UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

GERAÇÃO DE INTERFACES ANDROID A PARTIR DO DELPHI

DOUGLAS JÚLIO REZINI

BLUMENAU
2013

2013/1-12

DOUGLAS JÚLIO REZINI

GERAÇÃO DE INTERFACES ANDROID A PARTIR DO DELPHI

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.

Prof. Marcel Hugo, Mestre - Orientador

BLUMENAU
2013

2013/1-12

GERAÇÃO DE INTERFACES ANDROID A PARTIR DO DELPHI

Por

DOUGLAS JULIO REZINI

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

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Cláudio Ratke, Mestre – FURB
Membro:	Prof. Joyce Martins, Mestre – FURB

Blumenau, 9 de julho de 2013

Dedico este trabalho a minha família, amigos e principalmente aqueles que me incentivaram, apoiaram e ajudaram direta ou indiretamente na realização deste.

AGRADECIMENTOS

A Deus, pela vida que possuo.

À minha família, que sempre esteve presente e desde cedo me incentivou ao estudo e ingresso no curso superior.

Ao Prof. Mauro Marcelo Mattos por ter me dado a ideia que resultou neste trabalho.

Ao meu orientador, Prof. Marcel Hugo, por todo apoio e principalmente por ter acreditado na conclusão deste trabalho.

Não importa como você leve sua vida, sua inteligência o defenderá melhor do que uma espada. Trate de mantê-la afiada!

Patrick Rothfuss

RESUMO

Este trabalho apresenta a elaboração de um conversor de interfaces gráficas ao usuário do ambiente Delphi para a plataforma Android. São apresentados conceitos básicos no desenvolvimento de aplicações Android, assim como o funcionamento dessa estrutura. A ferramenta é desenvolvida em Delphi na forma de um componente, permitindo assim ser utilizada a partir do mesmo. Além da conversão da interface a ferramenta faz a geração de um projeto, a compilação do mesmo e instalação do arquivo compilado em uma máquina virtual ou dispositivo Android. Também são apresentadas as estruturas da ferramenta responsáveis pela entrada, na forma de leitura dos componentes em tempo de execução, e de saída, na geração dos arquivos e criação do projeto. Os componentes Delphi convertidos pela ferramenta são apresentados em uma tabela, onde também são especificados os componentes Android utilizados para a tradução destes.

Palavras-chave: Conversor. Interface gráfica ao usuário. Delphi. Android.

ABSTRACT

This paper presents the development of a graphical user interface converter from Delphi environment to Android platform. Some basic concepts about development of Android applications are presented, as well as the functioning of this structure. The tool is developed as a Delphi component, thereby it allows to be used from the same. Besides the interface conversion, the tool makes the generation of a project, the compilation of it and the installation of the generated compiled file in a virtual machine or Android device. Also are presented the tools structures which are responsible for the input, as runtime components reading, and for the output, as files generation and project creation. The Delphi components converted by the tool are presented in a table, where are also specified of the Android components that are used to translate them.

Key-words: Converter. Graphical user interface. Delphi. Android.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AAPT – *Android Asset Packaging Tool*

ADT – *Android Development Tools*

API – *Application Programming Interface*

CPU – *Central Processing Unit*

DFM – *Delphi ForM*

GPU – *Graphics Processing Unit*

GTK+ – *Gimp Tool Kit*

GPS – *Global Positioning System*

GUI – *Graphical User Interface*

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

iOS – iphone Operating System

OHA – Open Handset Alliance

RAD – *Rapid Application Development*

RF – Requisito Funcional

RGB – *Red Green Blue*

RGBA – *Red Green Blue Alpha*

RNF – Requisito Não Funcional

SDK – *Software Development Kit*

SMS – *Short Message Service*

SO – Sistema Operacional

UI – *User Interface*

URI – *Uniform Resource Identifier*

UML – *Unified Modeling Language*

VCL – *Visual Component Library*

WVGA – *Wide Video Graphic Array*

XML – *eXtensible Markup Language*

XMPP – *eXtensible Messaging and Presence Protocol*

LISTA DE SÍMBOLOS

@ - arroba

% - por cento

+ - mais

SUMÁRIO

1 INTRODUÇÃO

O mercado de dispositivos móveis, principalmente os celulares e *tablets*, está em evidência e conquistando um público cada vez maior, devido à diversidade de aparelhos e sua facilidade de aquisição. Segundo a Anatel (2012), o Brasil encerrou o ano de 2011 com mais de 242 milhões de aparelhos celulares em operação, representando um crescimento de 19,4% em relação ao ano anterior, que contava com cerca de 202 milhões de aparelhos. Ou seja, com o passar dos anos, a procura por este mercado está cada vez maior, tornando-o muito atrativo para todas as áreas.

As vendas mundiais de celulares chegaram a 419 milhões de unidades no segundo trimestre de 2012. Segundo Gartner, as vendas de *smartphones* representam 36,7% do total, com um crescimento de 42,7% no segundo trimestre de 2012 (PETTEY; GOASDUFF, 2012). Este número representa 153,7 milhões de *smartphones* novos chegando à mão dos consumidores. Isso mostra um aumento significativo na demanda de aplicativos comerciais, utilitários e de entretenimento.

Conforme Moretti (2011), observando o número de empresas que estão portando seus sistemas para dispositivos móveis, é possível comprovar o interesse da comunidade mundial em mobilidade. Este comportamento do mercado mundial direcionado para a mobilidade impulsiona, motiva e cria grande necessidade de maior integração do mundo real ao mundo virtual. Sabendo que os produtos estão sendo consumidos no mercado, os desenvolvedores destas tecnologias podem investir em pesquisa e inovar, sem se preocupar se a demanda existe.

O avanço da tecnologia de *hardware* dos dispositivos móveis permite o desenvolvimento de aplicativos cada vez mais robustos e com desempenho próximo ao do *hardware* de computadores de mesa (ZMOGINSKI, 2013). *Smartphones* topo de linha já vem há algum tempo com uma *Graphics Processing Unit* (GPU) dedicada, permitindo criar *softwares* de manipulação de gráficos em duas ou três dimensões sem a dificuldade de restringir a qualidade das imagens para não sobrecarregar o processador do dispositivo.

Um produto deste avanço tecnológico é o sistema operacional Android. Ele ocupa o posto de líder entre os mais vendidos no mundo, aumentando sua vantagem em 20,7% na participação do mercado no segundo trimestre de 2012 com um total de 64,1% do mercado mundial. Já o principal concorrente *iphone Operating System* (iOS) da Apple teve um crescimento de apenas 0,6% com um total de 18,8% do mercado mundial (PETTEY, GOASDUFF, 2012).

O crescimento da plataforma Android gera uma demanda de novos aplicativos e consequentemente atrai novos desenvolvedores para a plataforma. Isto abre espaço para a criação de ferramentas que utilizem outras linguagens e ambientes de desenvolvimento, já consolidados no mercado, para a conversão e desenvolvimento de aplicativos Android. De acordo com Wills (2010), o ambiente de desenvolvimento Delphi se destaca como sendo uma excelente ferramenta *Rapid Application Development* (RAD), devido a sua capacidade de construção de aplicações rápidas e seguras, utilizando os componentes prontos, e dos mecanismos da ferramenta, como o *drag-and-drop* (arrastar e soltar), que permitem a criação rápida e prática de interface visuais.

Neste contexto, a motivação para o desenvolvimento do presente trabalho surgiu a partir da experiência prática no desenvolvimento de aplicativos Delphi e do desejo de criar uma ferramenta capaz de viabilizar o uso do ambiente Delphi como um gerador de interfaces de usuário para projetos Android.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que permita a utilização do ambiente de desenvolvimento do Delphi para criação de interfaces para aplicações Android.

Os objetivos específicos do trabalho são:

a) identificar os componentes de interface do Delphi que possuam comportamento equivalente em interfaces Android;
b) disponibilizar uma aplicação que viabilize a conversão de componentes de interface de um formulário Delphi, gerando uma interface Android equivalente;
c) avaliar o grau de compatibilidade entre o comportamento da interface Delphi e da interface Android gerada.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho é dividido em quatro capítulos, sendo que o segundo capítulo contém a fundamentação teórica necessária para o entendimento do mesmo.

O terceiro capítulo apresenta o desenvolvimento do trabalho utilizando diagramas de casos de uso, de classes e de sequência que definem o componente e o seu funcionamento. Neste capítulo também são apresentados as principais rotinas utilizadas na aplicação, além de resultados e discussões que ocorreram durante o desenvolvimento deste trabalho.

Por fim, o quarto capítulo apresenta as conclusões do trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 descreve-se sobre a migração de código. Na seção 2.2 é apresentada a plataforma Android. Na seção 2.3 descreve-se sobre a programação Java para a plataforma Android. Na seção 2.4 é descrita a estrutura de um projeto Android. Na seção 2.5 é apresentado o ambiente de desenvolvimento Delphi e na seção 2.6 são apresentados os trabalhos correlatos.

2.1 MIGRAÇÃO DE CÓDIGO

Algumas das dificuldades enfrentadas pelas empresas de tecnologia são a manutenção e a migração de sistemas para as novas plataformas de desenvolvimento. Para reduzir custos na evolução de sistemas, usa-se a reengenharia de *software* através da tradução dos mesmos, quando um código fonte em uma linguagem de programação é traduzido para um código fonte em outra linguagem. A tradução do código fonte só é economicamente viável se um tradutor automatizado estiver disponível para fazer a maior parte da tradução (SOMMERVILLE, 2003). Assim, como recurso para a migração de *software*, encontram-se os geradores de código.

A geração de código é uma técnica de construção de código que utiliza determinadas ferramentas para gerar programas. Estas ferramentas podem variar de *scripts* de ajuda muito pequenos a aplicações que transformam modelos abstratos de lógica de negócio em sistemas completos, incluindo os geradores de interface com o usuário e os de acesso ao banco de dados. Não há nenhum estilo específico para as ferramentas de geração de código (DALGARNO, 2006): podem trabalhar na linha de comando ou possuir uma *Graphical User Interface* (GUI); podem gerar código para uma ou mais linguagens; podem gerar código uma vez ou múltiplas vezes e podem ter um número ilimitado de entradas e saídas.

Para a construção de um gerador de código, segundo Herrington (2003, p. 77), podem ser seguidas as seguintes etapas de desenvolvimento: (1ª) determinar qual deve ser a saída do gerador, identificando quais informações devem ser recuperadas da entrada; (2ª) definir qual será a entrada e como a mesma será analisada; (3ª) analisar a entrada, extraindo as informações necessárias para gerar a saída; e (4ª) gerar a saída a partir das informações extraídas da entrada.

2.2 ANDROID

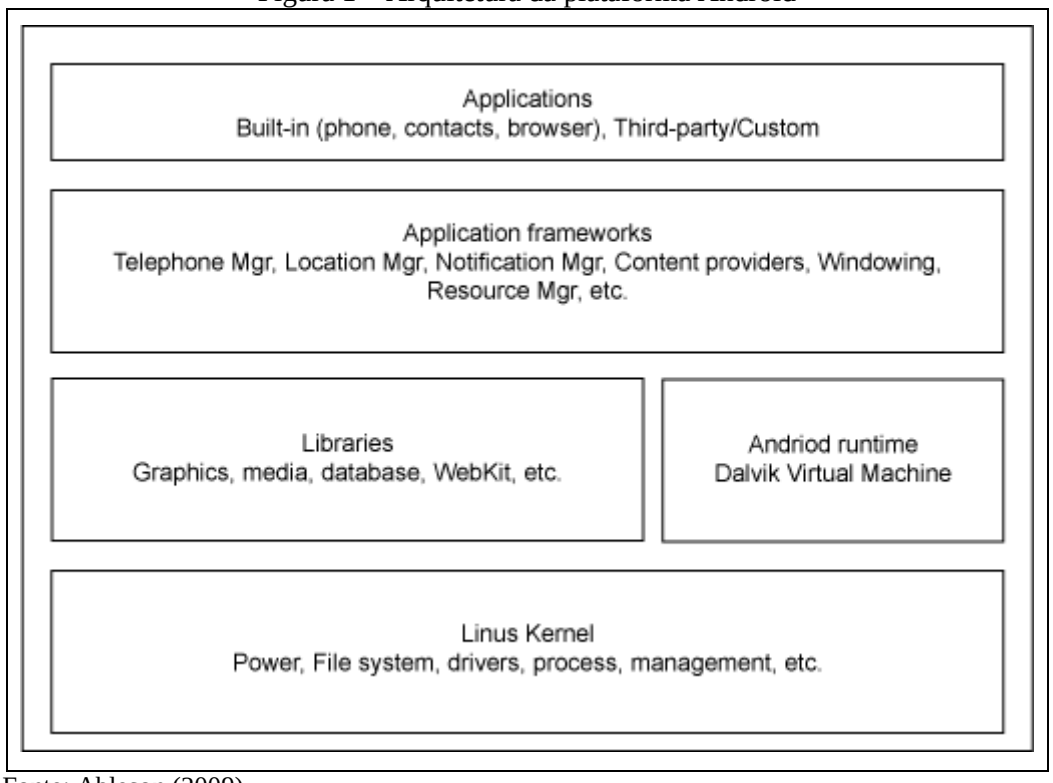
O Android surgiu de uma parceria entre a Google e a *Open Handset Alliance* (OHA), uma aliança onde atuam os principais atores do mercado móvel mundial. Quando se esperava

que a Google fizesse o lançamento de um novo celular, foi lançada também uma plataforma completa para dispositivos móveis, chamada de Android (FREITAS; SCHEMBERGER; VANI, 2009, p. 2).

Android é um conjunto de *softwares* para dispositivos móveis, que inclui o Sistema Operacional (SO), um *middleware* e um conjunto de aplicações. Para criar aplicações para esta plataforma é utilizada a linguagem de programação Java e a aplicação é executada sobre uma máquina virtual denominada *Dalvik*, desenvolvida especificamente para dispositivos com limitação de recursos (AQUINO, 2007, p. 4).

A arquitetura desta plataforma contém quatro camadas: o *kernel* GNU Linux, a máquina virtual e suas bibliotecas de apoio, *frameworks* para aplicações e as aplicações. A Figura 1 ilustra estas camadas.

Figura 1 – Arquitetura da plataforma Android



Fonte: Ableson (2009).

Segundo Aquino (2007, p. 5), os componentes são os seguintes:

a) *kernel* do Linux: o *kernel* do sistema funciona como uma camada de abstração entre o *hardware* e o resto dos *softwares* da plataforma. Esta camada já possui vários recursos necessários para a execução das aplicações, como gerenciamento de memória e de processos, protocolos de rede, módulos de segurança, entre outros módulos do núcleo de infraestrutura;
b) bibliotecas: o Android possui um conjunto de bibliotecas C/C++ que é usado por

vários componentes do sistema, onde as funcionalidades são expostas por um *framework* do Android, que contém bibliotecas gráficas, de banco de dados, encriptação de dados, entre outras;

c) Android *runtime*: todas as aplicações Android executam seu próprio processo, com sua própria instância da máquina virtual, que foi escrita de forma que permita que um dispositivo execute múltiplas máquinas virtuais concorrentes. O *core libraries* contém um conjunto de bibliotecas que fornece a maioria das funcionalidades presentes nas bibliotecas da linguagem de programação Java, que inclui classes de manipulação de arquivos, entrada e saída, entre outros;

d) *framework* de aplicações: nesta camada os desenvolvedores têm acesso completo à mesma *Application Programming Interface* (API) que é utilizada pelas aplicações essenciais da plataforma, onde constam alguns conjuntos de serviços como o *activity manager*, o qual gerencia o ciclo de vida das aplicações; *package manager*, que mantém quais aplicações estão instaladas; *window manager*, que gerencia as janelas das aplicações; *telephony manager*, que são os componentes que permitem acesso aos recursos de telefonia; *content providers*, que permite que as aplicações acessem ou compartilhem dados com outras aplicações; *resource manager*, que fornece acesso a recursos gráficos; *view system*, que é um conjunto de componentes de interface do usuário; *location manager*, que gerencia a localização do dispositivo; *notification manager*, que permite que aplicações exibam alertas na barra de *status*; *XMPP service*, que dá suporte para uso do protocolo *eXtensible Messaging and Presence Protocol* (XMPP);

e) aplicações: o Android fornece um conjunto de aplicações básicas, tais como cliente de *e-mail*, programa *Short Message Service* (SMS), calendário, mapas, navegador, entre outras.

2.3 DESENVOLVIMENTO JAVA PARA ANDROID

As aplicações para a plataforma Android são desenvolvidas utilizando a linguagem de programação Java. A compilação do código fonte é feita pelo Android *Software Development Kit* (SDK), que é responsável por compilar o código fonte, com qualquer informação ou arquivo de recurso utilizado pelo projeto, em um pacote Android: um arquivo com extensão `.apk`. Este arquivo por sua vez é utilizado pelo Sistema Operacional Android para instalar a aplicação (ANDROID DEVELOPERS, 2013).

Para o desenvolvimento de uma aplicação Android é necessário utilizar os Componentes de Aplicação, estes são pontos de entrada para o sistema. Alguns não são pontos de entrada reais para usuários e existem aqueles que dependem um do outro, mas cada um existe como uma entidade própria e desempenha um papel específico. Existem quatro tipos diferentes de componentes de aplicação (ANDROID DEVELOPERS, 2013):

a) atividades (*activities*): uma atividade representa uma tela única de interface com o usuário. Por exemplo, uma aplicação de e-mail possui uma atividade para exibir a lista de e-mails, outra para enviar um e-mail e outra para ler e-mails. Dessa forma embora as atividades trabalhem juntas, cada uma é independente da outra;
b) serviços (*services*): um serviço é um componente sem interface com usuário, este funciona em segundo plano para executar operações de longa duração ou para realizar trabalhos para processos remotos. Por exemplo, um serviço pode tocar músicas em segundo plano enquanto o usuário utiliza outros aplicativos, ou pode buscar dados na rede sem interromper a interação do usuário com a atividade. Outro componente, como uma atividade, pode iniciar um serviço e deixá-lo correr ou interagir com ele;
c) provedores de conteúdo (*content providers*): um provedor de conteúdo gerencia os dados compartilhados pela aplicação, como: armazenar arquivos no sistema, em um banco de dados, na internet ou em qualquer outro local de armazenamento;
d) receptores de transmissão (*broadcast receivers*): um receptor de transmissão gerencia as notificações e alertas da aplicação. Um exemplo disso seriam as notificações do sistema que avisam que a bateria está fraca, que uma ligação foi perdida ou que uma mensagem foi recebida. Apesar dos receptores de transmissão não possuírem interface com o usuário, eles podem criar uma notificação na barra de status para alertar o usuário quando ocorrer um evento. Porém é mais comum que o componente sirva de “porta de entrada” para outros componentes, como por exemplo, executar um serviço com base em um evento ocorrido.

Um aspecto único de projeto do Sistema Android é que qualquer aplicação pode iniciar um componente de outra aplicação. Como por exemplo, para tirar um foto a partir de um aplicativo, não é necessário desenvolver uma classe para isso, basta iniciar uma atividade do aplicativo de câmera que faça isso. Em relação a esse aspecto deve ser destacado de que quando o Sistema Android inicia um componente ele inicia o processo da aplicação e instancia as classes necessárias para o funcionamento do mesmo. Dessa forma ao iniciar um componente através de outra aplicação, este será executado em um processo diferente. Como

as aplicações funcionam em processos separados com permissões de arquivo que restringem o acesso a outras aplicações, esta não pode ativar diretamente um componente do outro. Assim, para fazer essa ativação, a aplicação deve enviar um mensagem para o Sistema Android especificando a requisição e então o Sistema Operacional faz a inicialização do componente em particular (ANDROID DEVELOPERS, 2013).

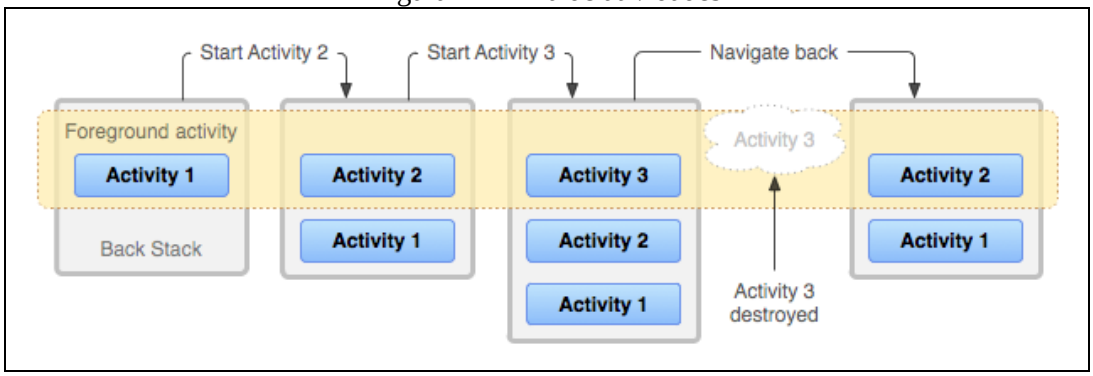
Um projeto Android é composto principalmente de um ou mais componentes de aplicação, sendo a atividade o componente mais utilizado pelas aplicações Android.

2.3.1 Atividade (*activity*)

A atividade é um componente de aplicação que fornece uma interface com a qual o usuário possa interagir. Portanto, uma aplicação consiste normalmente de múltiplas atividades ligadas umas às outras, normalmente com uma atividade especificada como principal, a qual é apresentada ao usuário ao iniciar o aplicativo (ANDROID DEVELOPERS, 2013).

Cada atividade após iniciada pode iniciar outra atividade, com o propósito de executar outras ações. Dessa forma cada vez que uma nova atividade é iniciada, a anterior é interrompida e preservada pelo sistema em uma pilha de atividades chamada de *back stack*. Conforme ilustra a Figura 2, a atividade `Activity 1` inicia a atividade `Activity 2` que por sua vez inicia a `Activity 3`. Nesse momento existem três atividades na pilha, onde a atividade `Activity 3` está no topo da pilha e sendo esta visualizada pelo usuário. Após ser pressionado o botão de retorno, o Sistema Operacional destrói a atividade `Activity 3` e o remove da pilha, assim a `Activity 2` volta a estar topo da pilha e recebe o foco (ANDROID DEVELOPERS, 2013).

Figura 2 – Pilha de atividades



Fonte: Android Developers (2013).

Cada vez que uma atividade é iniciada, ela é armazenada no topo da pilha de atividades e recebe o foco, enquanto a anterior permanece na pilha, porém parada. Assim quando o usuário pressiona o botão de retorno, a atividade atual é retirada do topo da pilha e

destruída, o Sistema Operacional então retorna à atividade anterior, que agora está no topo da pilha, colocando-a em primeiro plano novamente (ANDROID DEVELOPERS, 2013).

Para criar uma atividade é necessário gerar uma classe Java que estende da classe `Activity`. Nesta devem ser implementados os métodos de retorno que o sistema invoca na mudança de estado da atividade. Através da implementação dos métodos de retorno é possível gerenciar o ciclo de vida da aplicação, gerando assim uma aplicação forte e flexível, pois o ciclo de vida de uma atividade é diretamente afetado por sua associação com outras atividades. Uma atividade pode existir essencialmente em três estados (ANDROID DEVELOPERS, 2013):

a) resumida (*resumed*): nesse estado a atividade está em primeiro plano e possui o foco no usuário;
b) pausada (*paused*): uma atividade recebe este estado quando outra atividade está em primeiro plano e possui o foco do usuário, mas nesse caso a atividade pausada ainda está ativa, ou seja, existe uma outra atividade que está sobre ela de forma parcialmente transparente ou que não ocupa toda a tela;
c) interrompida (*stopped*): quando a atividade está sendo totalmente sobreposta por outra, ou seja, ela está em segundo plano e portanto recebe o estado de interrompida. Mesmo interrompida a atividade continua viva no sistema, porém ela não é visível nesse momento.

Quando uma atividade transita de um estado a outro, ela é notificada em diversos métodos de retorno que podem ser sobrescritos para executar ações adequadas à mudança de estado. Os métodos de retorno são (ANDROID DEVELOPERS, 2013):

a) `onCreate()`: ativado quando a atividade é criada pela primeira vez, sendo comumente utilizado para inicialização dos componentes estáticos da atividade;
b) `onRestart()`: apenas ativado após a atividade ter sido interrompida e então retomada novamente;
c) `onStart()`: chamado sempre antes da atividade se tornar visível para o usuário;
d) `onResume()`: chamado imediatamente antes da atividade permitir a interação com o usuário. Neste momento a atividade já está no topo da pilha de atividades;
e) `onPause()`: quando o sistema estiver prestes a retomar uma outra atividade, este método é ativado, sendo geralmente utilizado para confirmar alterações não salvas e parar ações que possam estar consumindo processamento da *Central Processing Unit* (CPU). Este deve ser executado rapidamente, pois a próxima atividade não será retomada até que o método termine de executar;

f) `onStop()`: método ativado quando a atividade já não é mais visível para o usuário, no caso da atividade estar sendo destruída ou uma outra atividade foi iniciada e a está cobrindo;

g) `onDestroy()`: chamado quando a atividade está sendo destruída. Esta é a última chamada de método que a atividade irá executar.

O Quadro 1 mostra uma classe Java de uma atividade implementando todos os métodos acima citados.

Quadro 1 - Representação de uma classe Java implementando todos os métodos de retorno da atividade

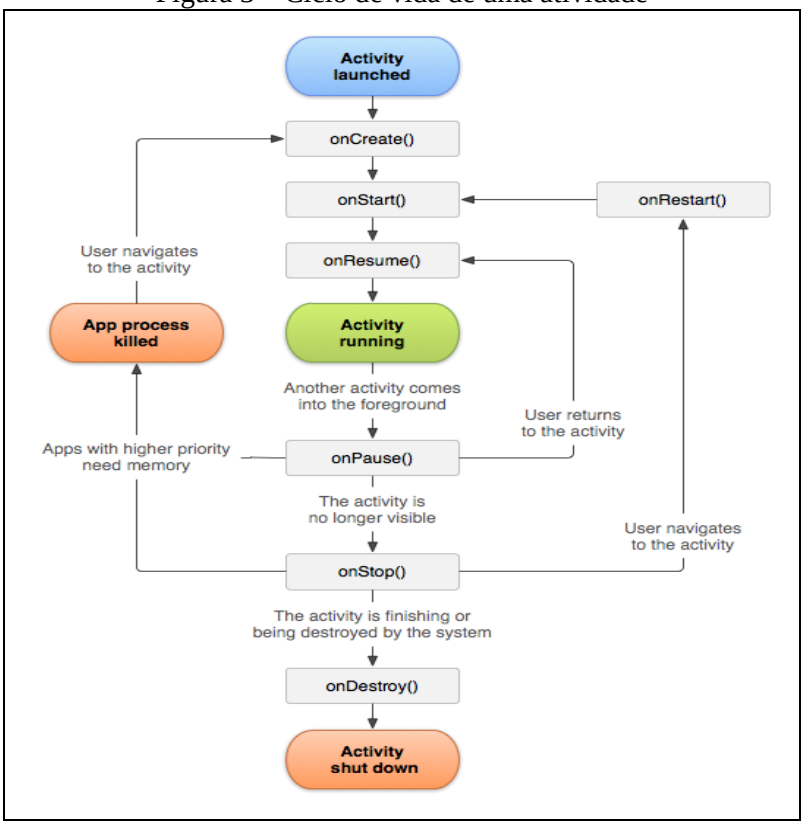
```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be
"paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Fonte: Android Developers (2013).

A Figura 3 ilustra os laços e caminhos que uma atividade pode tomar entre os estados acima citados. Nela pode ser observada a ordem de ativação dos métodos de retorno da atividade, assim como as situações que os ativam. Ao iniciar uma atividade os métodos `onCreate()`, `onStart()` e `onResume()` são ativados em sequência até tornar a atividade visível. Caso alguma outra atividade sobreponha essa, ela ativa o método `onPause()`. Nesse estado a atividade pode ser retomada quando posta em primeiro plano novamente, ativando o método `onResume()`, mas caso ela não esteja mais visível o método `onStop()` é ativado. Nesse ponto a atividade pode ser iniciada novamente através da navegação entre as atividades

ativando sequencialmente os métodos `onRestart()` e `onStart()`. No caso da finalização da atividade, além dos métodos `onPause()` e `onStop()`, por último é ativado o método `onDestroy()` dando fim ao ciclo de vida da atividade.

Figura 3 – Ciclo de vida de uma atividade



Fonte: Android Developers (2013).

2.3.2 Serviços (*services*)

Um serviço é um componente de aplicação sem interface de usuário que permite a execução de operações de longa duração em segundo plano. Este pode ser iniciado por componente de aplicação e mesmo assim continuará sendo executado em segundo plano, mesmo que mude para outro aplicativo. Além disso, um componente de aplicação pode interagir com um serviço e até mesmo realizar comunicação entre os processos. Um serviço pode possuir duas formas (ANDROID DEVELOPERS, 2013):

a) iniciado (*started*): o serviço é iniciado quando um componente de aplicação, como por exemplo uma atividade, o inicia utilizando o método `StartService()`. Uma vez iniciado, um serviço pode ser executado em segundo plano por tempo indeterminado, mesmo que o componente de aplicação que o iniciou tenha sido destruído. Normalmente um serviço iniciado executa uma única operação e não retorna um resultado para quem o iniciou, assim quando a operação é realizada o serviço pára por si mesmo;

b) vinculado (*bound*): um serviço é vinculado quando um componente de aplicação liga-se a ele utilizando o método `BlindService()`. Um serviço vinculado possui uma interface cliente-servidor, que permite aos componentes interagir com o serviço. O serviço vinculado por sua vez funcionará somente enquanto outro componente de aplicação estiver ligado a ele, sendo que vários componentes de aplicação podem ser ligados ao serviço, mas quando não restar nenhuma ligação entre eles o serviço será destruído.

Embora um serviço possa ter essas duas formas, ele ainda pode funcionar em ambos os sentidos, podendo ser iniciado e executado indefinidamente e também permitir o vínculo com outros componentes de aplicação. Para isso basta implementar os métodos `onStartCommand()` para permitir que outros componentes o iniciem e também o métodos `onBind()` para permitir o vínculo entre os componentes de aplicação.

2.3.3 Provedores de conteúdo (*content providers)*

Os provedores de conteúdo são componentes de aplicação responsáveis pela comunicação entre aplicações. Eles gerenciam um conjunto estruturado de dados além de promover o encapsulamento destes fornecendo mecanismos de definição de segurança de dados. Os provedores de conteúdo são principalmente usados para efetuar a comunicação entre aplicativos, permitindo integrar vários tipos de dados, como arquivos de áudio, vídeo, imagens, dados pessoais entre outros (ANDROID DEVELOPERS, 2013).

Um provedor de conteúdo pode ser implementado em uma ou mais classes de uma aplicação Android, desde que estejam informados no arquivo *manifest*. Para permitir que a aplicação efetue a transferência e o gerenciamento de dados com outra aplicação, é necessário criar uma instância da classe `ContentProvider`. Para que uma segunda aplicação possua acesso aos dados do provedor de conteúdo, é necessário utilizar um objeto da classe `ContentResolver`, o qual permite se comunicar com o provedor de conteúdo como um cliente (ANDROID DEVELOPERS, 2013).

2.3.4 Receptores de transmissão (*broadcast receivers)*

Os receptores de transmissão implementam outra variação de mecanismo de comunicação entre processos. Eles possuem um ciclo de vida mais simples que o de outros componentes de aplicação e também são similares a uma atividade, porém não possuem uma interface de usuário (MEDNIEKS *et al.*, 2013).

O mais próximo que se pode chegar de uma interface usando um receptor de transmissão é a partir das notificações geradas por ele na barra de *status* do sistema Android. Estas notificações podem ser usadas para dar alertas ao usuário, informar o andamento de um determinado evento interno, ou para executar uma determinada ação quando interagindo com este. Normalmente os receptores de transmissão são utilizados apenas como ponto de entrada para outros componentes, destinando-se a fazer o mínimo de trabalho, como por exemplo, iniciar um serviço para executar um determinado trabalho baseado no evento ocorrido (ANDROID DEVELOPERS, 2013).

2.3.5 Ativação dos componentes de aplicação

A ativação de três dos principais componentes de aplicação (*activities, services* e *broadcast receivers*) é feita através de mensagens, chamadas de *intent*. Uma *intent* faz a ligação entre componentes individuais em tempo de execução, não importando se o componente faz parte da mesma aplicação ou de outra. A *intent* é criada a partir de um objeto da classe `intent`, o qual define uma mensagem para ativar um objeto específico ou um tipo específico de componente. Ele pode ser explicito ou implícito respectivamente. Para *activities* ou *services* uma *intent* define a ação a ser executada, como ver ou enviar algo, para *broadcast receivers* a *intent* simplesmente define as notificações exibidas na barra superior do dispositivo (ANDROID DEVELOPERS, 2013).

O *content provider*, por sua vez, não é ativado por uma *intent*. Em vez disso a ativação do componente é feita através do pedido de um *contente resolver*. Este lida com todas as transações feitas diretamente ao *content provider* de modo que o componente que está realizando essas transações não precise chamar os métodos do objeto. Isto gera uma camada de abstração entre o *content provider* e o componente que fez a requisição da informação (ANDROID DEVELOPERS, 2013).

Para que o Sistema Operacional Android possa iniciar um componente de aplicação, ele deve conhecer a existência do mesmo. Isto é feito a partir da leitura do arquivo `AndroidManifest.xml` da aplicação utilizada, conhecido como arquivo *manifest*. Todos os componentes utilizados na aplicação devem ser adicionados neste arquivo, o qual deve estar na raiz do diretório do projeto da aplicação (ANDROID DEVELOPERS, 2013).

Além da declaração dos componentes, o arquivo *manifest* faz uma série de coisas, tais como:

a) identificar qualquer permissão de usuário que o aplicativo deva ter, como acesso a *intent* ou a leitura dos contatos;

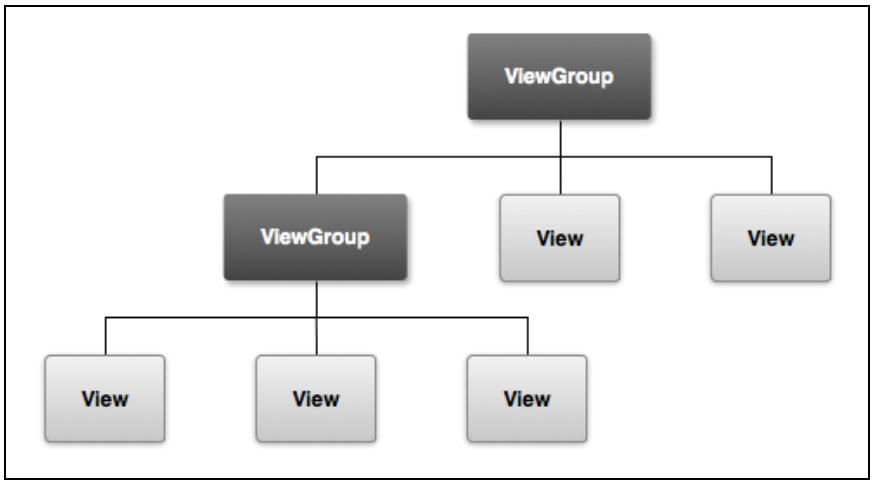
b) declarar o nível mínimo da API exigido para execução do aplicativo;
c) identificar os recursos de *hardware* e *softwares* exigidos pelo aplicativo, como câmera, *Global Positioning System* (GPS) ou serviços de *bluetooth*;
d) declarar as bibliotecas de API que precisam ser ligadas ao aplicativo, como para uso de mapas é necessário as bibliotecas do *Google Maps*.

A declaração dos componentes e outros aspectos do funcionamento do arquivo *manifest* são detalhados na seção 2.4.

2.3.6 Interface de usuário

Uma *User Interface* (UI) em Android é construída através de objetos derivados da classe `View` e `ViewGroup`. Cada objeto da classe `View` controla um determinado espaço retangular dentro da janela da atividade e pode responder à interação do usuário. A classe `View` é a classe base dos objetos de interface, como botões e campos de texto, chamados widgets. A classe `ViewGroup` é a classe base para os *layouts*, que são responsáveis por estruturar os componentes da interface. A combinação dessas duas classes bases define a hierarquia de uma interface Android, como pode ser visto na Figura 4 onde cada objeto da classe `ViewGroup` contém outros objetos filhos da classe `View` e `ViewGroup`. Essa hierarquia pode ser simples ou complexa, porém quanto mais simples melhor é o desempenho (ANDROID DEVELOPERS, 2013).

Figura 4 – Definição do *layout* da interface de usuário através de uma hierarquia de objetos da classe `View`



Fonte: Android Developers (2013).

O Android fornece uma série de objetos prontos, derivados da classe `View`, chamados *widgets*. Os *widgets* são componentes visuais e interativos que podem ser dispostos em tela, como um botão, um campo texto ou uma imagem. No Android um *layout* é um objeto

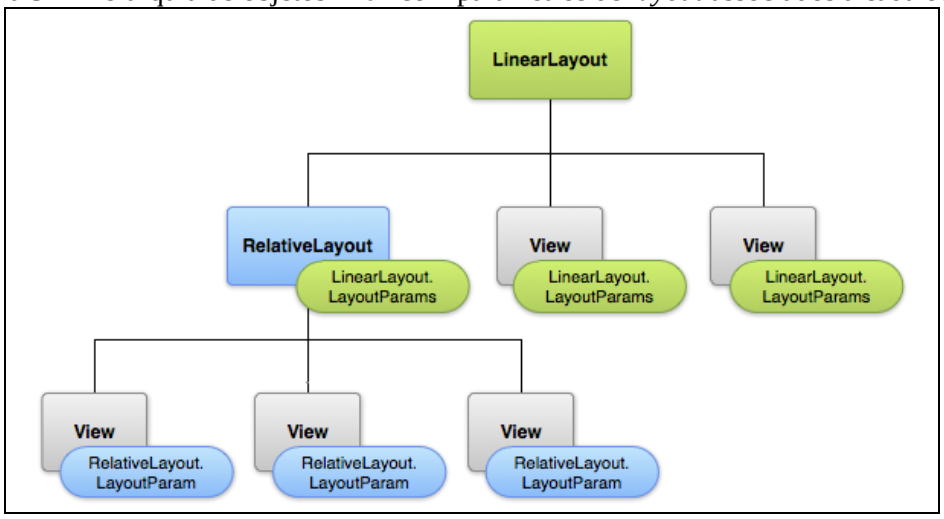
derivado da classe `ViewGroup` que fornece um modelo único de tela onde podem ser dispostos os *Widgets* ou outros componentes filhos da classe `ViewGroup`.

Os *layouts* definem a estrutura visual para uma interface de usuário, podendo ser declarado de duas formas: utilizando o vocabulário fornecido pelo Android para declarar os elementos de interface de usuário em um arquivo *eXtensible Markup Language* (XML) ou instanciando esses elementos em tempo de execução no código fonte da aplicação. Entretanto, a vantagem de declarar a interface de usuário em um arquivo XML está na melhor forma de separar o código de interface do código fonte da aplicação. Assim as descrições da interface estarão separadas do código do aplicativo, o que permite adaptar e alterar a interface sem a necessidade de alterar o código fonte (ANDROID DEVELOPERS, 2013).

Cada arquivo de *layout* deve conter exatamente um elemento raiz, que deve ser um objeto `View` ou `ViewGroup`. Uma vez definido o objeto raiz é possível adicionar outros objetos de *layout* ou *widgets* como filhos, para construir gradualmente uma hierarquia de objetos `View` e definir uma interface de usuário (ANDROID DEVELOPERS, 2013).

Todo objeto `View` e `ViewGroup` suporta uma variedade de atributos na sua construção no arquivo XML. Alguns atributos são específicos para um objeto `View`, mas estes também podem ser herdados por qualquer outro objeto descendente da classe `View` que possa estender essa classe. Alguns atributos são comuns a todos os objetos pois são herdados da classe raiz `View`, mas existem outros que são considerados parâmetros de *layout*, os quais descrevem certas orientações de *layout* do objeto `View` conforme definidos pelo objeto pai `ViewGroup`, como pode ser visto na Figura 5 (ANDROID DEVELOPERS, 2013).

Figura 5 – Hierarquia de objetos `View` com parâmetros de *layout* associados a cada objeto



Fonte: Android Developers (2013).

Cada classe `ViewGroup` implementa uma classe que estende de `ViewGroup.LayoutParams`. Esta subclasse contém propriedades que definem o tamanho e

posição para cada objeto `View` filho, conforme for apropriado ao objeto `ViewGroup`. Assim como pode ser visto na Figura 5, o objeto `ViewGroup` pai define parâmetros de *layout* para cada objeto `View` filho, incluindo o objeto `ViewGroup` filho.

As subclasses da classe `ViewGroup` fornecem uma maneira única de construir uma interface. As subclasses mais comuns utilizadas na criação de *layouts* e disposição de conteúdo fixo ou pouco mutável são: `LinearLayout` e `RelativeLayout`. Quando o conteúdo do *layout* é dinâmico, o Android permite a utilização da subclasse `AdapterView` para preencher o *layout* em tempo de execução. O `AdapterView` serve como um adaptador que liga os dados ao *layout*, convertendo cada entrada de dados em uma `View` que será exibida no *layout*. Os mais comuns são `ListView` e `GridView` (ANDROID DEVELOPERS, 2013).

2.3.6.1 `LinearLayout`

Todos os componentes, objetos `View`, posicionados dentro de um `LinearLayout` são empilhados um após o outro. Assim formam uma lista, com um único componente por linha, independente do tamanho deste. Cada linha possui o mesmo tamanho, sendo este definido pelo maior componente posicionado dentro do *layout* (ANDROID DEVELOPERS, 2013).

Por padrão, o `LinearLayout` organiza os componentes na forma horizontal. A orientação da organização dos componentes é feita através do atributo `android:orientation` no arquivo de definição XML, podendo receber os valores `vertical` ou `horizontal`, e através do código fonte utilizando o método `setOrientation`, podendo também receber os mesmos valores para o atributo XML (ANDROID DEVELOPERS, 2013).

2.3.6.2 `RelativeLayout`

Todos os componentes, objetos `View`, dispostos dentro de um `RelativeLayout` são exibidos em posições relativas. A posição de cada componente é relativa ao posicionamento de outro, desde que estejam no mesmo *layout*. Um componente pode ter sua posição especificada para ficar à esquerda, direita, embaixo, em cima, entre outros, de outro componente ou até mesmo em relação à área do `RelativeLayout` (ANDROID DEVELOPERS, 2013).

Por padrão, todos os componentes dentro de um `RelativeLayout` são dispostos na parte superior esquerda do *layout*. O posicionamento do componente em relação ao *layout* pode ser definido de diversas formas utilizando vários atributos que permitem o alinhamento deste tanto ao *layout* quanto a outro componente. O Quadro 2 apresenta alguns dos atributos que alinham a margem do componente a determinada margem do *layout*. Enquanto o Quadro

3 apresenta algumas propriedades que alinham o componente em relação a outro componente disposto no mesmo *layout* (ANDROID DEVELOPERS, 2013).

Quadro 2 – Atributos alinhamento do componente em relação ao *layout*

Atributo	Valor	Funcionamento
`layout_alignParentStart`	`true/false`	Alinha a margem inicial
`layout_alignParentTop`	`true/false`	Alinha a margem superior
`layout_alignParentRight`	`true/false`	Alinha a margem a direita
`layout_alignParentLeft`	`true/false`	Alinha a margem a esquerda
`layout_alignParentBottom`	`true/false`	Alinha a margem inferior
`layout_alignParentEnd`	`true/false`	Alinha a margem final
`layout_centerInParent`	`true/false`	Posiciona o componente ao centro, nos sentidos vertical e horizontal
`layout_centerHorizontal`	`true/false`	Posiciona o componente ao centro no sentido horizontal
`layout_centerVertical`	`true/false`	Posiciona o componente ao centro no sentido vertical

Quadro 3 – Alinhamento do componente em relação a outro componente no *layout*

Atributo	Valor	Funcionamento
`layout_above`	`@[package:]type:name`	Posiciona a margem inferior do componente acima da margem superior do componente referenciado
`layout_below`	`@[package:]type:name`	Posiciona a margem superior do componente abaixo da margem inferior do componente referenciado
`layout_toStartOf`	`@[package:]type:name`	Posiciona a margem final do componente à frente da margem inicial do componente referenciado
`layout_toRightOf`	`@[package:]type:name`	Posiciona a margem esquerda do componente ao lado da margem direita do componente referenciado
`layout_toLeftOf`	`@[package:]type:name`	Posiciona a margem direita do componente ao lado da margem esquerda do componente referenciado
`layout_toEndOf`	`@[package:]type:name`	Posiciona a margem inicial do componente atrás da margem final do componente referenciado

2.3.6.3 `ListView`

Este *layout* exibe uma lista de itens com a utilização de uma barra de rolagem. Os itens são adicionados na lista por meio de um adaptador. Este obtém as informações de uma fonte e converte cada item do resultado da busca em um componente `View` que é exibido na forma de um item na lista (ANDROID DEVELOPERS, 2013).

Para inserir registros no *layout* e criar itens na lista de forma dinâmica, é utilizada a classe `Adapter`, porém é possível adicionar itens pré-definidos através de uma lista fixa

também pré-definida em um arquivo de recurso do projeto. Para tanto, basta utilizar o atributo `entries` na definição do componente no arquivo de interface XML (ANDROID DEVELOPERS, 2013).

2.3.6.4 `GridView`

Este *layout* exibe uma lista de itens em uma grade bidimensional com a utilização de barras de rolagem. Assim como no `ListView`, os itens são adicionados por meio de um adaptador, a partida da classe `ListAdapter`. Entre os principais atributos do `GridView`, pode-se destacar: `android:numColumns`, permite determinar o número de colunas, `android:columnWidth`, determina o tamanho fixo das colunas e `android:gravity`, determina o posicionamento do conteúdo dentro de cada célula (ANDROID DEVELOPERS, 2013).

2.3.6.5 Controles de entrada

São componentes interativos na interface de usuário da aplicação. O Android oferece uma ampla variedade de controles para a criação da interface. Os mais comuns são:

a) botões: consiste de um texto, um ícone ou ambos e que aciona um evento quando for tocado. Utilizam as classes `Button` e `ImageButton`;
b) campos texto: permite o usuário digitar um texto na aplicação, que pode ser em uma única linha ou em várias. É possível especificar o tipo de teclado que será ativado para a entrada de dados no componente, como somente números ou letras em maiúsculo. utilizam a classe `EditText` e `AutoCompleteTextView`;
c) caixas de seleção: consiste comumente de um campo texto com uma caixa de seleção. Este permite o usuário selecionar uma ou mais opções em uma lista. Utilizam a classe `CheckBox`;
d) botão de opção: consiste de um campo texto com um círculo de marcação. Este permite o usuário selecionar apenas uma opção em uma lista. Utilizam as classes `RadioGroup` e `RadioButton`;
e) botões de alternância: permite o usuário alterar uma opção entre dois estados. Utilizam a classe `ToggleButton` e `Switch`;
f) caixas de seleção: fornece uma maneira rápida para selecionar um valor a partir de um conjunto. Em seu estado normal ele exibe o item selecionado, ao ser tocado é exibido um menu com todos os valores disponíveis. Utilizam a classe `Spinner` e `SpinnerAdapter`;

g) extratores: controles que permitem o usuário selecionar uma data ou hora de forma a garantir que elas sejam informações válidas. Utilizam as classes `DatePickerDialog` e `TimePickerDialog`.

Cada controle de entrada suporta um conjunto específico de eventos de entrada, como um botão, que pode ser pressionado para executar uma ação, ou um campo texto, que pode utilizar um controle para habilitar a opção de auto completar ao digitar um texto.

2.4 ESTRUTURA DE UM PROJETO ANDROID

Um projeto Android contém todos os arquivos que compõem o código fonte da aplicação Android. Estes são gerados no momento da criação do projeto, que por usa vez pode ser feito utilizando o ambiente Eclipse, desde que esteja configurado com o *plug-in Android Development Tools* (ADT), ou a partir da execução de linhas de comando utilizando as ferramentas do SDK Android. Independente da forma que o projeto é criado, ele conterá os diretórios onde estão os arquivos de código fonte e de recursos utilizados pela aplicação.

Um projeto Android é composto principalmente de arquivos XML, com o qual possuem uma ligação muito forte. Dentre as utilizações desses arquivos, as mais importantes são: o mapeamento dos componentes de aplicação do sistema, a definição da interface de usuário e dos atributos do sistema utilizados pela aplicação. O mapeamento dos componentes de aplicação e a declaração das permissões de acesso, entre outras informações, são feitas através do arquivo `AndroidManifest.xml`, este que por sua vez está localizado no diretório base do projeto Android. Por outro lado os arquivos de interface, listas pré-definidas, imagens entre outros recursos utilizados pela aplicação, são organizados em subpastas a partir do diretório `res`, destinado aos recursos do projeto Android. O acesso aos recursos é feito através da classe `R`, a qual é gerada automaticamente pela ferramenta *Android Asset Packaging Tool* (AATP) utilizando referências chamadas *id* (ANDROID DEVELOPERS, 2013).

2.4.1 Android *Manifest*

Toda aplicação Android deve possuir em seu diretório base o arquivo `AndroidManifest.xml`, chamado de arquivo *manifest*. Nele estão presentes informações essenciais sobre a aplicação e necessárias para o Sistema Operacional Android. O arquivo *manifest* organiza uma aplicação em uma estrutura bem definida, a qual é compartilhada por todas as aplicações e permite que o Sistema Operacional Android carregue e execute-as em um ambiente gerenciado (MEDNIEKS *et al.*, 2013).

Entre as principais declarações do arquivo *manifest*, podem ser destacados: as permissões de usuário, como acesso de internet ou leitura dos contatos do usuário; a declaração do nível mínimo de API necessário para rodar a aplicação; a declaração dos dispositivos de hardware utilizados pela aplicação, como câmera e *bluetooth*; as bibliotecas em que o aplicativo precisa se conectar, como a API de mapas do Google; e principalmente a declaração dos componentes de aplicação utilizados, como atividades e serviços. No Quadro 4 é exibido um arquivo *manifest* de uma aplicação (MEDNIEKS *et al.*, 2013).

Quadro 4 - Arquivo `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.oreilly.demo.pa.ch01.testapp"
          android:versionCode="1"
          android:versionName="1.0">

   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
   <uses-permission android:name="android.permission.CALL_PHONE" />
   <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
   <uses-permission android:name="android.permission.INTERNET" />

   <application android:icon="@drawable/icon"
                android:label="@string/app_name"
                android:debuggable="true">

       <activity android:name=".TestActivity"
                 android:label="Test Activity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN"/>
               <category android:name="android.intent.category.LAUNCHER"/>
           </intent-filter>
       </activity>

       <provider android:name=".TestProvider"
                 android:authorities= "com.oreilly.demo.pa.ch11.video.FinchVideo"
       />

       <service android:name=".TestService"
                android:label="Test Service"/>

       <receiver
                 android:name=".TestBroadcastReceiver"
                 android:label="Test Broadcast Receiver"/>
   </application>
   <uses-sdk android:minSdkVersion="7" />

</manifest>
```

Fonte: Mednieks *et al.* (2013).

Como todo arquivo XML, na primeira linha estão descritas a versão do XML e a codificação de caracteres utilizados. Nas linhas seguintes são definidos os parâmetros e as permissões requeridas pela aplicação. No nó raiz `<manifest>` estão descritos os atributos `package`, responsável por definir o pacote padrão, `android:versionCode`, valor inteiro utilizado para determinar a versão da aplicação, `android:versionName`, valor texto utilizando para determinar a versão da aplicação com vários caracteres. Em seguida são adicionadas quatro permissões de acesso, as quais serão solicitadas ao usuário no momento

da instalação da aplicação, a permissão de usuário é declarada utilizando `<uses-permission android:name="PERMISSÃO"/>`. Logo abaixo das permissões é declarado um novo nó `<application>` onde serão adicionados os atributos da aplicação. O atributo `android:icon` aponta para o ícone utilizado pela aplicação e localizado nos recursos do projeto, enquanto o atributo `android:label` descreve o nome da aplicação lido pelo usuário. Em seguida estão descritos os componentes de aplicação utilizados, começando pela atividade com o nó `<activity>` e em seguida por seus atributos: `android:name`, se refere ao nome da classe de atividade, `android:label`, define o título que deve ser exibido ao usuário no topo da tela, `intent-filter`, define um filtro de intenção que diz ao Sistema Operacional Android quando a atividade deve ser executada. Abaixo da atividade está declarado o componente provedor de conteúdo com o nó `<provider>`, ao qual o atributo `android:name` define a classe correspondente ao componente e o atributo `android:authorities` especifica as autoridades *Uniform Resource Identifier* (URI) que o componente deve seguir. Após o provedor de conteúdo, está declarado um componente de aplicação serviço com o nó `<service>`, ao qual o atributo `android:name` define a classe correspondente e `android:label`, define um título que possa ser lido pelo usuário. Por último está definido o componente receptor de transmissão com o nó `<receiver>`, ao qual o atributo `android:name` define a classe correspondente e `android:label`, define um título que possa ser lido pelo usuário (MEDNIEKS *et al.*, 2013).

2.4.2 *Resources* (recursos)

Além do código fonte, as aplicações Android podem precisar armazenar grandes quantidades de dados para serem utilizadas em tempo de execução. Estes dados podem ser imagens, textos, cores de fundo ou nomes de fontes. Essas informações são chamadas de recursos e de acordo com as melhores práticas de software, deve ser mantida separada do código (MEDNIEKS *et al.*, 2013).

De modo mais simples, recursos são arquivos adicionais com conteúdo estático utilizado pelo aplicativo e que ficam separados do código fonte. Todos os recursos utilizados na aplicação são organizados a partir da pasta `res`, a qual é criada junto com o projeto Android e por padrão contém as subpastas `drawable`, `layout` e `values`.

2.4.2.1 `res.drawable`

Na pasta `drawable` são armazenados componentes gráficos que podem ser exibidos em tela, sendo obtidos através dos métodos da API `getDrawable(int)` diretamente pelo código fonte ou aplicados em outros arquivos de recurso XML, utilizando os atributos

android:drawable e android:icon, como pode ser visto nos Quadros 5 e 6, respectivamente.

Quadro 5 – Aplicando a imagem a uma View

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

Fonte: Android Developers (2013).

Quadro 6 – Obtendo a imagem através dos recursos da aplicação

```
Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

Fonte: Android Developers (2013).

2.4.2.2 res.layout

A pasta layout contém os arquivos que definem a arquitetura para a interface de usuário em uma atividade ou um componente de interface. Estes arquivos possuem a extensão .xml e são construídos utilizando o vocabulário Android XML (ANDROID DEVELOPERS, 2013).

Um arquivo de *layout* é definido através de Views, que podem ser pré-definidas pelo ambiente ou por novas Views definidas pelo desenvolvedor. Cada arquivo de *layout* deve conter apenas um elemento principal, ao qual podem ser adicionados outros elementos gradualmente. Os elementos adicionados possuem atributos, alguns são específicos para cada objeto, mas os mais comuns e suportados por qualquer objeto view são: android:id, android:layout_width e android:layout_height como mostra o Quadro 7 (ANDROID DEVELOPERS, 2013).

Quadro 7 – Exemplo de *layout*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Fonte: Android Developers (2013).

Qualquer objeto View pode possuir um id associado a ele, que é utilizado posteriormente para ser referenciado por outros elementos no XML ou no código fonte da aplicação. Sua definição é feita através do atributo android:id, que deve receber um valor

texto no seguinte formato `@+pacote/nome`, onde @ identifica que o texto se refere a um `id` de um recurso, o símbolo + indica que um novo recurso deve ser criado na classe `R`, `pacote` se refere ao grupo em que o elemento será adicionado e o `nome` se refere ao nome do elemento que deve ser único no pacote. Os atributos `android:layout_width` e `android:layout_height` são obrigatórios e responsáveis por definir a dimensão dos elementos em tela, podendo receber os valores `wrap_content`, `match_parent` e `fill_parent` ou valores numéricos com a terminação: px (*pixels*), dp (*pixels* independentes de densidade), sp (pixels escalados), in (polegadas) e mm (milímetros). O valor `wrap_content` define a dimensão do elemento ao tamanho necessário para exibir o conteúdo desse elemento, `match_parent` e `fill_parent` definem a dimensão do elemento para corresponder ao elemento pai.

2.4.2.3 `res.values`

A pasta `values` contém arquivos XML com valores simples, como textos, números e cores. Por padrão quando a pasta é criada ela contém o arquivo `strings.xml`. Este fornece um arquivo de recurso que armazena textos simples que podem ser acessados a qualquer momento pelo desenvolvedor utilizando a referência `R.string.`*`string_name`* para uso na classe Java ou `@string/`*`string_name`* quando utilizado em XML (ANDROID DEVELOPERS, 2013).

2.4.3 Classe `R`

O acesso aos recursos do projeto pelas classes Java é feito por meio da classe `R`, que por sua vez é gerada no momento da compilação do projeto pela ferramenta *Android Asset Packaging Tool* (AAPT). O AAPT faz a compilação dos arquivos de recursos, como `AndroidManifest.xml` e outros arquivos XML, gerando o arquivo `R.java` que permite ao desenvolvedor referenciar os recursos do projeto no código Java (MEDNIEKS *et al.*, 2013).

Para cada tipo de recurso, presente no diretório `res`, há uma subclasse de `R`, como para o diretório `drawable` existe a subclasse `R.drawable`, e para cada recurso desse tipo existe um número inteiro e estático. No caso de uma imagem `icon.png` presente na pasta `drawable` seria `R.drawable.icon`. Este número inteiro é o *id* do recurso e que poderá ser utilizado para recuperar o recurso pelo código fonte Java (MEDNIEKS *et al.*, 2013).

2.5 AMBIENTE DELPHI

O ambiente Delphi é uma ferramenta de desenvolvimento amplamente utilizada no desenvolvimento de aplicações com interface de usuário para a plataforma Microsoft Windows. Esta é classificada como uma ferramenta RAD construída sobre linguagem *Object Pascal*, uma extensão orientada a objetos da linguagem Pascal, sendo também a principal linguagem de desenvolvimento utilizada pela ferramenta.

O que destaca o Delphi como uma ferramenta de desenvolvimento é a sua *Integrated Development Environment* (IDE), a qual fornece vários mecanismos que aceleram o processo de desenvolvimento de aplicações. Um exemplo disso é a criação de interfaces gráficas, esta é feita de modo prático e fácil pois o ambiente utiliza formulários ou janelas, que permitem ao desenvolvedor utilizar o recurso de *drag-and-drop* (arrastar-e-soltar), para adicionar os componentes visuais que irão formar a interface. Esses componentes por sua vez são dispostos em abas ou listas de acordo com a versão do ambiente, e que também podem ser ainda mais facilmente localizados por meio de ferramentas de busca presentes no ambiente. As Figuras 6 e 7 ilustram a disposição dos componentes através da paleta e da ferramenta de busca respectivamente (CANTU, 2003).

Figura 6 – Paleta de componentes do Delphi versão 7

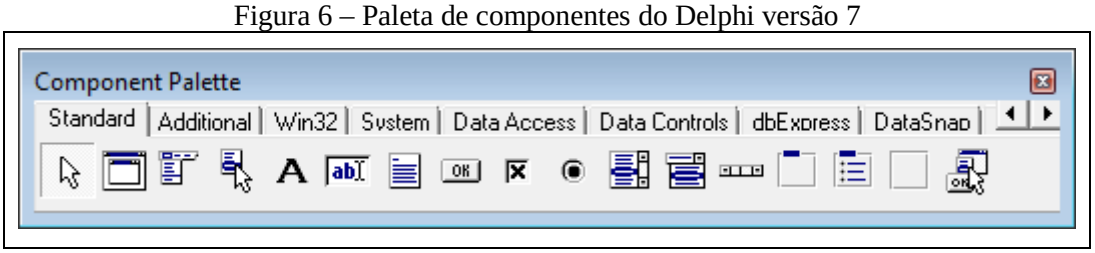
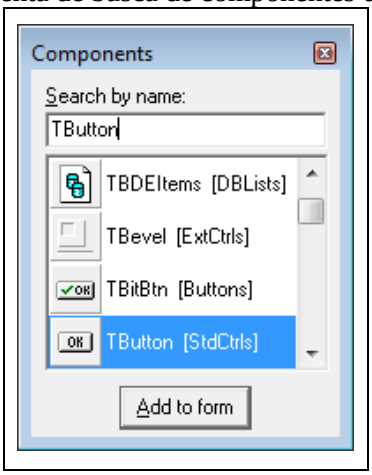


Figura 7 – Ferramenta de busca de componentes do Delphi versão 7

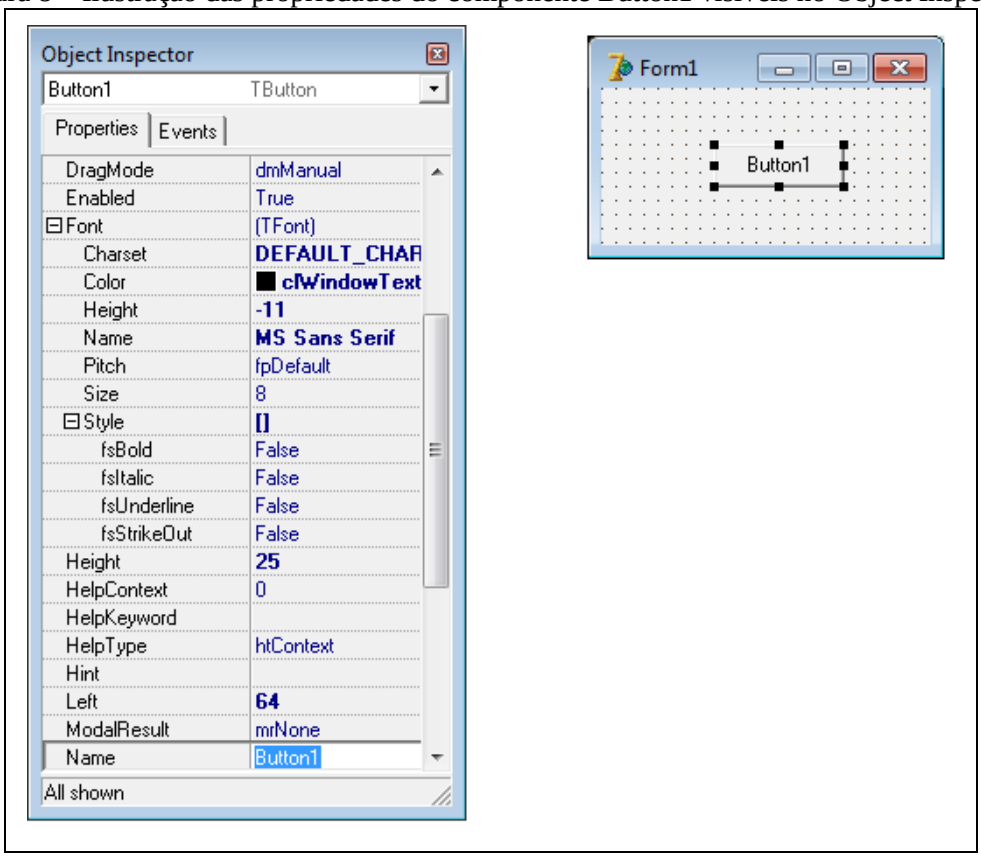


O Delphi utiliza a *Visual Component Library* (VCL), uma biblioteca de componentes visuais mapeada sobre a API do Windows e implementada em *Object Pascal*. Ela inclui controles nativos do Windows, como botões e caixas de edição, controles comuns como

modos de visualização em árvore ou lista, além de vários controles do Delphi ligados ao conceito de janela do Windows. De forma técnica os componentes são subclasses da classe `TComponent`, classe base ou raiz da hierarquia. A partir dessa classe os componentes visuais herdam também da classe `TControl`. Nessa eles obtém atributos que determinam posição e tamanho do componente em tela, além de serem exibidos em um formulário em tempo de projeto e execução. Após `TControl` um componente visual ainda pode herdar outras classes, obtendo assim outras propriedades e eventos (CANTU, 2003, p. 89).

Os atributos de uma classe podem ser acessados de acordo com a seção em que foram declarados, sendo elas: `public`, acessado por qualquer classe em qualquer parte do programa; `private`, acessado somente na própria classe; `protected`, acessado na própria classe ou subclasses; e `published`, pode ser acessado por qualquer classe e é publicada pelo compilador. Assim como uma classe normal, um componente possui seus atributos, chamados de propriedades, que podem ser acessados em tempo de projeto para determinar as propriedades do componente, como: nome, tamanho, posição, entre outros. Estes atributos são declarados na seção `published`, e assim publicados pelo Delphi durante a compilação do componente, fazendo com que estes se tornem visíveis e tornando possível o acesso a estes através do *Object Inspector*, como ilustra a Figura 8 (CANTU, 2003, p. 92).

Figura 8 – Ilustração das propriedades do componente Button1 visíveis no Object Inspector



As propriedades de um componente, assim como as de qualquer classe, podem ser acessadas em tempos de execução. Esse acesso pode ser feito diretamente utilizando o objeto do componente seguido de um ponto e a propriedade, ou através do método `GetPropValue`, utilizando como parâmetro um objeto e o nome literal da propriedade, retornando um valor do tipo `Variant`. Os objetos de um formulário podem ser obtidos também em tempo de execução, através da lista de componentes do formulário utilizando a propriedade `Components`. Esta propriedade armazena todos os componentes adicionados ao formulário com o tipo base `TComponent`. Além desta ainda existe a propriedade `ComponentCount` que armazena o número de componentes presentes no formulário e por consequentemente na lista (CANTU, 2003, p. 94).

2.6 TRABALHOS CORRELATOS

Fonseca (2005) desenvolveu uma ferramenta que permite a conversão de formulários Delphi em aplicações Java. Esta aplicação lê o conteúdo dos arquivos *Delphi ForM (*DFM) e produz classes Java que apresentam o mesmo comportamento dos formulários desenvolvidos em Delphi. A ferramenta Delphi2Java-II (SILVEIRA, 2006) complementou o projeto anterior e foi concebida como um gerador de interface e de código para acesso ao banco de dados. Em termos de funcionalidades, a ferramenta implementa:

a) conversão de formulários Delphi (arquivos com extensão DFM) para classes Java;
b) a conversão de 22 componentes de visualização (`TButton`, `TChebckBox`, `TComboBox`, `TCoolBar`, `TEdit`, `TForm`, `TLabel`, `TListBox`, `TMainMenu`, `TMemo`, `TMenuItem`, `TPageControl`, `TPanel`, `TProgressBar`, `TRadioButton`, `TRadioGroup`, `TScrollBar`, `TSpinEdit`, `TStatusBar`, `TStringGrid`, `TTabSheet` e `TToolBar`);
c) a conversão de 7 componentes de visualização de dados (`TDBCheckBox`, `TDBComboBox`, `TDBEdit`, `TDBGrid`, `TDBMemo`, `TDBRadioGroup` e `TDBText`);
d) a conversão de 4 componentes de acesso ao banco de dados (`TTable`, `TQuery`, `TDatabase` e `TDataSource`), implementando suas principais funcionalidades;
e) a interligação dos componentes de acesso ao banco de dados com os componentes de visualização de dados, implementada em Delphi através do componente `TDataSource`;
f) a geração de uma classe contendo a assinatura dos métodos para os principais eventos (`onCreate`, `onDestroy`, `onClick`, `onChange`, `onEnter`, `onExit`, `onCloseUp` e `onKeyPress`) habilitados na aplicação Delphi, permitindo que o

código seja futuramente inserido.

Foi desenvolvido por Souza (2005) uma ferramenta para conversão de formulários Delphi em páginas *HyperText Markup Language* (HTML). A ferramenta DelphiToWeb faz a interpretação de um arquivo DFM com as definições do formulário Delphi e depois a sua conversão para uma interface em páginas HTML. A ferramenta faz a conversão de 22 componentes de interface do Delphi, sendo considerados como critério de seleção os componentes mais utilizados e com equivalentes para HTML e Laszlo (SOUZA, 2005, p. 29). Os componentes convertidos 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).

Zimmermann (2011) desenvolveu uma ferramenta para conversão de interfaces gráficas desenvolvidas em Delphi para a biblioteca *Gimp Tool Kit* (GTK+). A ferramenta DelphiToGTK+ faz o processamento dos arquivos de definição de interface do Delphi, arquivos com extensão `.dfm`, e gera arquivos de definição de interface segundo as especificações da biblioteca `Libglade`, com extensão `.glade` (ZIMMERMANN, 2011, p. 33). No trabalho são convertidos pela ferramenta um total de 25 componentes de interface do Delphi, sendo considerado como critério de seleção os mais comumente utilizados. Foram identificadas três diferentes categorias de tradução para os componentes:

a) componentes de tradução simples: são aqueles que possuem um similar na biblioteca `Libglade`, os quais são: `TForm`, `TLabel`, `TEdit`, `TMemo`, `TButton`, `TCheckBox`, `TRadioButton`, `TComboBox`, `TImage`, `TScrollBox`, `TStaticText`, `TPageControl`, `TRichEdit`, `TProgressBar`, `TMonthCalendar`, `TStatusBar`, `TToolBar`, `TCoolBar`, `TSpinEdit` e `TCalendar` (ZIMMERMANN, 2011, p. 31-32);
b) componentes de tradução condicionada: são aqueles cuja tradução é condicionada ao valor de uma ou mais propriedades do componente, os quais são: `TMenuItem`, `TScrollBar` e `TTrackBar` (ZIMMERMANN, 2011, p. 32);
c) componentes de tradução especial: são aqueles cuja tradução para a biblioteca `Libglade` resultam em dois ou mais componentes para que se possa simular a aparência original, os quais são: `TPanel` e `TGroupBox` (ZIMMERMANN, 2011, p. 32-33).

3 DESENVOLVIMENTO

Esse capítulo contextualiza e descreve o desenvolvimento da ferramenta DelphiToAndroid. Inicialmente são apresentados os principais requisitos atendidos pela ferramenta. Em seguida são descritos os componentes visuais da VCL Delphi convertidos, assim como uma breve descrição dos mesmos e dos componentes Android utilizados na conversão. Ainda são descritos nesse capítulo a especificação da entrada e de saída da ferramenta, além da especificação da própria ferramenta através de diagramas *Unified Modeling Language* (UML) de casos de uso, de classes e de sequência. Por fim é descrito o processo de implementação da ferramenta, assim como as técnincas e outras ferramentas utilizadas, os resultados e discuções obtidos e os problemas e limitações identificados.

3.1 REQUISITOS

A seguir são apresentados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF), atendidos pela ferramenta. No Quadro 8 são apresentados os requisitos funcionais e sua rastreabilidade em relação aos casos de uso. O Quadro 9 lista os requisitos não funcionais da ferramenta.

Quadro 8 - Requisitos funcionais

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: Permitir o mapeamento de componentes de interface do Delphi para Android.	UC01
RF02: Permitir a criação de um projeto Android a partir da interface gerada.	UC02
RF03: Permitir a compilação do projeto Android e geração do arquivo `.apk`.	UC03
RF04: Permitir a instalação do arquivo compilado em um dispositivo ou máquina virtual Android.	UC04

Quadro 9 - Requisitos não funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Ser implementado utilizando a linguagem de programação Delphi.
RNF02: Ser desenvolvido para executar a partir do ambiente de desenvolvimento Delphi.

3.2 COMPONENTES CONVERTIDOS

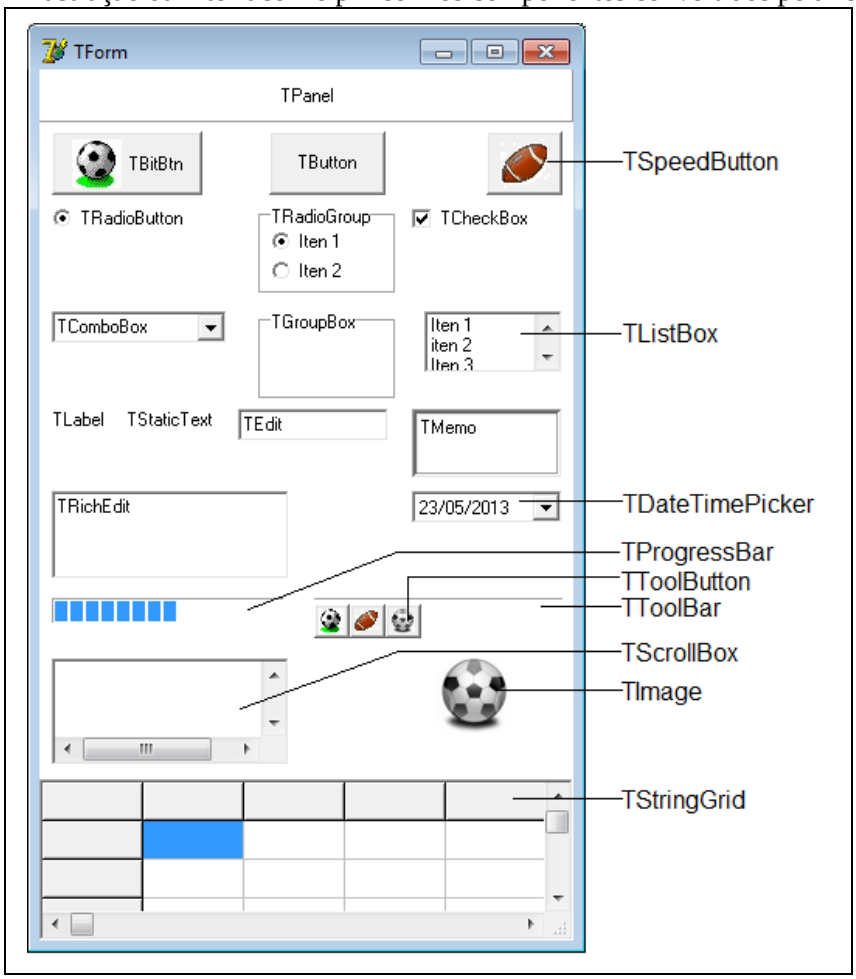
Essa seção apresenta os componentes gráficos mais utilizados no desenvolvimento de uma interface Delphi que foram convertidos pela ferramenta. Durante o estudo e desenvolvimento da ferramenta identificou-se um conjunto de componentes de interface Delphi que utilizam mais de um componente de interface Android para tradução.

No Quadro 10 são apresentados os componentes convertidos pela ferramenta, seguido por uma breve descrição destes e dos componentes de interface Android necessários para a tradução. Na Figura 9 estes componentes Delphi podem ser visualizados.

Quadro 10 – Descrição dos componentes traduzidos

DELPHI	DESCRIÇÃO	ANDROID
TForm	janela ou formulário da aplicação	RelativeLayout
TPanel	painel ou área de agrupamento de componentes	RelativeLayout, EditText
TBitBtn	botão com rótulo e figura	Button
TButton	botão com rótulo e sem figura	Button
TSpeedButton	botão sem rótulo e com figura	ImageButton
TRadioButton	botão de seleção única em forma de círculo seguido de um rótulo	RadioButton
TRadioGroup	painel com título, bordas e com itens semelhantes ao TRadioButton	RadioGroup, RadioButton, EditText
TCheckBox	caixa de seleção seguido de um rótulo	CheckBox
TComboBox	caixa de texto de única linha e com uma lista de opções	Spinner
TGroupBox	painel ou área de agrupamento com título e bordas	RelativeLayout, EditText
TListBox	caixa de texto de múltiplas linhas com uma lista de opções seguido de uma barra de rolagem	ListView
TLabel	rótulo	EditText
TStaticText	rótulo	EditText
TEdit	caixa de texto com uma única linha	EditText
TMemo	caixa de texto com uma ou mais linhas	EditText
TRichEdit	caixa de texto com uma ou mais linhas	EditText
TDateTimePicker	caixa de texto para preenchimento de data ou hora seguido de um botão	EditText
TStringGrid	tabela composta de linhas e colunas	GridView
TProgressBar	barra de progresso	ProgressBar
TToolBar	barra de ferramentas para agrupamento de botões	RelativeLayout
TToolButton	botão sem rótulo e com figura, somente junto ao componente TToolBar	Button
TScrollBox	painel ou área de agrupamento com barras de rolagem vertical e horizontal	ScrollView, RelativeLayout
TImage	exibe uma imagem	ImageView

Figura 9 – Ilustração da interface Delphi com os componentes convertidos pela ferramenta



3.3 ESPECIFICAÇÃO DA SAÍDA

Os componentes de interface Delphi que compõem o formulário definido pelo usuário, são convertidos para Android em arquivos `.XML` seguindo os padrões de definição do SDK. Além da interface, a ferramenta gera um projeto Java Android com as divisões e diretórios padrões estabelecidos.

Os arquivos gerados são estruturados da seguinte forma:

a) diretório raiz: possui além dos outros diretórios o arquivo `AndroidManifest.xml`, responsável pelas definições, configurações e permissões do projeto gerado;

b) `src`: diretório que possui os pacotes e classes Java, contém a classe `Main` cujo nome é o mesmo do projeto Delphi;

c) `gen`: diretório que possui as classes Java geradas pela SDK, contém principalmente a classe `R.java`, responsável pela ligação entre os arquivos de recurso e as classes Java;

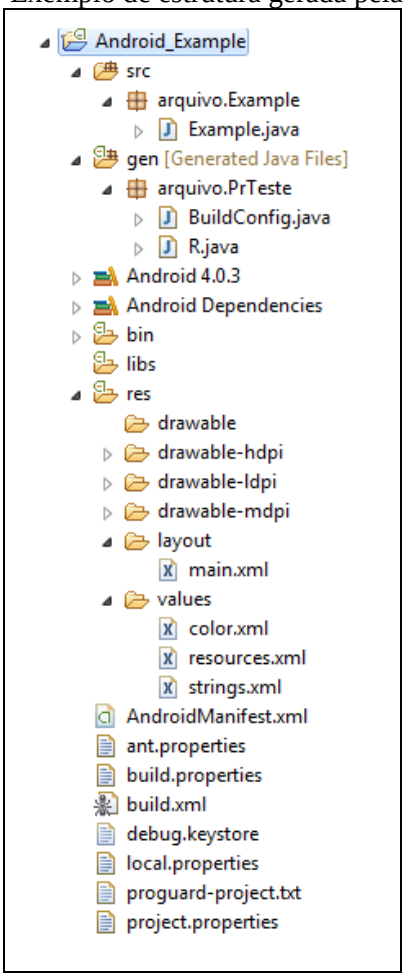
d) `res`: diretório que possui os recursos da aplicação, imagens, cores, definição de interface entre outros arquivos. Esses recursos são distribuídos em outros

subdiretórios:

- `drawable`: contém os arquivos de imagens utilizados nos componentes traduzidos,
- `layout`: contém os arquivos de definição de interface no formato XML, onde está disposto o arquivo resultante da interface convertida `main.xml`,
- `values`: contém os arquivos XML que descrevem cores, estilos, texto pré-definidos entre outros recursos. Nele estão contidos os arquivos `color.xml`, `resources.xml`.

A partir do projeto gerado ainda é necessário adicionar arquivos de configuração e de chaves para permitir ao SDK a compilação e geração de um arquivo `.apk`. Na Figura 10 é apresentado um exemplo da uma estrutura gerada pela ferramenta.

Figura 10 – Exemplo de estrutura gerada pela ferramenta



3.4 ESPECIFICAÇÃO DA ENTRADA

Como mencionado anteriormente, a ferramenta DelphiToAndroid tem como entrada o formulário de interface Delphi. De forma técnica é um objeto da classe `TForm` pelo qual é obtido a lista de componentes em tempo de execução através do atributo `TForm.Components`.

Através da lista de componentes é feita a varredura dos objetos e inicialmente identificado os componentes ligados ao formulário, aqueles dispostos diretamente sobre o componente `TForm`. A partir de cada componente identificado é feita a tradução deste e uma nova varredura é iniciada, nesta são identificados os componente dispostos diretamente sobre o componente identificado, de forma que durante a construção hierarquica da interface Android, atravéz da leitura e tradução dos componente Delphi, seja gerada a mesma distribuição e sobreposição dos componentes na interface Delphi. Para isso, a rotina funciona de modo recursivo, reutilizando o mesmo método para a tradução de um componente, o qual retorna para o método anterior após a sua execução.

Quando um componente é identificado, este é uma referência para a classe base `TComponent`, esta por sua vez não possui muitos atributos que possam ser acessados nesse nível de classe. Assim quando o componente é identificado pela ferramenta, é possível por meio de *TypeCast* determinar o tipo da classe específica do objeto `TComponent`, e a partir deste ter acesso a todos os atributos que definem o componente, obtendo assim as informações necessárias para a conversão do mesmo. Dessa forma, após a conversão de todos os componentes do formulário, o conteúdo da interface traduzida é armazenado em um atributo da classe responsável pela conversão, a qual irá gerar um novo *layout* Android.

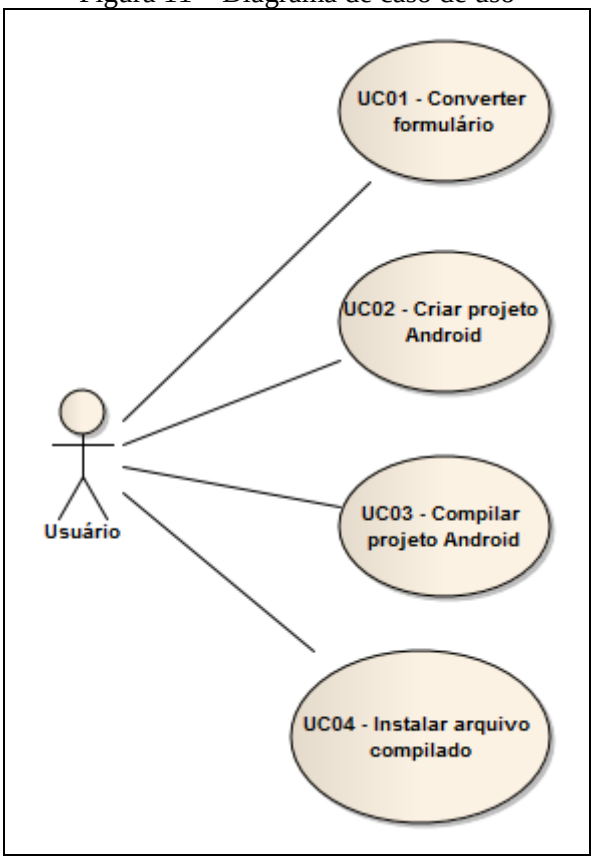
3.5 ESPECIFICAÇÃO

A especificação da ferramenta DelphiToAndroid foi gerada com o software Enterprise Architect, utilizando conceitos de orientação a objetos e com base nos diagramas UML. Gerou-se como resultado os diagramas de caso de uso, classe e sequência, apresentados nos tópicos subsequentes.

3.5.1 Diagrama de caso de uso

A Figura 11 mostra os diagramas de caso de uso da ferramenta e no Quatro 11 é apresentado o detalhamento dos mesmos.

Figura 11 – Diagrama de caso de uso



Os casos de uso identificado como UC02, UC03 e UC04 somente podem ser executados após a execução do caso de uso identificado como UC01.

Quadro 11 – Detalhamento dos casos de uso

UC01 – Converter formulário

Pré-condições: Utilizar um formulário que herde de `TFrmTemplate`.
O projeto deve possuir apenas um formulário sendo convertido.

Cenário principal:
1. O usuário cria um novo projeto Delphi.
2. O usuário cria um novo formulário herdando de `TFrmTemplate`.
3. O usuário desenvolve a interface.
4. O usuário executa o projeto a partir do Delphi.
5. A ferramenta faz a conversão dos componentes e gera os arquivos convertidos.

Exceção 01:
Componente não identificado
No passo 5, a ferramenta não identifica um componente do formulário e portanto a tradução é interrompida.

Pós-condições: Arquivos de recurso e de definição da interface convertida gerados.

UC02 – Criar projeto Android

Pré-condições: Execução do caso de uso UC01.

Cenário principal:
1. A ferramenta cria um projeto Android no diretório atual do projeto Delphi.
2. A ferramenta atualiza os arquivos de recursos e de definição de interface.

Pós-condições: Um esquema de pastas e os arquivos necessários para um projeto Android gerados.

UC03 – Compilar projeto Android

Pré-condições: Execução do caso de uso UC02.

Cenário principal:
1. A ferramenta identifica o local do projeto criado.
2. A ferramenta gera as chaves para assinatura do arquivo compilado.
3. A ferramenta faz a compilação do projeto utilizando as ferramentas do SDK Android.

Pós-condições: Arquivo compilado e assinado com extensão `.apk`.

UC04 – Instalar arquivo compilado

Pré-condições: Execução do caso de uso UC03.

Cenário principal:
1. A ferramenta identifica o local do arquivo compilado.
2. A ferramenta instala o arquivo compilado em um dispositivo ou máquina virtual.

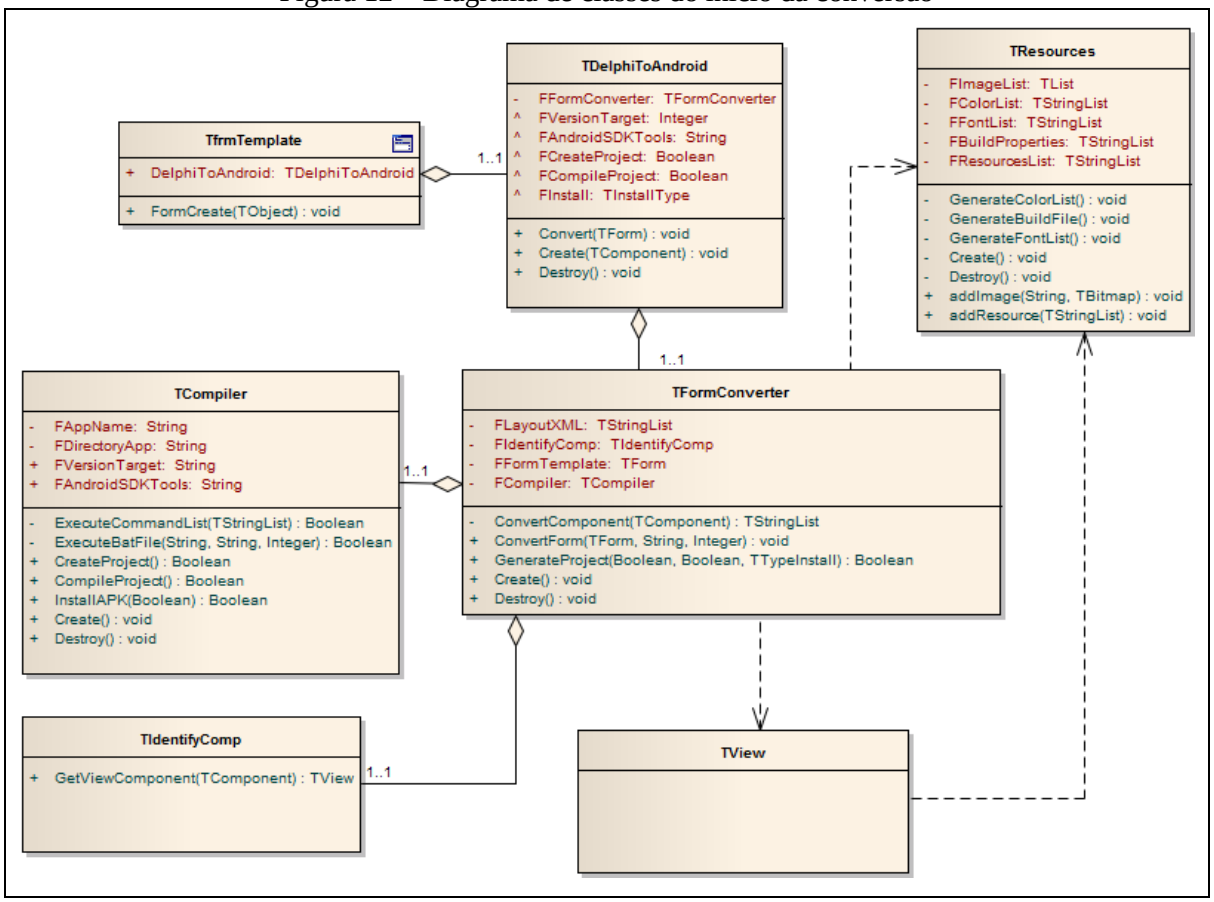
Exceção 01:
Nenhum dispositivo identificado.
No passo 2, a ferramenta não identifica nenhum dispositivo e, portanto, a instalação é interrompida.

Pós-condições: Aplicação instalada no dispositivo ou máquina virtual.

3.5.2 Diagrama de classes

Nessa seção são apresentadas as classes que compõem a estrutura da ferramenta, seus relacionamentos e propriedades. O diagrama de classes apresentado na Figura 12 descreve as classes responsáveis pelo início da conversão da interface.

Figura 12 – Diagrama de classes do início da conversão



A classe `TFrmTemplate` contém o formulário base para criação da interface dimensionado no padrão *Wide Video Graphic Array* (WVGA) (480x800). Este por sua vez contém um componente `DelphiToAndroid` responsável pela conversão do formulário, o qual está devidamente configurado. O construtor desta classe inicia a conversão através do componente utilizando o método `Convert`. A partir do método `Convert` é instanciando um objeto da classe `TFormConverter` e invocado o método `ConvertForm`, que recebe por parâmetro o próprio objeto do formulário.

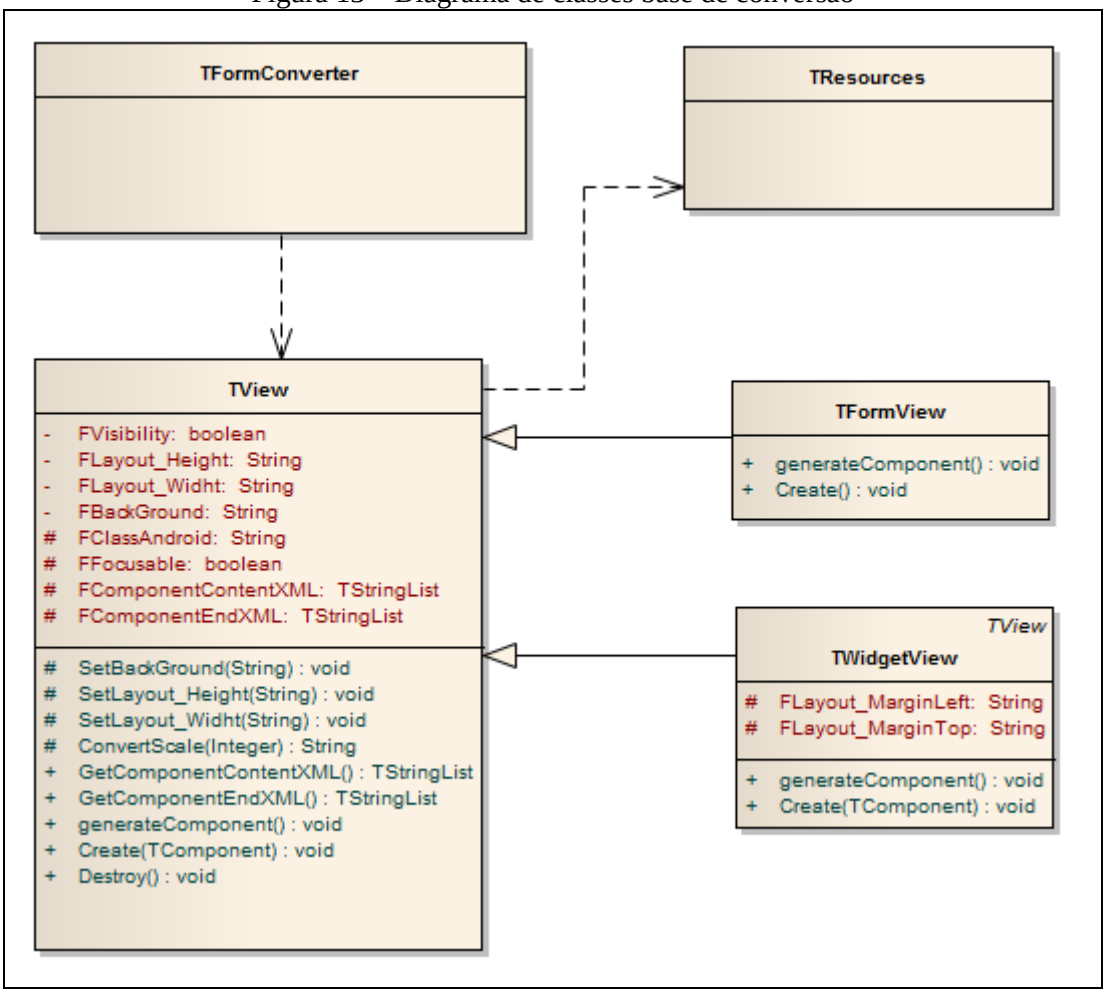
A classe `TFormConverter` é responsável por gerenciar toda a rotina de conversão do projeto, ao qual inicia a partir do método `ConvertForm`. Neste são inicializados os atributos: `FFormTemplate`, com o parâmetro recebido pelo método, e `FLayoutXML`, com a especificação base de uma interface Android. No método `ConvertForm` são identificados os componentes dispostos diretamente no formulário e convertidos por meio do método `ConvertComponent`, o

qual utiliza a classe `TIdentifyComp` para identificação do componente e retorna o valor convertido para o atributo `FLayoutXML`.

Após a conversão dos componentes são invocados os métodos responsáveis pela criação e compilação dos arquivos resultantes no projeto Android, presentes na classe `TCompiler`, e os métodos de geração e gravação dos arquivos de recurso obtidos através da conversão dos componentes e necessários para geração do projeto Android, presentes na classe `TResources`.

A classe `TResourses` implementa o padrão de projeto *singleton*, garantindo assim que esta terá somente uma única instância de objeto e que todas a classes que se comunicam com esta terão acesso ao mesmo objeto. Dessa forma a classe `TView` e as classes que herdam desta utilizam os métodos `addImage` e `addResource` da classe `TResources` para armazenar os recursos obtidos durante a conversão de determinado componente. Na Figura 13 é apresentado o diagrama de classes com as classes base de conversão dos componentes.

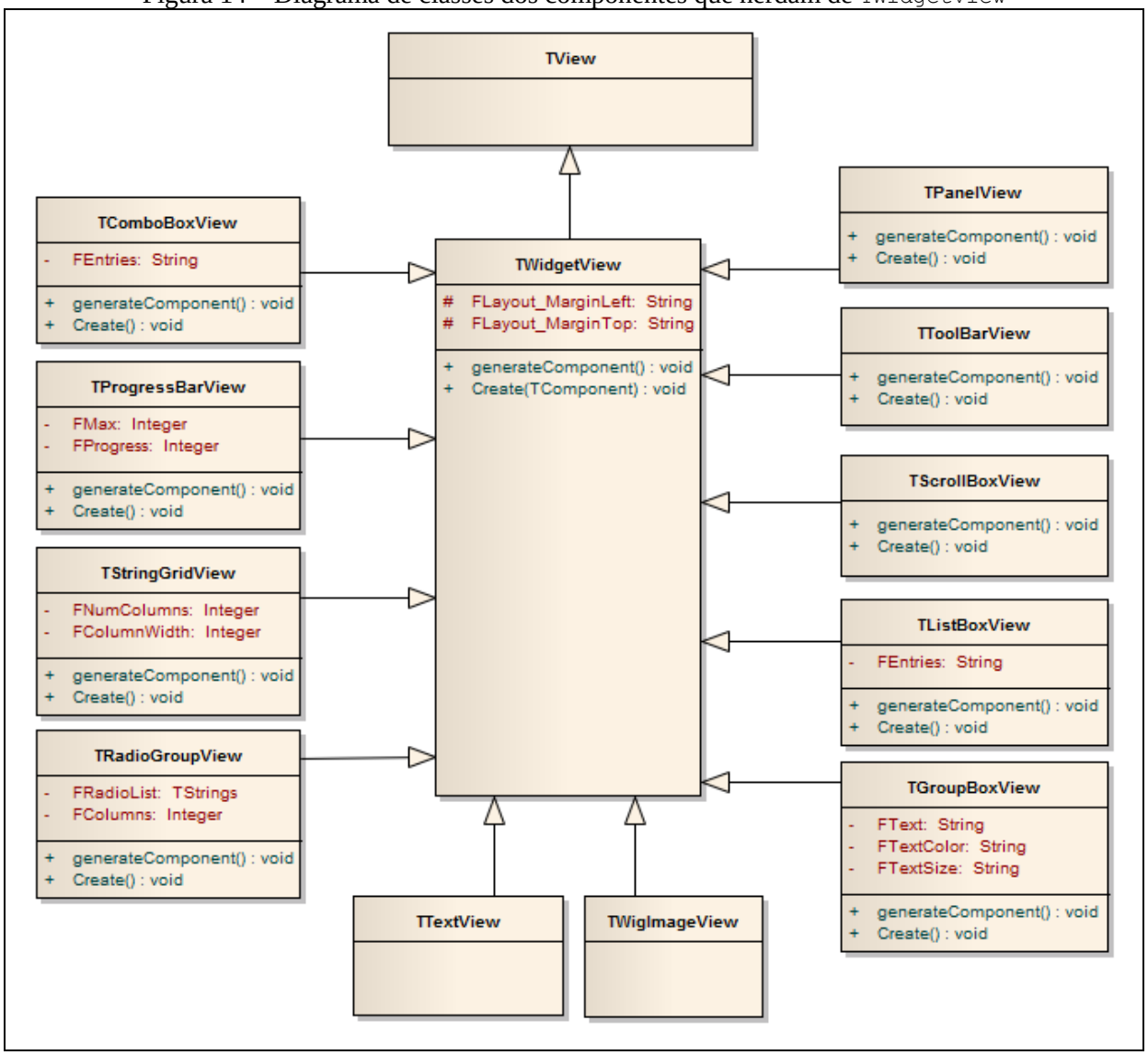
Figura 13 – Diagrama de classes base de conversão



Na classe `TView` estão dispostos os atributos bases de um componente de interface Android, como: visibilidade, tamanho, cor ou imagem de fundo, nome da classe e foco. O

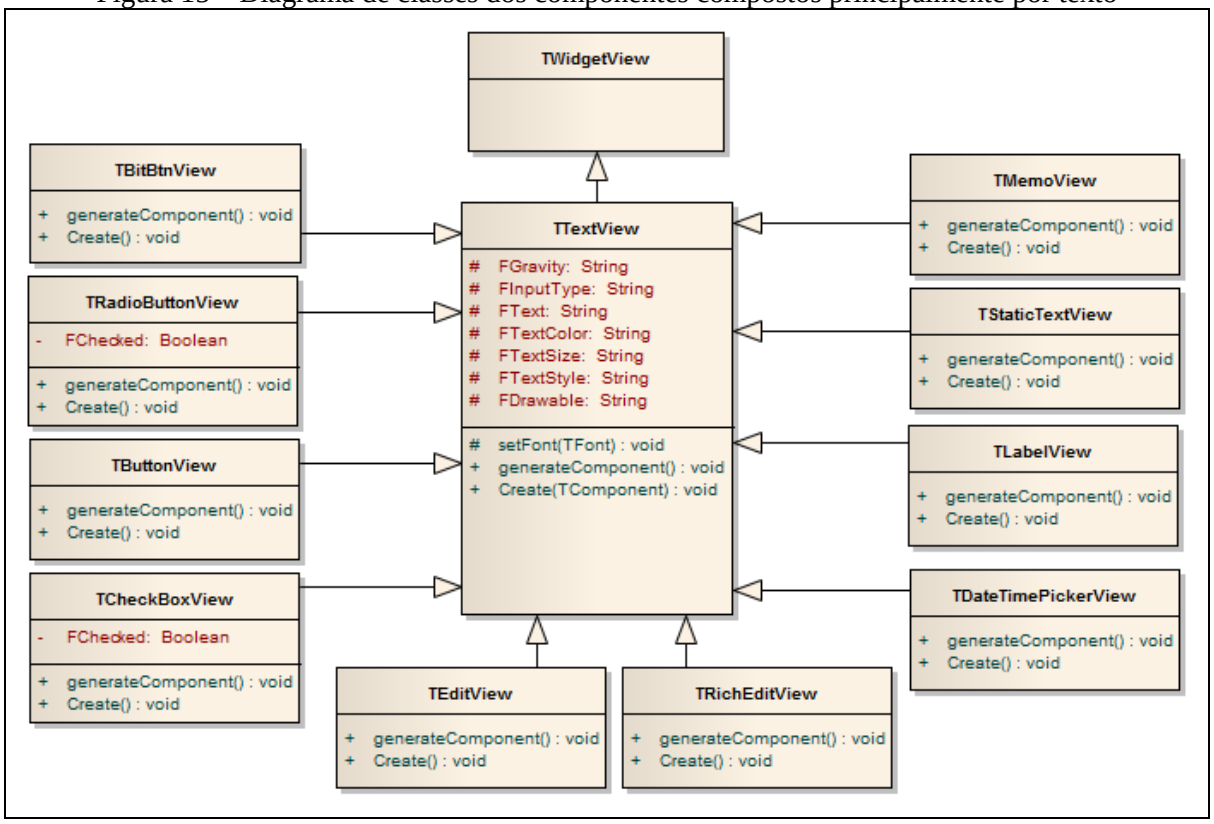
atributo `FClassAndroid` determina a *tag* do componente no XML de interface, que é definido nas classes filhas da classe `TView`, assim como outros atributos. No método construtor os atributos correspondentes a cada classe são inicializados ou alterados com as definições do objeto `TComponent` correspondente ao parâmetro do componente convertido. Cada classe filha interpreta esse objeto como um tipo específico obtendo assim os valores necessários para preenchimento dos atributos e dos recursos adicionados a classe `TResources`. Após a chamada do construtor os atributos de definição são inicializados e por meio do método `generateComponent` o componente Android é gerado e armazenado nos atributos `FComponentContentXML` e `FComponentEndXML`. Assim como o construtor, o método `generateComponent` é implementado por toda classe filha, que por sua vez completa a construção do componente seguindo a definição específica do mesmo. A classe `TFormView` gera o componente formulário base para a interface. A classe `TWidgetView` possui dois atributos, `FLayout_MarginLeft` e `FLayout_MarginTop`, responsáveis pela disposição do componente em tela. Na Figura 14 é apresentado o diagrama de classes com alguns dos componentes de conversão final filhos da classe `TWidgetView`.

Figura 14 – Diagrama de classes dos componentes que herdam de `TWidgetView`



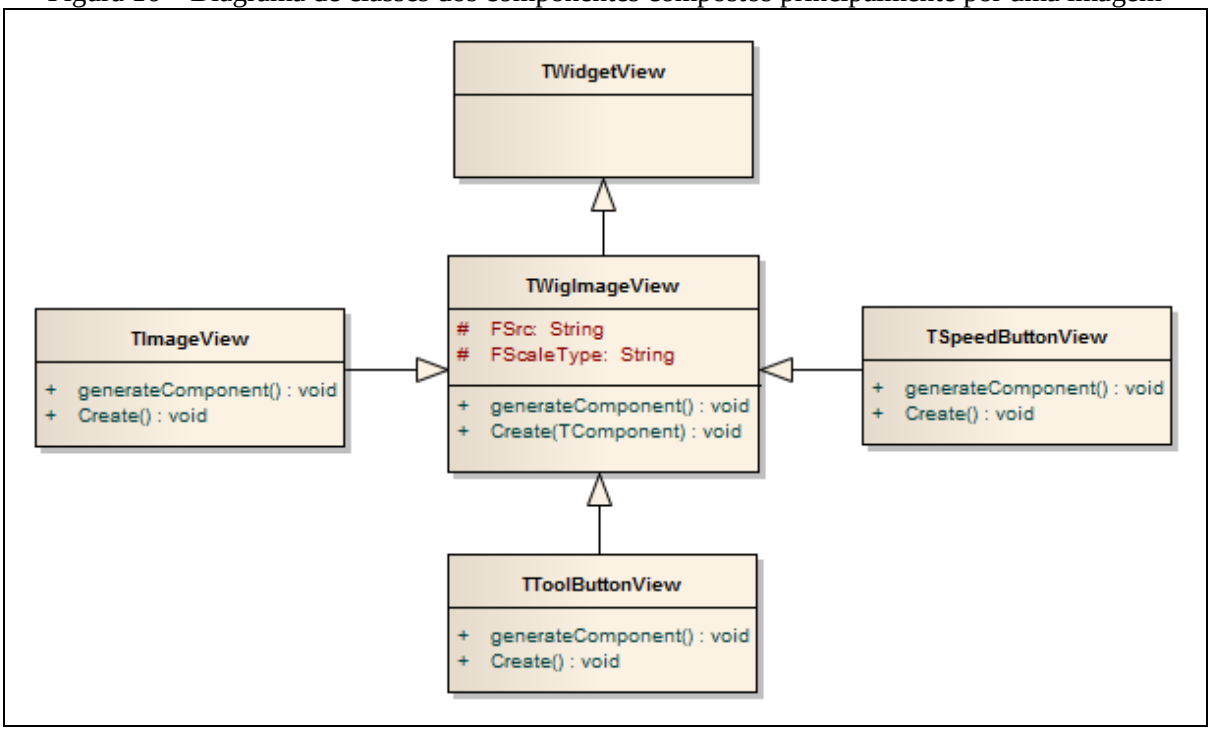
A classe `TWidgetView` representa o componente de exibição base. A partir dele são definidas várias classes de conversão de componentes, onde cada uma possui sua peculiaridade ou condições de montagem do componente. As classes `TTextView` e `TWigImagemView` geram ainda dois agrupamentos de herança apresentados nos diagramas de classes das figuras 15 e 16, respectivamente.

Figura 15 – Diagrama de classes dos componentes compostos principalmente por texto



A classe `TTextView` define atributos comuns a componentes que possuem dados textuais. O método `setFont` implementado por essa classe inicializa os atributos `TTextColor`, `TTextSize` e `TTextSyte` de acordo com o objeto `TForm` recebido por parâmetro, o qual é comum aos componentes representados pelas classes.

Figura 16 – Diagrama de classes dos componentes compostos principalmente por uma imagem

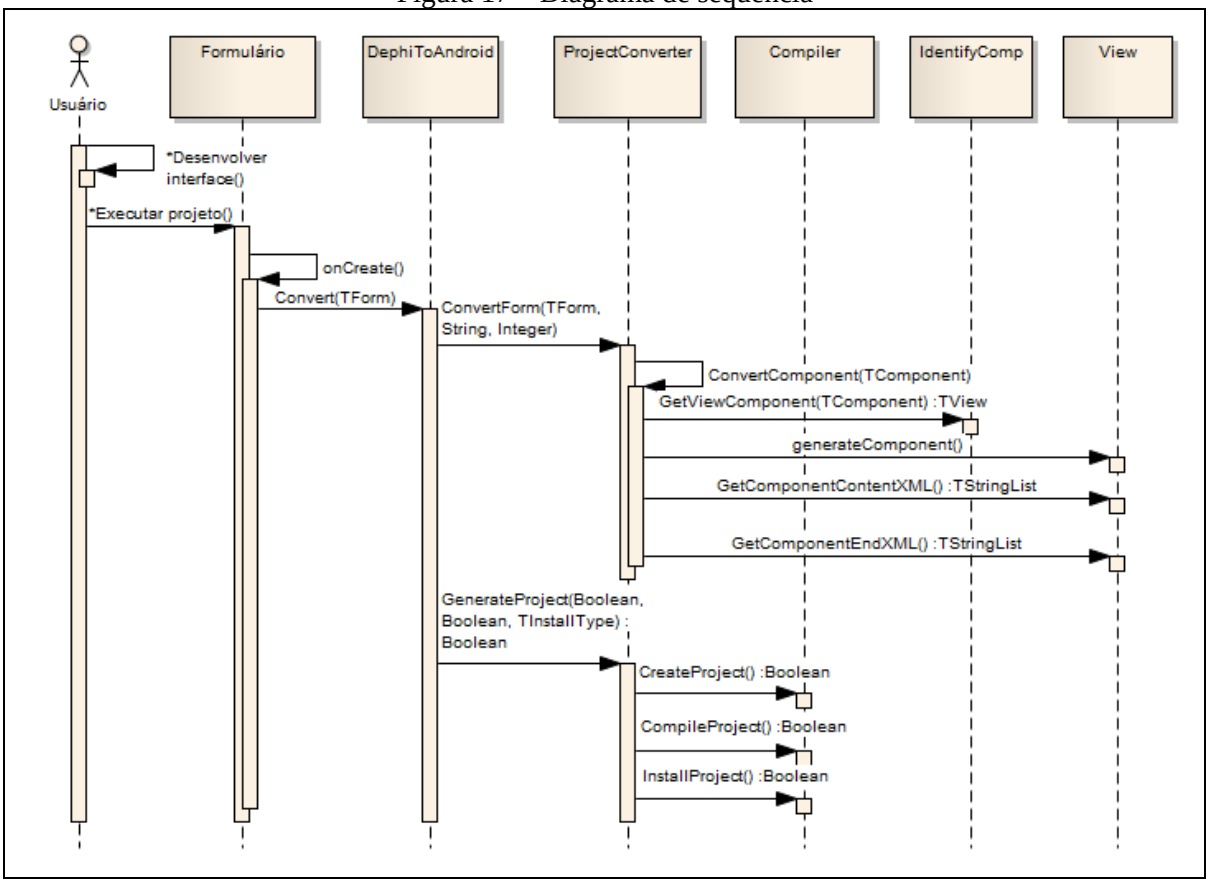


A classe `TWigImageView` define atributos comuns a componentes que possuem imagens em sua composição.

3.5.3 Diagrama de sequência

O diagrama de sequência da Figura 17 ilustra a interação do `Usuário` com a ferramenta DelphiToAndroid, nas rotinas de conversão de interface e de geração, compilação e instalação do projeto Android gerado.

Figura 17 – Diagrama de sequência



3.6 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas, bem como a implementação da ferramenta.

3.6.1 Técnicas e ferramentas utilizadas

O desenvolvimento da ferramenta foi efetuado na linguagem *Object Pascal*. O ambiente de desenvolvimento utilizado para o desenvolvimento da ferramenta foi o Delphi

versão 7.0. Ainda no desenvolvimento da ferramenta foi utilizado o Android SDK para geração do projeto Android. A interface convertida assim como o projeto Android gerado foram testados utilizando o IDE Eclipse versão Indigo junto com o Android SDK e o Android *Development Tools* (ADT) *for* Eclipse.

3.6.2 Implementação da ferramenta

A implementação da ferramenta iniciou-se na pesquisa da geração da interface Android, quanto aos componentes, formatação do arquivo XML e as *tags* interpretadas pelo SDK Android. Após a pesquisa foram determinados os componentes de interface Delphi a serem convertidos e portanto as classes responsáveis pela conversão. Estas formam uma hierarquia de classes compostas de atributos que por sua vez determinam os valores das *tags* resultante no componente convertido.

A análise da entrada inicia na classe `TFormConverter` através da chamada do métododo `ConvertForm`. Este método inicializa o atributo `FLayoutXML`, responsável por armazenar os componentes convertidos, e em seguida faz a identificação e conversão do objeto formulário `TForm`, criando assim a estrutura base da interface convertida. Em seguida a rotina efetua uma varredura no objeto `TForm` identificando os componentes diretamente ligados ao formulário e que serão convertidos através da chamada do método `ConvertComponent`. O Quadro 12 mostra um trecho do código citado.

Quadro 12- Código fonte do método `ConvertForm` da classe `TFormConverter`

```
procedure TFormConverter.ConvertForm(pForm:TForm; pAndroidSDKTools:String;
pVersionTarget:Integer = 5);
...
  try
    FLayoutXML.Clear;
    FLayoutXML.Add(Layout_Start);

    oViewComponent := FIdentifyComp.GetViewComponent(FFormTemplate);
    oViewComponent.generateComponent;

    FLayoutXML.Add(oViewComponent.GetComponentContentXML.Text);

    for i := 0 to FFormTemplate.ComponentCount-1 do
    begin

      oXMLComponent := nil;
      oComponent    := FFormTemplate.Components[i];

      if(TControl(oComponent).Parent = FFormTemplate)then
        oXMLComponent := ConvertComponent(oComponent);

      if(oXMLComponent <> nil) and (oComponent <> nil)then
        FLayoutXML.Add(oXMLComponent.Text);

    end;

  finally
    FLayoutXML.Add(oViewComponent.GetComponentEndXML.Text);
    FLayoutXML.Add(Layout_End);
  end;
end;
```

Inicialmente no método `ConvertComponent` (Quadro 13), é feita a identificação do componente convertido por meio do método `GetViewComponent` da classe `TIdentifyComp`, o qual retorna um objeto filho da classe `TView` que por sua vez faz a conversão. Após a identificação e conversão do componente, o conteúdo correspondente ao componente gerado é armazenado no atributo `oXMLComponent`. A partir do objeto componente obtido através dos parâmetros do método, é feita uma varredura no formulário identificando componentes diretamente ligados a este. Quando um componente é identificado, o método `ConvertComponent` é invocado recursivamente e a rotina de conversão é repetida. Após o fim da sua execução é retornado pela mesma o atributo `oXMLComponent` com o conteúdo do componente convertido.

Quadro 13 - Código fonte do método `ConvertComponent` da classe `TFormConverter`

```
function TFormConverter.ConvertComponent(pComp: TComponent):TStringList;
...
  try
  ...
    oViewComponent := FIdentifyComp.GetViewComponent(pComp);
    ...
    oViewComponent.generateComponent;
    oViewComponent.GetComponentContentXML;
    ...
    oXMLComponent  := TStringList.Create;
    oXMLComponent.Add(oViewComponent.GetComponentContentXML.Text);

    for i := 0 to TControl(FFormTemplate).ComponentCount-1 do
    begin

      oXMLCompChild   := nil;
      if(TControl(TControl(FFormTemplate).Components[i]).Parent = pComp)then
        oXMLCompChild := ConvertComponent(TControl(FFormTemplate).Components[i]);

      if(oXMLCompChild <> nil)then
      begin
        oXMLComponent.Add(oXMLCompChild.Text);
        FreeAndNil(oXMLCompChild);
      end;

    end;

    oXMLComponent.Add(oViewComponent.GetComponentEndXML.Text);
    ...
  finally
    FreeAndNil(oViewComponent);
    if(oXMLCompChild <> nil)then
      FreeAndNil(oXMLCompChild);

    result := oXMLComponent;
  end;
  ...
end;
```

O método `GetViewComponent` (Quadro 14) da classe `TIdentifyComp` é responsável por identificar o componente Delphi e determinar a classe conversora do mesmo. A partir do objeto obtido nos parâmetros do método, é utilizada a propriedade `ClassName` do mesmo para o identificar e classificar. Uma vez classificado, este instancia um objeto filho da classe `TView` que será retornado pelo método e responsável pela tradução específica do componente.

Quadro 14 - Código fonte do método `GetViewComponent` da classe `TIdentifyComp`

```
function TIdentifyComp.GetViewComponent(pComp: TComponent): TView;
...
  Try
    ...
    case (AnsiIndexStr(pComp.ClassName,['TForm', 'TPanel', 'TBitBtn', 'TButton',
'TSpeedButton', 'TRadioButton', 'TRadioGroup', 'TCheckBox', 'TComboBox',
'TGroupBox', 'TListBox', 'TLabel', 'TStaticText', 'TEdit', 'TMemo', 'TRichEdit',
'TDateTimePicker', 'TStringGrid', 'TProgressBar', 'TToolBar', 'TScrollBox',
'TImage', 'TToolButton'])) of
      0  : oComp := TFormView.Create(pComp);
      1  : oComp := TPanelView.Create(pComp);
      2  : oComp := TBitBtnView.Create(pComp);
      ...
      10 : oComp := TListBoxView.Create(pComp);
      11 : oComp := TLabelView.Create(pComp);
      12 : oComp := TStaticTextView.Create(pComp);
      ...
      20 : oComp := TScrollBoxView.Create(pComp);
      21 : oComp := TImageView.Create(pComp);
      22 : oComp := TToolButtonView.Create(pComp);
      ...
    end;
    ...
    result := oComp;
    ...
  except
    ...
  end;
end;
```

Quando um objeto da classe `TView` é criado, ele recebe como parâmetro um objeto `TComponent` com o componente Delphi que está sendo convertido. A partir desse objeto são extraídas as características necessárias para gerar o componente correspondente no Android. As informações extraídas são armazenadas em atributos, os quais podem ser alterados nas classes filhas, e utilizados na geração do componente Android pelo método `generateComponent`. Assim como o método construtor, este pode ser sobrescrito pelas classes filhas, as quais adicionarão características e recursos diferentes a cada tipo de componente. No Quadro 15 é mostrado um trecho dos métodos construtor e `generateComponen` da classe `TView`.

Quadro 15 - Código fonte dos métodos Create e `generateComponent` da classe `TView`

```
constructor TView.Create(pComp: TComponent);
  ...
  FComponentContentXML    := TStringList.Create;
  FComponentEndXML        := TStringList.Create;
  ...
  FID             := AnsiLowerCase(pComp.Name);
  FClickable      := TControl(pComp).Enabled;
  ...
  FVisibility     := TControl(pComp).Visible;
  ...
end;
...
procedure TView.generateComponent;
...
  with (FComponentContentXML) do
  begin
    Add(Format('<%s',[FClassAndroid]));
    Add(Format('  android:id="@+id/%s"',[FID]));
    Add(Format('  android:layout_width="%s"',[FLayout_Widht]));
    Add(Format('  android:layout_height="%s"',[FLayout_Height]));
    ...
  end;
  ...
  FComponentEndXML.Add(Format('</%s>',[FClassAndroid]));
  ...
end;
```

A tradução de cada componente Delphi para o Android difere devido aos atributos e características que o compõe. Alguns atributos devem ser armazenados de forma difenciada, como no caso de imagens ou listas de itens pré-definidos. A classe `TResources` é responsável por armazenar estes atributos. Ela implementa um *singleton* que permite que todas classes obtenham a mesma instância dela e mantenham as listas constantemente atualizadas. As imagens são armazenadas em um atributo do tipo `TList`, o qual armazena uma lista de ponteiros do tipo `TImageComp`, enquanto as listas são armazenadas em um atributo `TStringList`. Os métodos responsáveis pela alimentação dessas listas são `addImage` e `addResource`, ambos mostrados no Quadro 16.

Quadro 16 - Código fonte dos métodos `addImage` e `addResource` da classe `TResources`

```
procedure TResources.addImage(pName: String; pImage: TBitmap);
...
  New(oImageComp);
  oImageComp^.ImageName := pName;
  oImageComp^.ImageObj  := pImage;
  FImageList.Add(oImageComp);
end;
...
procedure TResources.addResource(pName:String; pResource:TStrings);
...
  with FResourcesList do
  begin
    Add(Format(' <string-array name="%s">',[pName]));
    for i := 0 to pResource.Count-1 do
      Add(Format('  <item>%s</item>',[pResource.Strings[i]]));
    Add(' </string-array>');
    ...
  end;
end;
```

Com os dados necessários armazenados, a classe `TCompiler` é responsável por gerar o projeto Android com a chamada do método `createProject`. Após a geração do projeto, os arquivos armazenados correspondentes à tradução da interface são adicionados ao projeto criado e então é invocado o método `compileProject`. Este efetua a compilação do projeto gerando a aplicação Android e criando um arquivo `.apk`, que pode ser instalado em um dispositivo Android conectado ao computador ou em uma máquina virtual através da chamada do método `installAPK`. A criação, compilação e instalação do projeto são feitos através de linhas de comando utilizando as ferramentas disponibilizadas pela SDK Android como ilustrado nos Quadros 17, 18 e 19.

Quadro 17 - Código fonte dos métodos `createProject` da classe `TCompiler`

```
function TCompiler.createProject: Boolean;
...
  Try
    ...
    oCommands.Add( Format('MKDIR "%s"',[FDirectoryApp]));
    oCommands.Add( Format('CD %s',[FAndroidSDKTools]));
    oCommands.Add( Format('android create project --name "%s" --target %s --path
"%s" --activity %s --package "arquivo.%s"', [FAppName, FVersionTarget,
FDirectoryApp, FAppName, FAppName]));
    ...
  finally
    ...
  end;
end;
```

Quadro 18 - Código fonte dos métodos `compileProject` da classe `TCompiler`

```
function TCompiler.compileProject: Boolean;
...
  try
    ...
    oCommands.Add( Format( 'CD %s',[FDirectoryApp]));
    if not(FileExists(FDirectoryApp+'\debug.keystore'))then
    begin
      createAntFile(FdirectoryApp);
      oCommands.Add('keytool -genkey -keyalg RSA -alias debug -keystore
debug.keystore -storepass 123456 -validity 10000 -keysize 2048 -dname "CN=DOUGLAS,
OU=JULIO, O=REZINI, L=BLUMENAU, S=SANTA CATARINA, C=BR" -keypass 123456');
    end;
    oCommands.Add('ant release');
    ...
  finally
    ...
  end;
end;
```

Quadro 19 - Código fonte dos métodos `installAPK` da classe `TCompiler`

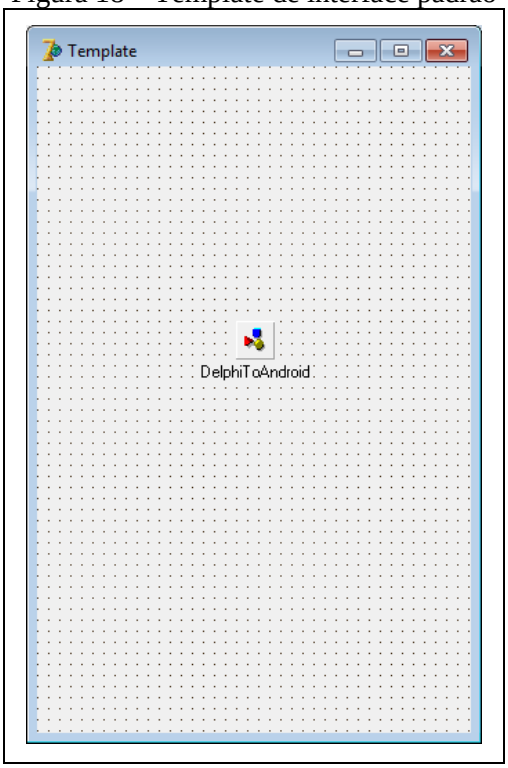
```
function TCompiler.installAPK: Boolean;
...
  try
    ...
    oCommands.Add( Format( 'CD %s',[FDirectoryApp]));
    oCommands.Add('CD ..\');
    oCommands.Add('CD platform-tools');
    ...
    if(FInstallOnDevice)then
      oCommands.Add( Format('adb -d install "%s\bin\%s-
release.apk"',[FDirectoryApp, FAppName]))
    else
      oCommands.Add( Format('adb install "%s\bin\%s-release.apk"',[FDirectoryApp,
FAppName]));
    ...
  finally
    ...
  end;

end;
```

3.6.3 Operacionalidade da implementação

Nesta seção é apresentado o funcionamento da ferramenta DelphiToAndroid. Esta é aplicada na forma de um componente Delphi, que por sua vez trabalha em conjunto com um formulário *template*, o qual determina o tamanho de uma interface padrão do dispositivo Android. A Figura 18 mostra o formulário com o componente.

Figura 18 – Template de interface padrão



Para que o componente faça a geração do projeto Android, é necessário configurar as seguintes propriedades:

a) `AndroidSDKTools`: diretório em que o Android SDK está instalado;

b) `CompileProject`: determina se o projeto será compilado após a conversão;

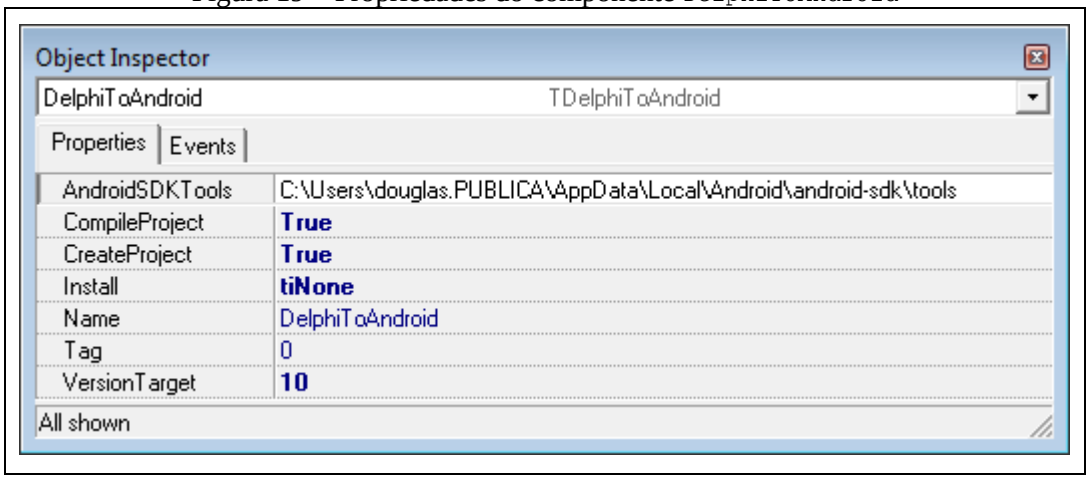
c) `CreateProject`: determina se a ferramenta irá criar um projeto Java Android com a interface convertida;

d) `Install`: determina se a aplicação será instalada em um dispositivo conectado ao computador (`tiDevice`) ou a uma máquina virtual (`tiVirtual`) ou não será instalada (`tiNone`);

e) `VersionTarget`: determina a versão do Android que o projeto será criado.

A Figura 19 mostra as propriedades do componente citadas acima e um exemplo do preenchimento destas.

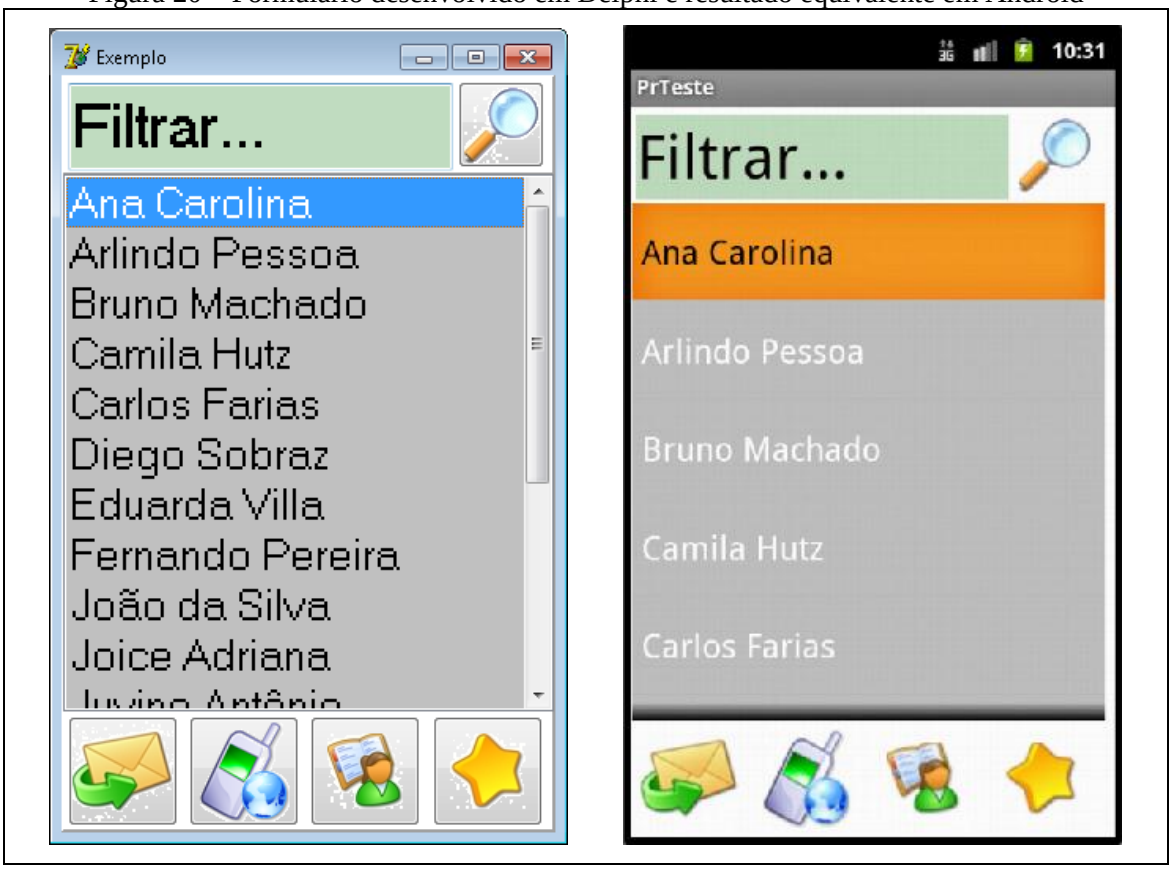
Figura 19 – Propriedades do componente `DelphiToAndroid`



Após a configuração do componente, o usuário efetua o desenvolvimento da interface utilizando o formulário *template* ou um formulário que herde deste, de forma a garantir que a dimenção do formulário seja equivalente ao padrão WVGA (480x800). Contudo a ferramente permite a conversão de qualquer formulário Delphi, desde que o componente `DelphiToAndroid` esteja devidamente configurado e o método `Convert(Self)` seja executado no evento `FormCreate` do formulário.

Com a interface desenvolvida, o usuário deve executar o programa ativando o botão *run* do ambiente Delphi. Dessa forma a ferramenta fará a conversão do formulário e a geração do projeto Android com uma interface equivalente, o qual por sua vez é compilado e instalado em uma máquina virtual ou dispositivo, conforme as propriedades do componente. A Figura 20 ilustra uma interface desenvolvida a partir do *template* em Delphi (esquerda) e o resultado obtido na conversão exibido em uma máquina virtual (direita).

Figura 20 – Formulário desenvolvido em Delphi e resultado equivalente em Android

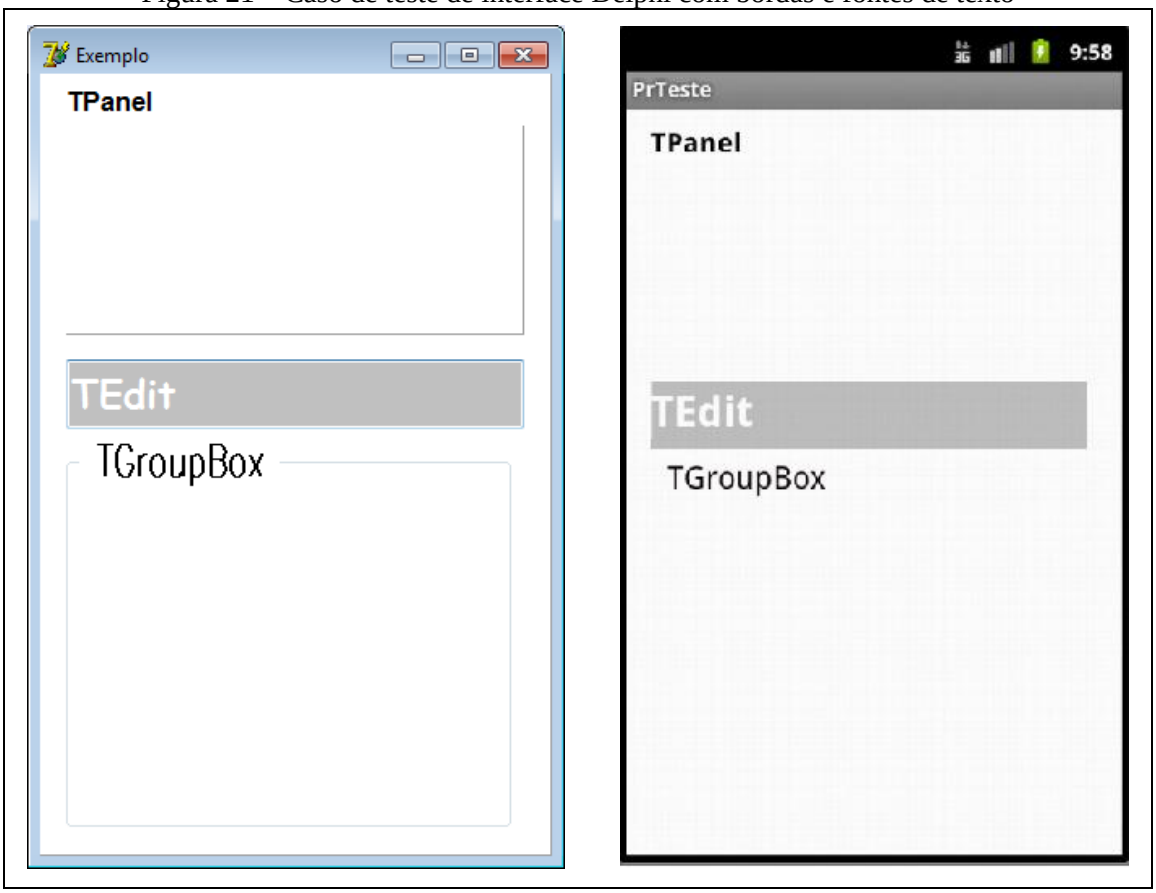


3.7 RESULTADOS E DISCUSSÃO

Este trabalho trouxe como principal desafio a utilização do ambiente de desenvolvimento Delphi para a geração de interface de usuário em Android. Durante o desenvolvimento do trabalho, esse foco foi ampliado para que além de criar a interface pelo ambiente, fosse possível também criar um projeto, compilá-lo e instalar o arquivo compilado em um dispositivo ou máquina virtual Android, os quais foram alcançados e geraram três novos requisitos funcionais.

Quanto à conversão dos componentes, foram identificadas algumas características comuns em componentes Delphi que não foram possíveis replicar nos componentes Android. Uma destas é a aplicação de bordas - essa propriedade não existe nos componentes Android. As bordas somente marcam a área destes e por sua vez não exibem a linha comum dos componentes de interface Delphi. Outra característica que se perde na conversão é a utilização de fontes de texto, os quais são aplicados aos componentes Android principalmente por meio da chamada de eventos nas classes Java. A Figura 21 apresenta uma interface Delphi com as características de bordas e fontes de texto aplicadas, imagem à esquerda, e o resultado obtido na conversão exibido em uma máquina virtual, imagem à direita.

Figura 21 – Caso de teste de interface Delphi com bordas e fontes de texto



Quanto à compatibilidade entre o comportamento da interface Delphi e da interface Android gerada, exceto pelas características de fonte e de bordas, os componentes convertidos em sua maioria possuem uma aparência e comportamento similar ou muito similar com os componentes Android utilizados na conversão, tornando assim a interface gerada semelhante à interface convertida.

Ao comparar a ferramenta DelphiToAndroid com as ferramentas Delphi2Java-II (SILVEIRA, 2006), DelphiToGTK+ (ZIMMERMANN, 2011) e DelphiToWeb (SOUZA, 2005) pode-se destacar como principal diferença entre estas a forma de obtenção dos dados e consequentemente a forma de execução da ferramenta. Enquanto a ferramenta DelphiToAndroid é executada a partir do ambiente Delphi na forma de um componente, as ferramentas Delphi2Java-II, DelphiToGTK+ e DelphiToWeb fazem a conversão a partir de um executável externo, o qual utiliza como entrada a importação de arquivos de definição de interface `.dfm`.

No que se refere à forma de análise de entrada, a ferramenta DelphiToAndroid faz a identificação dos componentes e obtenção dos seus atributos em tempo de execução, através da leitura dos componentes do formulário. As ferramentas Delphi2Java-II, DelphiToGTK+ e

DelphiToWeb recuperam as informações de interface a partir de analisadores léxico, sintático e semântico.

O Quadro 20 apresenta uma comparação entre as ferramentas, levando em consideração as principais características da ferramenta DelphiToAndroid.

Quadro 20 – Comparativo entre as ferramentas

Característica	**Delphi2Java-II (SILVEIRA, 2006)**	**DelphiToGTK+**	**DelphiToWeb**	**DelphiToAndroid**
linguagem de programação	Java	Pascal/Object Pascal	Java	Delphi/Object Pascal
conversão de componentes de visualização	22	25	22	23
uso de analisadores (léxico, sintático e semântico)	Sim	Sim	Sim	Não
conversão de componentes em tempo de execução a partir da IDE	Não	Não	Não	Sim
cria, compila e instala um projeto produto da interface convertida	Parcialmente	Não	Não	Sim
conversão de componentes para acesso a banco de dados	Sim	Não	Não	Não

3.7.1 Problemas e limitações

Embora todos os objetivos propostos para este trabalho tenham sido atingidos, a ferramenta gerada possui alguns problemas e limitações:

a) conversão de interface se limita a somente um formulário por vez;
b) embora o tamanho e o posicionamento dos componentes sejam semelhantes, o tamanho padrão dos componentes Android difere muito dos componentes Delphi, fazendo com que estes fiquem cortados quando a sua definição de tamanho não alcança o valor padrão;
c) devido a grande diversidade de tamanho de interface para dispositivos Android, ficou definido na geração do projeto o padrão WVGA (480x800);
d) definição de cores está limitada às constantes padrões do Delphi, devido à

necessidade de conversão destas de *Red Green Blue* (RGB) para *Red Green Blue Alpha* (RGBA);

e) definição de fontes de texto também não são traduzidas.

4 CONCLUSÕES

Com a popularização dos *smartphones* e *tablets* que rodam principalmente com a plataforma Android, o desenvolvimento de aplicativos se torna cada vez mais atrativo para novos e experientes desenvolvedores. Para estes uma mudança de tecnologia muitas vezes faz com que o desenvolvedor perca no processo de desenvolvimento a agilidade e facilidades antes obtidas em seu ambiente original. Pode ser destacada também a questão da necessidade de migração das aplicações *desktop* para *mobile*, onde a semelhança da interface visual se torna às vezes uma exigência dos usuários. Em vista de tais argumentos, a ferramenta DelphiToAndroid possibilita que o desenvolvedor da linguagem Delphi continue utilizando sua agilidade e prática no desenvolvimento de interfaces para gerá-las para a plataforma Android, ou até mesmo converter uma interface já desenvolvida.

A ferramenta DelphiToAndroid foi implementada na forma de um componente Delphi, o qual permite que através da sua adição em um formulário possa efetuar a conversão do mesmo. Essa abordagem na forma de conversão foi adotada para possibilitar o uso da IDE Delphi como uma opção no desenvolvimento de interfaces Android. Quanto à forma de conversão da interface, se trata da leitura dos componentes, posicionados no formulário, em tempo de execução. Destaca-se que essa abordagem difere das soluções apresentadas por outras ferramentas estudadas nesse trabalho, ou seja, não usar analisadores léxico, sintático e semântico, e sim converter os componentes a partir de suas propriedades dinâmicas, as quais podem ser acessadas somente enquanto a aplicação está sendo executada.

Todos os objetivos do trabalho foram cumpridos. Foram identificados 23 componentes que possuem um comportamento equivalente em Android, apesar de alguns desses serem compostos por mais de um componente Android, como apresentado na seção 3.2. O grau de compatibilidade entre a interface a ser convertida e a gerada foi satisfatório, sendo esse aspecto melhor detalhado na seção 3.7. Quanto à conversão da interface, deve ser destacado que além do objetivo ser cumprido, ele também foi ampliado para permitir que a interface faça a geração de um projeto Java Android, a compilação do mesmo e por fim a instalação da aplicação gerada em um dispositivo ou máquina virtual Android.

4.1 EXTENSÕES

Como extensões da ferramenta sugere-se:

a) implementar a conversão de mais componentes de interface, como `TPageControl` e `TTabSheet`;
b) efetuar a conversão de mais de um formulário por vez para um mesmo projeto

Android;

c) gerar projetos com tamanhos de interface diferenciados, além do padrão WVGA (480x800);
d) permitir outras configurações de geração do projeto, além das definidas nesse trabalho;
e) efetuar a conversão de componentes de banco de dados; e
f) efetuar a geração de eventos de componentes convertidos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABLESON, Frank. **Introdução ao desenvolvimento do android**. [S.1.], 2009. Disponível em: <http://www.ibm.com/developerworks/br/library/os-android-devel/>. Acesso em: 05 nov. 2012.

ANATEL. **Relatório 2011**. [S.1.], 2012. Disponível em: <http://www.anatel.gov.br/Portal/verificaDocumentos/documento.asp?numeroPublicacao=278637&pub=original&filtro=1&documentoPath=278637.pdf>. Acesso em: 23 ago. 2012.

ANDROID DEVELOPERS. **The developer's guide**, [S.l.], 2013. Disponível em: <http://developer.android.com/guide/ >. Acesso em: 30 mar. 2013.

AQUINO, Juliana F. S. **Plataforma de desenvolvimento para dispositivos móveis**. 2007. 14 f. Monografia (Pós-Graduação em Informática) – Departamento de Informática, Pontífica Universidade Católica do Rio de Janeiro, Rio de Janeiro.

CANTU, Marco. **Dominando o Delphi 7**: a bíblia. Tradução Kátia Aparecida Roque. São Paulo: Pearson Education do Brasil, 2003.

DALGARNO, M. **Frequently asked questions about code generations**. [S.l.], 2006. Disponível em: <http://www.codegeneration.net>. Acesso em: 29 abr. 2006.

FONSECA, Fabricio. **Ferramenta conversora de interfaces gráficas**: Delphi2Java-II. 2005. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FREITAS, Ivonei; SCHEMBERGER, Elder E.; VANI, Ramiro. **Plataforma android.** Paraná, 2009.

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

MEDNIEKS , Zigurd et al. **Programming Android:** second edition. Sebastopol: O'Reilly Media, 2012. Disponível em: <http://oreilly.com/catalog/errata.csp?isbn=9781449316648>. Acesso em: 27 abr. 2013.

MORETTI, João. **Empresas devem investir em mobilidade?** [S.l.], ago 2011. Disponível em: <http://www.ecommercebrasil.com.br/artigos/empresas-devem-investir-em-mobilidade/>. Acesso em: 02 set. 2012.

PETTEY, Christy; GOASDUFF, Laurence. **Gartner says worldwide sales of mobile phones declined 2.3 percent in second quarter of 2012.** Egham, 2012. Disponível em: <http://www.gartner.com/it/page.jsp?id=2120015>. Acesso em: 02 set. 2012.

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.

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

SOUZA, Ariana. **Ferramenta para conversão de formulário Delphi em páginas HTML**. 2005. 69 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WILLS, Michelle. **Rapid application development on Delphi.** [S.1], 2010. Disponível em: <http://www.gather.com/viewArticle.action?articleId=281474978541661>. Acesso em: 02 maio 2013.

ZIMMERMANN, Josimar. **Ferramenta para conversão de interfaces gráficas desenvolvidas em Delphi para a biblioteca GTK+**. 2011. 90 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ZMOGINSKI, Felipe. **Smartphones já são poderosos como PCs**. São Paulo, 2013. Disponível em: <http://info.abril.com.br/noticias/tecnologia-pessoal/smartphones-ja-equivalem-a-pcs-de-2010-10042013-0.shl>. Acesso em: 09 jul. 2013.