um elemento gráfica na biblioteca `Libglade`. A nome da classe que representa o elemento na biblioteca GTK+ é recuperada do objeto que mapeou o componente VCL.

O método `TranslatePropertied`, responsável pela tradução de componentes da categoria de mapeamento condicionado à propriedade, recebe como parâmetro uma instância da classe `TPropertiedMapped`, resgatada à classe `TComponentMap` por ocasião da execução do método `Translate`. A execução inicia percorrendo a lista de possíveis elementos gráficos GTK+ para os quais o componente VCL pode ser traduzido. Esta lista está armazenada no atributo `Widgets` do objeto `TPropertiedMapped` enviado como argumento à classe. O Quadro 30 isola trecho do código do método `TranslateProperties` que percorre a lista dos elementos de possível tradução.

```
function TComponentTranslator.TranslatePropertied(const Mapped:
   TPropertiedMapped; const VCLComponent: TVCLComponent): TWidget;
var
   I, J: Integer;
   Translatable: Boolean;
   MappedWidget: TConditionedWidget;
   Condition: TPropertyCondition;
   Equality: TEqualityCondition;
   VCLProperty: TVCLProperty;
begin
   Result := nil;
   for I := 0 to Mapped.Widgets.Count - 1 do
   begin
      Translatable := True;
      MappedWidget := Mapped.Widgets.Items[I];
{...}
```

Quadro 30 - Trecho inicial do código do método `TranslatePropertied`

Para cada objeto encontrado na lista, o método verifica se o componente VCL atende às condições exigidas para que seja traduzido. As condições estão armazenadas no atributo `Conditions` do objeto `TMappedWidget` da iteração corrente. Uma nova instrução `for` é inicializada para que cada condição seja avaliada no componente VCL. Como já mencionado na especificação da ferramenta, há dois tipos de condições; igualdade e existência, representadas pelas classes `TEqualityCondition` e `TExistisCondition` respectivamente. O método `TranslatePropertied` verifica o tipo da condição para avaliar o componente VCL. Para condição de igualdade o valor armazenado no objeto `TEqualityCondition` é comparado ao valor da propriedade equivalente no componente VCL. Para a condição de existência apenas verifica se a propriedade indicada pelo atributo `PropertyName` da classe `TExistsCondition` está presente no componente VCL. Para a tradução ser realizada, todas as condições devem ser atendidas. O trecho de código responsável pela avaliação das condições no componente VCL é apresentado no Quadro 31.

```
{...}
for J := 0 to MappedWidget.Conditions.Count - 1 do begin
   Condition := MappedWidget.Conditions.Items[J];
   if Condition is TEqualityCondition then begin
      Equality := TEqualityCondition(Condition);
      VCLProperty := VCLComponent.FindProperty(Equality.PropertyName);
      if VCLProperty = nil then
         Translatable := False
      else
         Translatable := VCLProperty.Equals(Equality.PropertyValue);
   end
   else begin
      VCLProperty := VCLComponent.FindProperty(Condition.PropertyName);
      Translatable := VCLProperty <> nil;
   end;
   if not Translatable then
      Break;
end;

if Translatable then begin
   Result := TWidget.Create(MappedWidget.Name, MakeNewId(MappedWidget.Name));
   AssignProperties(MappedWidget, Result);
   Break;
end;
{...}
```

Quadro 31 - Trecho de código responsável por avaliar as condições para tradução

Os componentes cujo mapeamento foi categorizado como especial, são traduzidos através da chamada ao método `TranslateSpecial`. Este método não executa a tradução propriamente dita, ele verifica se uma classe tradutora foi informada para o componente. Se a classe tradutora existe, o método `Translate` da classe tradutora é acionado, enviando como parâmetro o objeto `TVCLComponent` que deve ser traduzido. As classes tradutoras são descendentes da classe `TAbstractSpecialTranslator`, que implementa a interface `ISpecialTranslator`. No Quadro 32 é apresentado o código fonte do método `TranslateSpecial`.

```
function TComponentTranslator.TranslateSpecial(const Mapped: TSpecialMapped;
   const VCLComponent: TVCLComponent): TWidget;
begin
   if Mapped.Translator = nil then
      Result := nil
   else
      Result := Mapped.Translator.Translate(VCLComponent);
end;
```

Quadro 32 - Código fonte do método `TranslateSpecial` da classe `TComponentTranslator`

Após a tradução do componente VCL, o controle é retornado ao método `PerformTranslate` da classe `TDFMTranslator`. Este, por sua vez, verifica se a tradução foi efetuada com sucesso. Se a tradução ocorreu com sucesso, os próximos passos envolvem traduzir as propriedades, traduzir os eventos e posicionar o componente na interface gráfica.

3.6.2.5 Tradução das propriedades

A tradução das propriedades é realizada através do método `TranslateProperties`, para o qual é enviado como parâmetro o objeto `TVCLComponent` cujas propriedades deseja-se traduzir. O retorno do método é uma lista de objetos do tipo `TProperty`, do pacote `Glade`. O método `TranslateProperties` consulta o mapa de componentes, cuja instância é referenciada pelo atributo `FPropertyMap`, para verificar se a classe do componente VCL possui propriedades mapeadas.

O processo inicia com uma chamada ao método `FindClass` da classe `TPropertyMap`, enviando como parâmetro o nome da classe do componente na biblioteca VCL. O retorno é um objeto do tipo `TMappedClass` que representa um componente VCL cujo propriedades foram mapeadas e devidamente carregadas para a aplicação. Se a referência para o objeto `TMappedClass` é válida, o método percorrerá todas as propriedades presentes no parâmetro `VCLComponent` e enviá-los-á para a classe `TPropertyMap` através do método `AssignProperty` por ela disponibilizado. O método `AssignProperty` recebe dois parâmetro, sendo o primeiro a propriedade do componente VCL e o segundo o nome da classe referente ao componente, e realiza o preenchimento dos objetos do tipo `TMappedProperty` que compõem os grupos de tradução, representados pela classe `TMappedPropertyGroup`.

Uma vez que todas as propriedades e seus valores foram preenchidos no mapa de propriedades, a tradução pode ser realizada. O método `TranslateProperties` percorre a lista de grupos de tradução do componente VCL mapeado para realizar chamada ao método `Translate` do seu tradutor. O conteúdo do método `TranslateProperties` é apresentado no Quadro 33.

```
function TDFMTranslator.TranslateProperties(const VCLComponent: TVCLComponent):
   TPropertyList;
var
   I, J: Integer;
   MappedClass: TMappedClass;
   Properties: TPropertyList;
begin
   Result := TPropertyList.Create;
   MappedClass := FPropertyMap.FindClass(VCLComponent.VCLClassName);
   if MappedClass <> nil then begin
      MappedClass.Clear;
      for I := 0 to VCLComponent.Properties.Count - 1 do
         FPropertyMap.AssignProperty(VCLComponent.Properties.Items[I],
         VCLComponent.VCLClassName);

      for I := 0 to MappedClass.Groups.Count - 1 do
         if MappedClass.Groups.Items[I].Translator <> nil then begin
            Properties := MappedClass.Groups.Items[I].Translator.Translate;
            if Properties <> nil then
               for J := 0 to Properties.Count - 1 do
                  Result.Add(Properties.Items[J]);
         end;
   end;
end;
```

Quadro 33 - Código fonte do método `TranslateProperties` da classe `TDFMTranslator`

Traduzidas as propriedades, o controle retorna ao método `PerformTranslation`, que atribuirá as propriedades traduzidas ao componente traduzido. O trecho de código responsável por esta operação é apresentado no Quadro 34.

```
function TDFMTranslator.PerformTranslation(const VCLComponent: TVCLComponent):
   TWidget;
var
   Properties: TPropertyList;
   I: Integer;
begin
   {...}
   Properties := TranslateProperties(VCLComponent);
      for I := 0 to Properties.Count - 1 do
         Result.Properties.Add(Properties.Items[I]);
   {...}
end;
```

Quadro 34 - Trecho do código do método `PerformTranslate` responsável atribuir as propriedades traduzidas

3.6.2.6 Tradução dos eventos

A definição de eventos de componentes VCL em arquivos `.dfm` possuem notação similar à definição de propriedades. Tal como as propriedades, os eventos são armazenados em objetos do tipo `TVCLProperty`, do pacote `VCL Abstract`, de forma que não é possível diferenciá-los das propriedades durante o processamento do arquivo `.dfm`. Portanto, depois de realizada a tradução das propriedades, o método `PerformTranslation` aciona o método `TranslateEvents`, enviando como parâmetro o objeto do tipo `TVCLComponent` que referência

o componente VCL que está sendo traduzido.

O método `TranslateEvents` percorrerá todos os elementos do tipo `TVCLProperty` contidos pelo parâmetro `VCLComponent`. Somente é possível definir se um objeto do tipo `TVCLProperty` armazena as informações de um evento carregado de um arquivo `.dfm` a partir do momento que sabe-se que este está mapeado ou não pelo mapa de eventos. Com auxílio da classe `TEventTranslator`, instanciada no atributo `FEventTranslator`, efetua-se uma chamada ao método `IsMapped` para verificar se o objeto `TVCLProperty` em questão é ou não um evento.

Se o resultado da chamada a `IsMapped` é `True`, significa que o objeto `TVCLProperty` é um evento mapeado e, portanto, sua tradução é realizada acionando o método `Translate` da classe `TEventTranslator`, que recebe como parâmetros o nome da classe VCL correspondente ao componente, o nome do evento e o seu valor. O valor indica o nome do método que foi definido para manipular o evento na aplicação Delphi. O Quadro 35 apresenta o código fonte do método `TranslateEvents`.

```
function TDFMTranslator.TranslateEvents(const VCLComponent: TVCLComponent):
   TSignalList;
var
   I: Integer;
   VCLProperty: TVCLProperty;
   Value: string;
   Signal: TSignal;
begin
   Result := TSignalList.Create;

   for I := 0 to VCLComponent.Properties.Count - 1 do begin
      VCLProperty := VCLComponent.Properties.Items[I];
      if FEventTranslator.IsMapped(VCLComponent.VCLClassName,VCLProperty.Name)
      then
         if VCLProperty.Value.CopyTo(Value) then begin
            Signal := FEventTranslator.Translate(VCLComponent.VCLClassName,
                      VCLProperty.Name, Value);
            Result.Add(Signal);
         end;
   end;
end;
```

Quadro 35 - Código fonte do método `TranslateEvents` da classe `TDFMTranslator`

Depois de traduzidos, os eventos são retornados ao método `PerformTranslation` em uma lista de objetos do tipo `TSignal`. Os objetos da lista são percorridos e adicionados ao objeto `TWidget` traduzido de `TVCLComponent`. O Quadro 36 isola parte do código fonte do método `PerformTranslation` responsável por atribuir os sinais GTK+ traduzidos para o objeto `TWidget`.

```
function TDFMTranslator.PerformTranslation(const VCLComponent: TVCLComponent):
   TWidget;
var
   Signals: TSignalList;
   I: Integer;
begin
   {...}
   if Result <> nil then begin
      {...}
      Signals := TranslateEvents(VCLComponent);
      for I := 0 to Signals.Count - 1 do
         Result.Signals.Add(Signals.Items[I]);
      {...}
   end;
   {...}
end;
```

Quadro 36 - Trecho de código do método `PerformTranslation` que atribui os sinais traduzidos

3.6.2.7 Posicionamento dos elementos gráficos

Depois de finalizadas todas as etapas de tradução do componente, o método `PerformTranslation` realiza o processo de posicionamento dos elementos gráficos da interface. A biblioteca VCL do Delphi fornece diferentes opções para realizar o posicionamento e alinhamento dos componentes na interface gráfica. O posicionamento do componente é sempre relativo ao componente no qual ele está contido. O tipo de alinhamento aplicado ao componente é armazenado na propriedade `Align`. Os possíveis valores para `Align` e sua descrição são apresentados no Quadro 37.

valor	descrição
`alBottom`	Posiciona o componente na parte inferior do componente que o contém.
`alClient`	O componente ocupa todo o espaço não ocupado no componente que o contém.
`alCustom`	Permite ao desenvolvedor customizar o alinhamento do componente.
`alLeft`	Posiciona o componente à esquerda dentro do componente que o contém.
`alNone`	Nenhum alinhamento é aplicado ao componente em relação ao componente que o contém.
`alRight`	Posiciona o componente à direita dentro do componente que o contém.
`alTop`	Posiciona o componente na parte superior do componente que o contém.

Quadro 37 - Possibilidades de alinhamento de componentes VCL

Além da espécie de alinhamento, outras propriedades também influenciam o posicionamento do componente em relação ao componente que o contém. Estas propriedades e sua finalidade são listadas no Quadro 38.

nome	descrição
`Left`	Posição horizontal inicial do componente em relação ao componente que o contém.
`Top`	Posição vertical inicial do componente em relação ao componente que o contém.
`Width`	Comprimento do componente.
`Height`	Altura do componente.

Quadro 38 - Propriedades do componente VCL que condicionam seu posicionamento

Para os alinhamentos do tipo `alTop` e `alBottom`, o comprimento do componente, indicado na propriedade `Width`, está condicionado ao componente que o contém, visto que o componente com este tipo de alinhamento ocupará todo o espaço horizontal disponível. Já para os alinhamentos `alLeft` e `alRight` a altura, indicada pela propriedade `Height`, está condicionada à altura disponibilizada pelo componente que o contém.

Na biblioteca GTK+ o alinhamento e posicionamento dos componentes são realizados de forma distinta da VCL. A GTK+ disponibiliza componentes específicos para realização desta tarefa, denominados *containers* e que descendem da classe `GtkContainer`. Para reproduzir o alinhamento e posicionamento do Delphi na `Libglade`, foram utilizados os *containers* `GtkFixed`, `GtkHBox` e `GtkVBox`. Ao passo que na VCL o componente possui a informação sobre o seu alinhamento em relação ao componente que o contém, na GTK+ o *container* é responsável pelo alinhamento dos elementos nele contidos. As informações sobre posicionamento, no entanto, estão armazenadas no elemento que está contido.

Na biblioteca VCL os alinhamentos verticais; `alBottom` e `alTop`, têm precedência maior, seguidos pelos alinhamentos horizontais; `alLeft` e `alRight` e os alinhamentos `alNone` e `alClient` têm precedência inferior. Desta forma, para realizar o alinhamento, o método `PerformTranslation` verifica se o componente que está sendo traduzido possui algum componente contido cujo alinhamento é vertical. Caso sim, criar-se-á um novo elemento GTK+ do tipo `GtkVBox`, uma caixa para armazenamento de elementos alinhados verticalmente. Contudo, esta nova caixa somente será criada se o próprio componente traduzido não é do tipo `GtkVBox`. No Quadro 39 é apresentado o trecho de código responsável pela verificação do alinhamento e criação do *container* `GtkVBox`.

```
if (alTop in ChildsAlign) or (alBottom in  ChildsAlign) then begin
   if Result.Name = 'GtkVBox' then
      VBox := Result
   else begin
      VBox := TWidget.Create('GtkVBox', MakeNewId('GtkVBox'));
      VBox.Properties.Add(TProperty.Create('orientation',
      TStringValue.Create('vertical')));
      Result.Childs.Add(VBox);
   end;
   VBox.Properties.Add(TProperty.Create('visible', TBooleanValue.Create(True)));
end
else
   VBox := nil;
```

Quadro 39 - Trecho do método `PerformTranslation` responsável pela verificação do alinhamento vertical

Depois de verificar se o componente contém outros componentes alinhados verticalmente, o método `PerformTranslation` verifica a existência de componentes com alinhamento horizontal, ou seja, cujo valor da propriedade `Align` é `alLeft` ou `alRight`. Se

encontrar algum componente com este tipo de alinhamento, criar-se-á um novo *container* da classe `GtkHBox`, que representa uma caixa para armazenamento de elementos alinhados horizontalmente. Da mesma forma como no alinhamento vertical, uma nova caixa horizontal será criada somente se o próprio componente convertido não é do tipo `GtkHBox`. Se o componente traduzido possui componentes com alinhamentos vertical e horizontal, a caixa de alinhamento horizontal é armazenada pela caixa de alinhamento vertical, de forma a simular a precedência de alinhamentos da biblioteca VCL. O Quadro 40 apresenta o trecho de código do método `PerformTranslation` responsável por esta verificação.

```
if (alLeft in ChildsAlign) or (alRight in ChildsAlign) then begin
   if Result.Name = 'GtkHBox' then
      HBox := Result
   else begin
      HBox := TWidget.Create('GtkHBox', MakeNewId('GtkHBox'));
      if VBox = nil then
         Result.Childs.Add(HBox)
      else
         VBox.Childs.Add(HBox);
   end;
   HBox.Properties.Add(TProperty.Create('visible', TBooleanValue.Create(True)));
end
else
   HBox := nil;
```

Quadro 40 - Código responsável pela verificação dos componentes alinhados horizontalmente

Finalizada a verificação da existência de componentes horizontalmente alinhados, verifica-se a existência de componentes sem alinhamento, ou seja, o valor da propriedade `Align` é `alNone`. Para simular este tipo de alinhamento na biblioteca GTK+ usa-se a classe `GtkFixed`. Todo elemento inserido em um *container* `GtkFixed` tem seu posicionamento e dimensões fixas, migradas das propriedades `Left`, `Top`, `Width` e `Height` da especificação da interface gráfica do arquivo `.dfm`. Para simular a precedência de alinhamento da VCL, o *container* `GtkFixed` criado é inserido no *container* anteriormente criado, que pode ser um `GtkHBox` ou `GtkVBox`, se existir; caso contrário é inserido no próprio componente traduzido, se este não é do tipo `GtkFixed`. O trecho de código do método `PerformTranslation` responsável pela verificação da existência de componentes sem alinhamento é apresentado no Quadro 41.

```
if (alNone in ChildsAlign) or (alCustom in  ChildsAlign) then begin
   if Result.Name = 'GtkFixed' then
      Fixed := Result
   else begin
      Fixed := TWidget.Create('GtkFixed', MakeNewId('GtkFixed'));
      if HBox = nil then
         if VBox = nil then
            Result.Childs.Add(Fixed)
         else
            VBox.Childs.Add(Fixed)
      else
         HBox.Childs.Add(Fixed);
   end;
   Fixed.Properties.Add(TProperty.Create('visible', TBooleanValue.Create(True)));
end
else
   Fixed := nil;
```

Quadro 41 - Código do método `PerformTranslation` que verifica componentes sem alinhamento

Finalizados os processos de verificação dos alinhamentos dos componentes filhos, iniciar-se-á o processo de tradução e posicionamento dos componentes filhos. Através de uma instrução `for`, os componentes filhos do componente que está sendo traduzido são percorridos e traduzidos com uma chamada recursiva para `PerformTranslation`. Verifica-se o retorno da chamada para conhecer se a tradução foi realizada com sucesso. O fragmento de código que percorre os componentes filhos e realiza sua tradução é apresentado no Quadro 42.

```
for I := 0 to VCLComponent.Childs.Count - 1 do begin
   // Guarda referência para o componente filho
   VCLChild := VCLComponent.Childs.Items[I];

   // Realiza a tradução do componente filho
   Child := PerformTranslation(VCLChild);

   // Se a tradução foi bem sucedida realiza o posicionamento
   if Child <> nil then
   {...}
```

Quadro 42 - Código que percorre os componentes contidos e realiza sua tradução

Na sequência, se a tradução do componente filho foi realizada com sucesso, seu alinhamento em relação ao *container* é verificado para definir onde ficará armazenado. Componentes cujo alinhamento é do tipo `alTop` ou `alBottom` são armazenados na caixa `GtkVBox` e a sua altura é migrada da propriedade `Height` da VCL para a propriedade `height_request` na `Libglade`. Componentes com alinhamento horizontal, `alLeft` ou `alRight`, são armazenados na caixa `GtkHBox` e o seu comprimento é migrado da propriedade `Width` da VCL para a propriedade `width_request` na `Libglade`. Os componentes sem alinhamento, ou alinhamento `alNone`, são armazenados no *container* `GtkFixed` e suas coordenadas de posicionamento, definidas nas propriedades `Left` e `Top` da biblioteca VCL, são migradas para as propriedade `x` e `y` da `Libglade` e definidas no objeto `TPacket` do objeto `TWidget` que representa o componente traduzido. Os componentes definidos com alinhamento

alClient no arquivo .dfm, são armazenados no *container* de menor precedência criado, ocupando todo o espaço restante disponibilizado pelo *container*.

3.6.2.8 Geração do arquivo .glade

Finalizando todas as suas instruções de tradução, iniciadas com uma chamado ao método Translate, a classe TDFMTranslate devolve o controle do processo à classe TProject, responsável por gerenciar todas as etapas envolvidas, desde o carregamento do arquivo .dfm até a geração do arquivo .glade correspondente. Esta última etapa é de responsabilidade da classe TLibgladeWriter. Com auxílio das outras classes do pacote Glade Transform, esta classe é capaz de interpretar um objeto TGladeInterface e transformá-lo em um arquivo XML segundo as definições da biblioteca Liblglade para uma interface gráfica GTK+.

O processo é iniciado com uma chamada ao método Write da classe TLibgladeWriter, para o qual são passados como parâmetros o objeto TGladeInterface retornado pela classe TDFMTranslate como resultado da chamada à Translate e o diretório no qual o arquivo .glade criado deve ser armazenado.

O processo de transformação do objeto TGladeInterface para o formato XML é desencadeado com uma chamada ao método AsDOMNode da classe TTransformerGlade. Depois de transformar o objeto TGladeInterface para um formato XML legível à Libglade, os objetos do tipo TWidget nele contidos são percorridos e transformados com auxílio das classes TTransformerContainer e TTransformerWidget. Para cada objeto do tipo TWidget suas propriedades, sinais e empacotamento são transformados com auxílio das classes TTransformerProperty, TTransformerSignal e TTransformerPacket, respectivamente.

A transformação dos objetos em conteúdo XML é realizado com auxílio das classes e funções para manipulação de XML disponibilizadas pelo compilador FreePascal. Para criação de *tags* XML é usado a classe TDOMNode; para criação de atributos, a classe TDOMAttr e para criação de valores de texto para *tags*, a classe TDOMText. O código fonte do método AsDOMNode, da classe TTransformerProperty, utiliza os recursos mencionados e seu conteúdo é apresentado no Quadro 43.

```
function TTransformerProperty.AsDOMNode(const AOwner: TDOMDocument): TDOMNode;
var
   NameAttr, TransAttr: TDOMAttr;
   ValueNode: TDOMText;
begin
   Result := AOwner.CreateElement('property');
   NameAttr := AOwner.CreateAttribute('name');
   NameAttr.Value := FProperty.Name;
   Result.Attributes.SetNamedItem(NameAttr);
   if FProperty.Translatable then begin
      TransAttr := AOwner.CreateAttribute('translatable');
      TransAttr.Value := 'yes';
      Result.Attributes.SetNamedItem(TransAttr);
   end;
   ValueNode := AOwner.CreateTextNode(FProperty.Value.AsString);
   Result.AppendChild(ValueNode);
end;
```

Quadro 43 - Código fonte do método `AsDOMNode` da classe `TTransformerProperty`

3.6.3 Operacionalidade da implementação

Esta seção fala sobre a operacionalidade da ferramenta, apresentando imagens dos elementos da sua interface gráfica e suas funcionalidades. Na Figura 16 é apresentada a tela principal da ferramenta DelphiToGTK+.

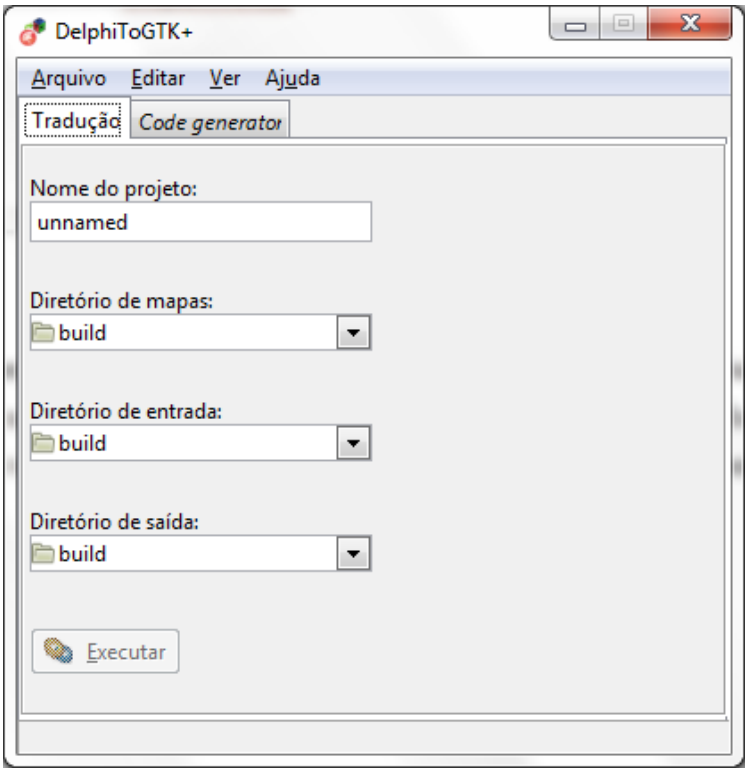

Figura 16 - Janela principal da ferramenta DelphiToGTK+

O campo denominado `Nome do projeto` foi elaborado com o objetivo de permitir ao usuário informar o nome para um projeto de tradução e salvar as configurações relacionadas ao projeto de forma a carregá-las posteriormente. Contudo, tal funcionalidade não foi implementada nesta versão da ferramenta por tratar-se de um requisito não levantado no trabalho e por não interferir diretamente na proposta do mesmo.

No campo `Diretório de mapas` deve-se informar o diretório onde estão localizados os arquivos `componentes.xml`, `properties.xml` e `events.xml`. Na Figura 17 é apresentado um exemplo de seleção de diretório de mapas.

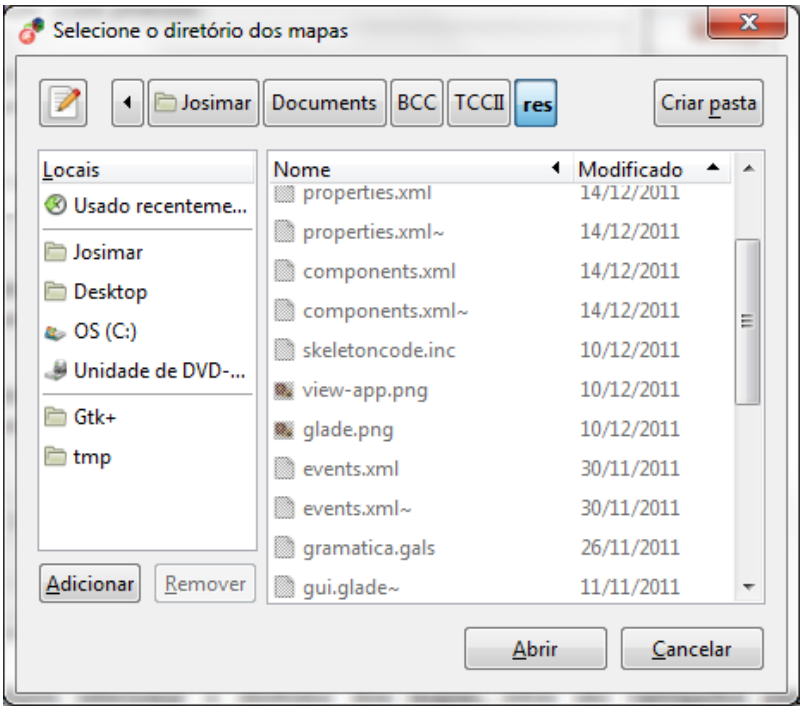

Figura 17 - Seleção do diretório de localização dos mapas

Após selecionar o diretório dos mapas, estes são carregados para a aplicação. Contudo, é necessário informar quais arquivos deseja-se traduzir através do campo `Diretório de entrada`. Este campo permite informar um diretório no qual estão localizados os arquivos `.dfm` que serão traduzidos. A Figura 18 apresenta a janela para seleção do diretório de entrada.

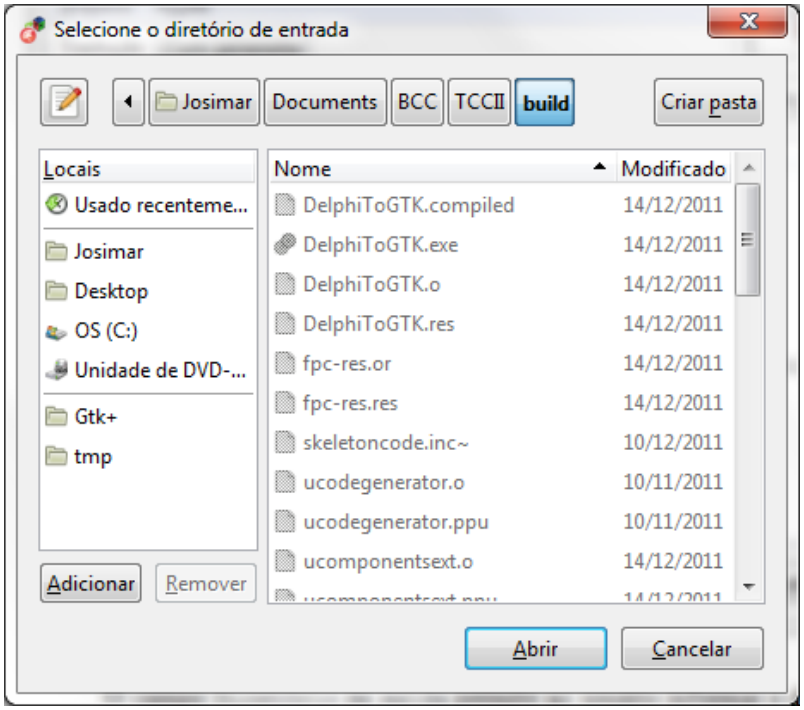

Figura 18 - Seleção do diretório de entrada

O campo `Diretório de saída` permite ao usuário informar o diretório onde deseja que os arquivos `.glade`, provenientes do processo de tradução, sejam armazenados. A Figura 19 apresenta um exemplo de seleção de diretório de saída.

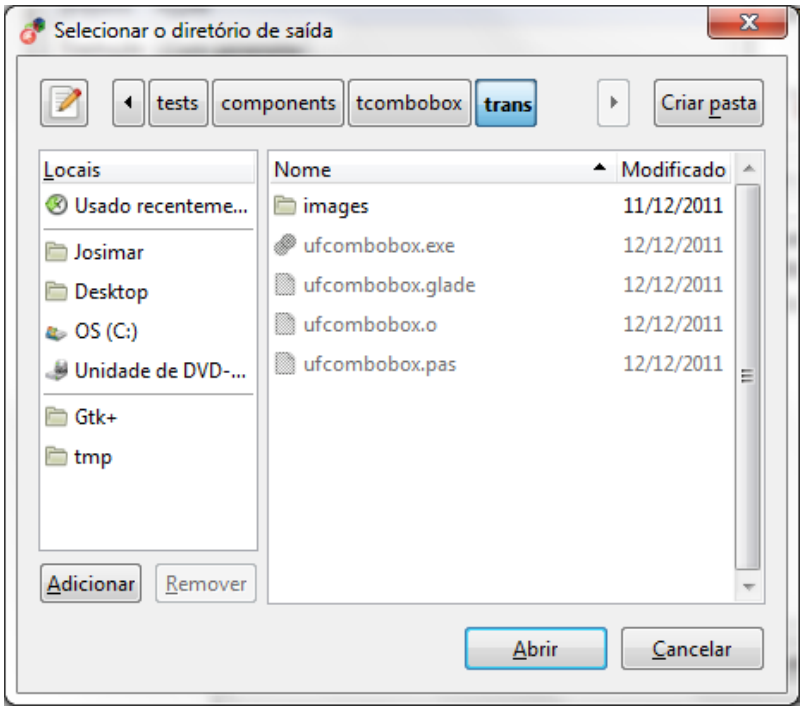

Figura 19 - Seleção do diretório de saída

Uma vez que todos os diretórios foram selecionados, a aplicação está apta a realizar o processo de tradução dos arquivos `.dfm`. Este processo é acionado pelo botão `Executar`, destacado na Figura 20.

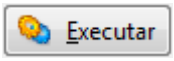

Figura 20 - Botão para executar a tradução

Finalizado o processo de tradução, os arquivos `.glade` estarão armazenados no diretório de saída. O arquivo traduzido recebe o nome do arquivo `.dfm` correspondente, diferenciando-se apenas por sua extensão.

3.7 RESULTADOS E DISCUSSÃO

A ferramenta produzida como resultado deste trabalho atende aos requisitos propostos, apresentando uma interface gráfica de fácil usabilidade e compreensão. Realiza a conversão dos componentes visuais mais comumente utilizados no desenvolvimento de aplicações com interface gráfica do usuário no Delphi.

A ferramenta DelphiToGTK+ foi totalmente desenvolvida sobre software livre e portável, o que permite que ela seja migrada para outras plataformas que suportam a versão da GTK+ usada no seu desenvolvimento. A adoção de padrões de tradução através de arquivos XML permite que novos componentes sejam adicionados aos mapas e traduzidos pela ferramenta sem a necessidade de reconstrução do código fonte.

Quanto aos componentes que apresentam um grau de complexidade que não permite o mapeamento, pode-se optar pelo desenvolvimento de uma classe específica para sua tradução sendo necessário, neste caso, recompilar a aplicação.

Foram realizados quatro diferentes testes sobre a ferramenta DelphiToGTK+ com o intuito de detectar possíveis falhas, bem como os resultados obtidos da sua aplicação sobre casos de testes. Os casos de testes incluem as três diferentes categorias de mapeamento de componentes, além da tradução de propriedades e eventos.

A Figura 21 apresenta o primeiro caso de teste, uma interface gráfica desenhada no Delphi. Na sequência (Figura 22), é apresentada a interface gráfica traduzida para a biblioteca GTK+.

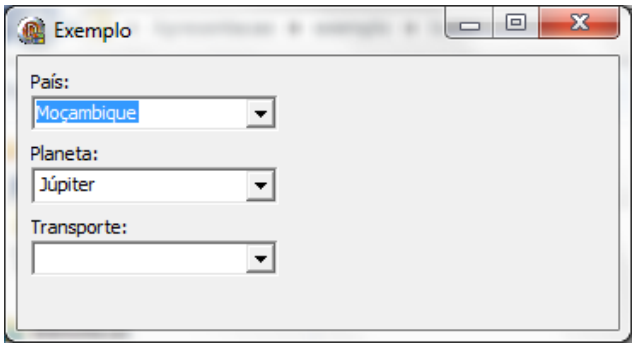

Figura 21 - Interface gráfica - primeiro caso de teste

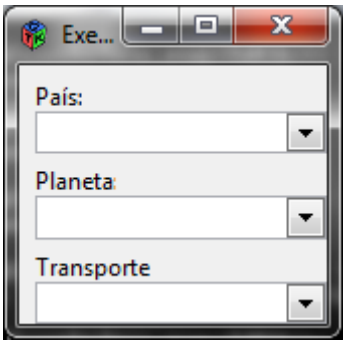

Figura 22 - Interface gráfica traduzida - primeiro caso de teste

Nota-se no primeiro caso de teste que todos os componentes foram corretamente traduzidos para a biblioteca GTK+. Contudo, alguns problemas são notados ao se realizar uma comparação entre as duas figuras:

a) as dimensões das janelas não foram corretamente traduzidas;

b) dois dos três componentes do tipo `TLabel` da VCL não foram corretamente desenhados na GTK+, omitindo parte do seu conteúdo;

c) os valores iniciais das duas primeiras caixas de seleção, componente `TComboBox` da VCL, não foram selecionados nos componentes traduzidos.

No segundo caso de teste, tal como no primeiro, são utilizados componentes de tradução categorizada como simples. A Figura 23 apresenta a interface desenhada no Delphi, ao passo que a Figura 24 apresenta o resultado obtido com a tradução.

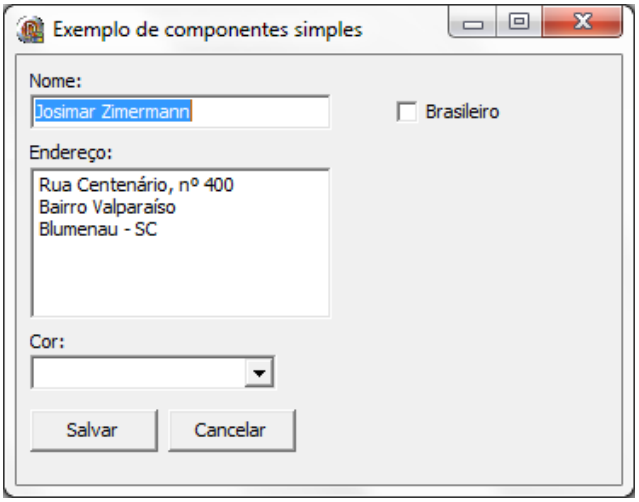

Figura 23 - Interface gráfica - segundo caso de teste

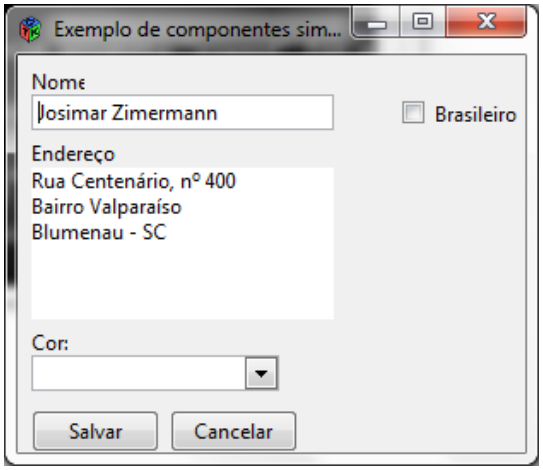

Figura 24 - Interface gráfica traduzida - segundo caso de teste

Neste segundo caso de teste todos os componentes inseridos na interface gráfica desenhada no Delphi foram migrados para a biblioteca GTK+. No entanto, os dois primeiros problemas identificados no primeiro caso de testes, também foram detectados.

Na sequência é apresentado o terceiro caso de teste aplicado sobre a ferramenta DelphiToGTK+. Este caso de teste focaliza na tradução de componentes cuja tradução é categorizada como condicionada à propriedade. A Figura 25 apresenta a interface gráfica de origem desenhada com auxílio do Delphi, ao passo que a interface traduzida para a GTK+ através da utilização da ferramenta DelphiToGTK+ é apresentada na Figura 26. Pode-se observar que os componentes presentes na interface gráfica original foram devidamente traduzidos para a biblioteca GTK+, incluindo seu posicionamento dentro da interface como algumas de suas propriedades, exceto um dos componentes `TLabel` que não foi corretamente desenhado na interface GTK+, pois parte do seu conteúdo foi omitido.

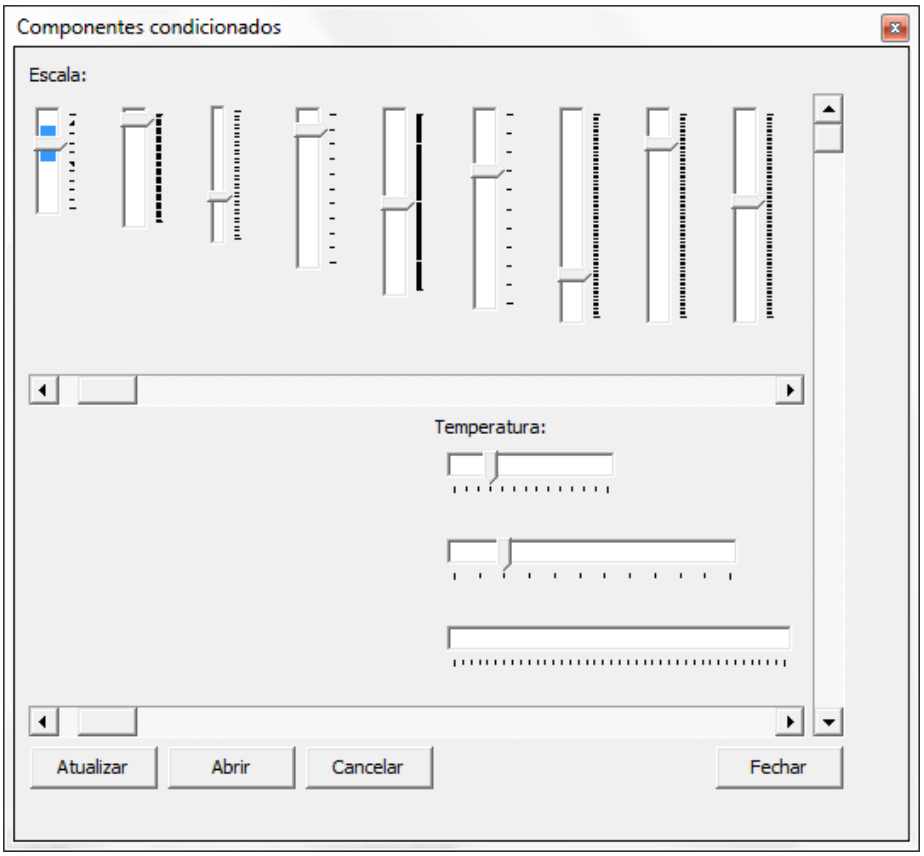

Figura 25 - Interface gráfica - terceiro caso de teste

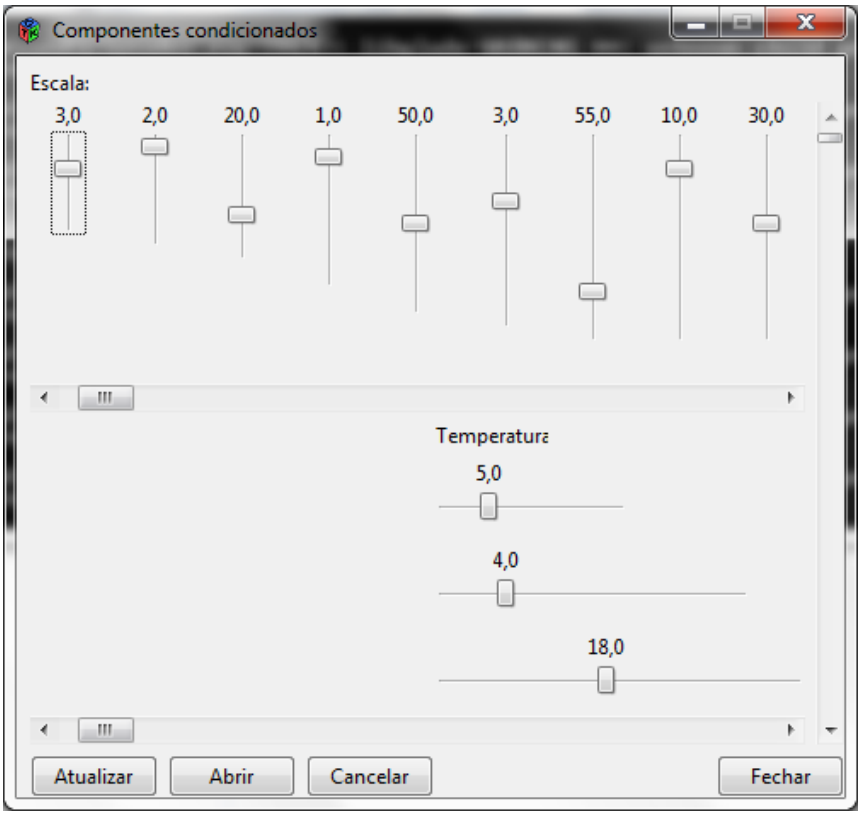

Figura 26 - Interface gráfica traduzida - terceiro caso de teste

No quarto e último caso de teste é aplicada a tradução sobre os dois componentes especiais suportados pela ferramenta: `TGroupBox` e `TPanel`. A Figura 27 apresenta a interface gráfica originalmente desenhada no Delphi e sua versão traduzida para a GTK+ é apresentada na Figura 28.

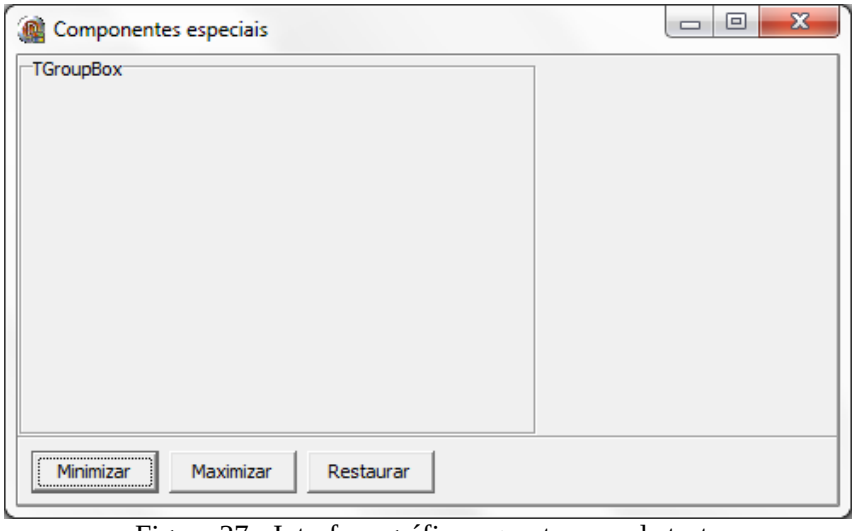

Figura 27 - Interface gráfica - quarto caso de teste

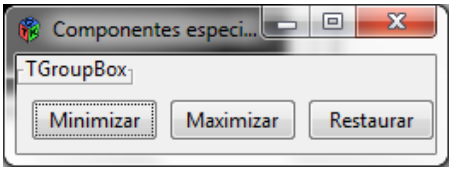

Figura 28 - Interface gráfica traduzida - quarto caso de teste

No quarto caso de teste aplicado sobre a ferramenta DelphiToGTK+ todos os

componentes presentes na interface gráfica original foram traduzidas para a biblioteca GTK+. Contudo, verificou-se que as dimensões da janela e do componente `TGroupBox` não foram corretamente traduzidas.

O Quadro 44 apresenta características relevantes à ferramenta DelphiToGTK+ e às ferramentas apresentadas como trabalho correlatos, traçando um comparativo entre elas.

característica	**DelphiToWeb**	**Extensão da ferramenta Delphi2Java-II**	**ScriptCase**	**DelphiToGTK+**
linguagem de programação	Java	Java	PHP e Javascript	Pascal/Object Pascal
flexibilidade para conversão	Não	Não	-	Sim
tradução para tecnologia portável	Sim	Sim	-	Sim
suporte para acesso a banco de dados	Não	Sim	Sim	Não
interface traduzida pode ser migrada para diferentes linguagens de programação	Não	Não	-	Sim

Quadro 44 - Quadro comparativo entre as ferramentas

Nota-se no Quadro 44 que a ferramenta DelphiToGTK+ apresenta características desejáveis a uma ferramenta de conversão. Dentre tais características destacam-se a flexibilidade para conversão e a possibilidade de portar a interface traduzida para diferentes linguagens de programação. Embora o resultado de processamento dos trabalhos correlatos sejam tecnologias portáveis entre diferentes plataformas, ainda assim limita-se a uma única linguagem de programação, o que impossibilita ao usuário a adoção de outra linguagem com a qual esteja familiarizado. Tal limitação está ausente na ferramenta DelphiToGTK+, visto que o resultado do processo de conversão é um arquivo portável e legível a biblioteca gráfica GTK+, que fornece suporte para várias linguagens de programação, dentre elas o Java, alvo de tradução das ferramentas DelphiToWeb e Delphi2Java-II.

Destaca-se também no trabalho a possibilidade do usuário reaproveitar grande porção do código fonte da aplicação traduzida. Isto se dá pelo fato da biblioteca GTK+ fornecer suporte as linguagens de programação Pascal e Object Pascal, que fornece sintaxe idêntica à linguagem de aplicativos criados no Delphi.

4 CONCLUSÕES

Toda aplicação construída para o público leigo à área de computação faz-se necessária a construção de uma interface gráfica do usuário de forma a facilitar a utilização e compreensão do aplicativo. Em vista de tal necessidade, surgiram bibliotecas para desenvolvimento de interfaces gráficas para as mais diversas linguagens de programação. Contudo, geralmente tais bibliotecas são compatíveis somente à linguagem ou ambiente de desenvolvimento para o qual foi construída, dificultando a migração do software para outras linguagens de programação.

Em vista da crescente necessidade de compatibilizar seus aplicativos para diferentes plataformas, os desenvolvedores têm buscado soluções em linguagens portáveis como o Java. Contudo, a migração da interface gráfica constitui uma etapa que toma tempo considerável no processo de conversão. Em face de tais argumentos, além de automatizar o processo de migração de interfaces gráficas, a ferramenta DelphiToGTK+ a faz para o formato portável XML, legível a biblioteca `Libglade`, componente da biblioteca gráfica GTK+.

Por tratar-se de uma biblioteca multi-plataforma e multi-linguagem, as interfaces gráficas definidas nos arquivos `.glade` podem ser carregados por diferentes linguagens de programação em diferentes sistemas operacionais, bastando que a linguagem possua implementado um *binding* para a GTK+.

Outra vantagem na utilização desta biblioteca diz respeito a alterações na interface gráfica. Uma vez que a interface gráfica é carregada dinamicamente do arquivo `.glade`, quaisquer alterações necessárias na interface gráfica dispensa a necessidade de recompilar o código fonte da aplicação.

4.1 PROBLEMAS E LIMITAÇÕES

Embora atenda todos os requisitos propostos, incluindo a tradução de considerável quantidade de componentes da biblioteca VCL para a biblioteca GTK+, a ferramenta desenvolvida como produto deste trabalho apresenta limitações e problemas:

a) nos casos de testes apresentados na seção de resultados e discussão, as dimensões das janelas não foram corretamente traduzidas para a biblioteca GTK+, de forma

que seu tamanho foi condicionado ao tamanho dos componentes inseridos na interaface gráfica;

b) embora a unidade de medida para definição do posicionamento e tamanho do componente na VCL e GTK+ seja o mesmo, a unidade de *pixel*[6], a biblioteca GTK+ apresentou ligeira diferença na pintura dos componentes gráficos, de forma que os problemas tornam-se mais visíveis quando o tamanho do componente a traduzir aumenta;

c) para os componentes cuja tradução não é suportada pela ferramenta, o espaço que seria ocupado por este componente será utilizado por outro componente na sequência que está na mesma hierarquia de armazenamento ou o espaço será suprimido caso um outro componente na mesma hierarquia não seja encontrado.

4.2 EXTENSÕES

Ao longo do desenvolvimento da ferramenta, bem como durante a elaboração da especificação do trabalho, foram identificadas melhorias que podem ser aplicadas à ferramenta, de forma a torná-la mais completa e flexível. Em escala crescente de complexidade, no Quadro 45 são apresentadas as extensões desejáveis à ferramenta DelphiToGTK+.

[6] Menor ponto ou elemento que forma uma imagem digital.

extensão	complexidade
Integrar a ferramenta ao motor de *script* Pascal Script de forma a flexibilizar a construção de rotinas para tradução de componentes complexos, dispensando a necessidade de recompilar a aplicação.	10
Para componentes da VCL que não podem ser traduzidos segundo as especificações da `Libglade`, verificar a possibilidade de gerar código capaz de criar os elementos gráficos em tempo de execução.	10
Criar uma extensão para permitir gerar interfaces gráficas para o formato suportado pela classe `GladeXML` da GTK+.	8
Desenvolver uma interface gráfica para auxiliar na construção de arquivos de mapeamento de componentes, propriedades e eventos.	8
Criar esquemas XML para validação dos mapas de tradução para realizar a validação na aplicação, diminuindo o risco de erros.	8
Gerar código para carregamento da interface gráfica traduzida para diferentes linguagens de programação que suportam a GTK+.	7
Realizar a tradução do componente `TFrame` da biblioteca VCL.	1

Quadro 45 - Sugestões de melhorias e extensões para a ferramenta

REFERÊNCIAS BIBLIOGRÁFICAS

ANTUNES, Jonathan L. **Conhecendo o Delphi**. [S.l.], 2008. Disponível em: <http://www.oficinadanet.com.br/artigo/745/conhecendo_o_delphi>. Acesso em: 25 set. 2010.

EMBARCADERO Delphi. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/CodeGear_Delphi>. Acesso em: 26 set. 2010.

GALE, Tony et al. **GTK+ 2.0 tutorial**. [S.l.], 2010. Disponível em: <http://library.gnome.org/devel/gtk-tutorial/stable/>. Acesso em: 02 set. 2010.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. 2003. 150 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://gals.sourceforge.net>. Acesso em: 26 set. 2010.

GNOME. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/GNOME>. Acesso em: 25 set. 2010.

HARLOW, Eric. **Developing Linux applications with GTK+ and GDK**. Indianapolis: New Riders, 1999.

HAROLD, Elliotte R.; Means, W. S. **XML**: in a Nutshell. 2nd. Sebastopol: O'Reilly, 2002.

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

INTERFACE gráfica do utilizador. In: WIKIPÉDIA, a enciclopédia livre. [S.1.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Interface_gr%C3%A1fica_do_utilizador>. Acesso em: 02 set. 2010.

KDE. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/KDE>. Acesso em: 25 set. 2010.

KRAUSE, Andrew. **Foundations of GTK+ development**. New York: Apress, 2007.

LIRA, Marcelo. **Mantendo a sanidade com o Glade**. [Recife], 2010. Disponível em: <http://www.cin.ufpe.br/~cinlug/wiki/index.php?title=Mantendo_A_Sanidade_Com_O_Glade#Programa.C3.A7.C3.A3o_Orientada_a_Eventos>. Acesso em: 04 set. 2010.

NETMAKE. **ScriptCase**. [Recife], 2010. Disponível em: <http://www.scriptcase.com.br/site/home/home.php>. Acesso em: 20 set. 2010.

NULLSOFT. **Winamp**. [S.l.], 2010. Disponível em: <http://www.winamp.com/>. Acesso em: 25 set. 2010.

PREECE, Jennifer; ROGERS, Yvonne; SHARP, Helen. **Design de interação**: além da interação homem-computador. Porto Alegre: Bookman, 2005.

QT. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Qt>. Acesso em: 25 set. 2010.

SCRIPTCASE. In: WIKIPÉDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/ScriptCase>. Acesso em: 20 set. 2010.

SILVEIRA, Janira. **Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados**. 2006. 81 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SOUZA, Ariana. **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.

THE GLADE PROJECT. **Glade:** a user interface designer. [S.l.], 2009. Disponível em: <http://glade.gnome.org/>. Acesso em: 04 set. 2010.

VISUAL component library. In: WIKIPÉDIA, the free encyclopedia. [S.1.]: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/wiki/Visual_Component_Library>. Acesso em: 05 set. 2010.

XFCE. In: WIKIPÉDIA, a enciclopédia livre. [S. l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Xfce>. Acesso em: 25 set. 2010.

APÊNDICE A – Modelo de código para carregamento da interface

No Quadro 46 é apresentado o modelo de código usado para geração do código fonte para carregamento da interface gráfica traduzida.

```
program $(PROGRAM_NAME);

uses
   gtk2, gdk2, glib, libglade2, sysutils;

$(CALLBACK_IMPL)

exports
   $(CALLBACK_HEADERS)

var
   GladeXML: Pointer;

begin
   gtk_init(@argc, @argv);
   GladeXML := glade_xml_new(PChar('$(UI_DEF_PATH)'), nil, nil);
   glade_xml_signal_autoconnect(GLADE_XML(GladeXML));
   gtk_main;
end.
```

Quadro 46 - Modelo de código para carregamento da interface

ANEXO A – Gramática do `.dfm`

No Quadro 47 é apresentada a gramática, concebida sobre a notação BNF, que define a estrutura sintática de arquivos de definição de interfaces gráficas desenhadas no Delphi.

```
<DFM> ::= <Object> ;

<Object> ::= OBJECT identifier #1 ":" <Type> #2 <PropertyList> <ObjectList> END
#22 ;

<Type> ::= identifier | TForm | TMainMenu | TMenuItem | TPopupMenu | TLabel |
TEdit | TMemo | TButton | TCheckBox | TRadioButton | TListBox | TComboBox |
TScrollBar | TGroupBox | TRadioGroup | TPanel | TBitBtn | TSpeedButton |
TMaskEdit | TStringGrid | TDrawGrid | TImage | TShape | TBevel | TScrollBox |
TCheckListBox | TSplitter | TStaticText | TControlBar | TValueListEditor |
TLabeledEdit | TColorBox | TColorListBox | TCategoryButtons | TButtonGroup |
TDockTabSet | TTabSet | TFlowPanel | TGridPanel | TChart | TTabControl |
TPageControl | TRichEdit | TTrackBar | TProgressBar | TUpDown | THotkey |
TAnimate | TDateTimePicker | TMonthCalendar | TTreeView | TListView |
THeaderControl | TStatusBar | TToolBar | TCoolBar | TPageScroller | TComboBoxEx |
TGauge | TColorGrid | TSpinButton | TSpinEdit | TCalendar ;

<PropertyList> ::= î | <Property> <PropertyList> ;

<ObjectList> ::= î | <Object> <ObjectList> ;

<Property> ::= <PropertyName> "=" #6 <PropertyValue> #21 ;

<PropertyName> ::= identifier #3 <PropertyName_> ;
<PropertyName_> ::= î | "." #4 identifier #5 ;

<PropertyValue> ::= <NumericValue> | string_constant #11 | #12 <PropertyName> |
<BooleanConstant> | "[" #15 <ValueList1> "]" #16 | "(" #17 <ValueList2> ")" #18 |
"{" #19 <ValueList2> "}" #20 | <Collection> ;

<NumericValue> ::= <Signal> <NumericValue_> ;
<NumericValue_> ::= integer_constant #9 | real_constant #10 ;
<Signal> ::= î | "+" #7 | "-" #8 ;

<BooleanConstant> ::= FALSE #13 | TRUE #14 ;

<ValueList1> ::= î | <PropertyValue> <ValueList1_> ;
<ValueList1_> ::= î | "," <PropertyValue> <ValueList1_> ;

<ValueList2> ::= <PropertyValue> <ValueList2_> ;
<ValueList2_> ::= î | <ValueList2> ;

<Collection> ::= "<" <CollectionList> ">" ;

<CollectionList> ::= <CollectionItem> <CollectionList_> ;
<CollectionList_> ::= î | <CollectionList> ;

<CollectionItem> ::= identifier <PropertyList> END ;
```

Fonte: adaptado de Souza (2005, p. 36).

Quadro 47 - Gramática do `.dfm`