colloquium exactarum

FRAMEWORK PARA CONVERSÃO DE APLICATIVOS DELPHI DESKTOP EM APLICATIVOS ANDROID NATIVO

FRAMEWORK FOR CONVERSION OF DELPHI DESKTOP APPLICATIONS FOR ANDROID APPLICATIONS NATIVE

Rodrigo da Silva Riquena[1]; Francisco Assis da Silva[2]; Francisco Virginio Maracci[2]; Mário Augusto Pazoti[2]

Discente da Faculdade de Informática da UNOESTE[1]; Docente da Faculdade de Informática da UNOESTE[2].

RESUMO - Com o uso cada vez maior de dispositivos móveis por empresas e organizações há uma demanda crescente de produção de aplicativos para plataforma móvel. Para algumas empresas, o sucesso nos negócios pode depender de uma aplicação móvel que as aproximem dos clientes ou melhore o desempenho de processos internos. Entretanto, desenvolver softwares para a plataforma móvel é um processo oneroso que consome tempo e recursos. Um *framework* para conversão de aplicativos Delphi Desktop em aplicativos Android nativo de forma automática constitui uma ferramenta útil para arquitetos e desenvolvedores de softwares, podendo auxiliar na fase de implementação do aplicativo. Diante disso, esse trabalho baseia-se em métodos e processos de reengenharia de software como o PRE/OO (Processo de Reengenharia Orientada a Objetos), para a conversão automática de um aplicativo desenvolvido no ambiente Delphi em um aplicativo para plataforma móvel Android. Por fim, foi realizado um experimento com um caso real para comprovação dos objetivos.

Palavras-chave: Delphi; Android; Tecnologia Móvel; Reengenharia; Engenharia Reversa; Tradução de Programa.

ABSTRACT - With the growing use of mobile devices by companies and organizations there is an increasing demand applications in production mobile platform. For certain companies, business success may depend on a mobile application which approaches the customers or improve the performance of internal processes. However, developing software for the mobile platform is an expensive process which takes time and resources. A framework to convert Delphi Desktop applications into native Android applications in an automatic way constitutes a useful tool for architects and software developers can contribute with the implementation phase of the application. Therefore, this work is based on methods and processes for software reengineering as the PRE / OO (Process of Reengineering Object Oriented), for automatic conversion of an application developed in Delphi environment in an application for Android mobile platform. At last, an experiment was performed with a real case to corroborate the goals.

Keywords: Delphi; Android; Mobile Technology; Reengineering, Reverse Engineering; Change Platform; Translation Program.

Recebido em: 10/05/2014
Revisado em: 16/08/2014
Aprovado em: 15/09/2014

1 INTRODUÇÃO

A utilização de Smartphone e Tablets é uma atividade comum nas empresas e muitas tentam incorporar aplicações móveis em seu dia-a-dia integrando-as com seus sistemas de *back-end*[1]. Por possuírem Sistema Operacional (plataforma que permite desenvolver, instalar e executar aplicativos), os dispositivos móveis se parecem com um computador de mão (LECHETA, 2010).

Assim como já mencionava Kalakota e Robinson (2002, p. 22), a próxima geração de líderes empresariais de sucesso seriam aqueles que iriam abraçar o conceito móvel totalmente, em vez de um produto ou um serviço específico, como ponto de partida de inovação móvel. A história das empresas tem mostrado que, em tempos de criação e mudanças tecnológicas, a liderança do mercado acontece para os que entendem e implementam estratégias para criar novo valor.

Em 2013, o uso de aplicativos para dispositivos móveis cresceu 115% e as projeções para os próximos anos demonstram que esse crescimento continuará crescendo. Enquanto um aplicativo é usado por seus consumidores, a empresa tem a oportunidade de analisar esse público, gerando percepções, extraindo novas ideias e aprimorando o serviço de acordo com a usabilidade. As percepções feitas enquanto os consumidores utilizam o aplicativo guiam o negócio para o crescimento (INSPIREIDEIAS, 2014).

Dentre os aplicativos voltados a smartphones, enumeram-se principalmente dois tipos de plataformas, os aplicativos chamados nativos e os aplicativos Web. Aplicativos nativos são aqueles instalados no sistema operacional presente nos aparelhos móveis, enquanto que aplicativos Web necessitam de navegadores de Internet, tais como os presentes nos computadores pessoais, para serem utilizados (TOLEDO; DEUS, 2013, p.13).

A adequação a novos paradigmas sempre é seguida de muitos desafios. Uma mudança para plataforma móvel implica em novos softwares escritos em uma linguagem específica, como Java na plataforma *Android* do Google[2]. Isto pode tornar o investimento mais alto no início do desenvolvimento, por exigir treinamento e profundo conhecimento na nova plataforma (TOLEDO; DEUS, 2013, p.15).

O desenvolvimento de software para plataforma móvel exige uma razoável

[1]Sistemas *back-end*: Programa de computador que fica em segundo plano, ou reside em um servidor. Um usuário, em geral, usa somente a interface de uma aplicação *front-end* e raramente (ou nunca) precisa acessar aplicativos *back-end* (que normalmente permanecem "transparente" para ele) (BUSINESSDICTIONARY, 2014).

[2]Google: É uma empresa multinacional americana de serviços online e software. O Google hospeda e desenvolve uma série de serviços e produtos baseados na internet e muito do seu lucro é gerado pela publicidade do AdWords. A empresa foi fundada por Larry Page e Sergey Brin (SIGNIFICADOS,2014).

customização para que possam ser continuamente executados em diversas plataformas e versões (MARTINS; ANTONIO; OLIVEIRA, 2013).

Visto a importância das empresas estarem na plataforma móvel e as dificuldades de migração encontradas por elas, este trabalho propõe a reutilização de software, visando reaproveitar ao máximo o trabalho de análise, projeto e implementação já realizados.

Grahl (1997) define reutilização de software como sendo uma técnica que pode ser utilizada para reduzir os esforços despendidos no desenvolvimento e manutenção de programas, ou seja, reutilização é a reaplicação de informações e artefatos de um sistema já definido, em outros sistemas semelhantes.

Partindo das afirmações e necessidades mencionadas anteriormente busca-se a conversão de aplicativos desenvolvidos no ambiente de programação Delphi para uma plataforma móvel. Todo código fonte procedural de um aplicativo Delphi a ser traduzido passa a ser orientado a objetos. Dessa forma, favorece a desenvolvedores e engenheiros de software na produção de aplicativos em um curto espaço tempo.

A principal técnica utilizada para conversão é a reengenharia de software, uma forma de reuso que permite obter o entendimento do domínio da aplicação, recuperando as informações das etapas de análise e projeto e organizando-as de forma coerente e reutilizável (NOVAIS; PRADO, 2013).

A reengenharia é a forma que muitas organizações estão buscando para manter/refazer seus softwares. Livrando-se das manutenções difíceis e da degeneração de suas estruturas elas diminuem os custos de desenvolvimento e garantem um tempo de vida maior para os aplicativos (LEMOS et al., 2003, p.1).

Este trabalho apresenta o desenvolvimento de um *framework*[3] para a conversão de aplicativos Delphi, que faça uma leitura e interpretação dos mesmos, gerando código Java para o ambiente *Android* pronto para o uso. O ambiente de desenvolvimento Delphi é muito utilizado no mercado de software brasileiro, desta forma, há uma infinidade de aplicativos que poderiam ser convertidos para plataforma móvel (LARANJEIRA, 2011).

Trata-se de um *framework* de auxílio ao processo de desenvolvimento de sistemas que atua na fase de implementação do projeto, gerando o código-fonte que seria criado pelo programador. As regras de negócio e disposição das interfaces gráficas

[3] *Framework*: É um conjunto de classes que incorpora um projeto abstrato de soluções para uma família de problemas relacionados. O conjunto de classes deve ser flexível e extensível para permitir a construção de vários aplicativos com pouco esforço, especificando apenas as particularidades de cada aplicativo (JOHNSON; FOOTE, 1988).

são preservadas do sistema original, o *framework* minimiza o esforço da atividade de reengenharia de sistemas procedimentais para sistemas orientados a objetos.

Sebesta (2002, p.320) define: "Procedimentos são coleções de instruções que definem computações parametrizadas. Estas, por sua vez, são ativadas por instruções de chamada simples".

Orientação a Objetos é um paradigma para o desenvolvimento de software que se baseia na utilização de componentes individuais (objetos) que colaboram para construir sistemas mais complexos. A colaboração entre os objetos é feita por meio do envio de mensagens (SOUZA, 2014).

As demais seções deste trabalho estão organizadas da seguinte maneira: na Seção 2 são apresentados os trabalhos relacionados; na Seção 3 a descrição do *framework* e seu funcionamento; na Seção 4 são descritos os métodos de análise dos códigos; na Seção 5 é apresentada a solução implementada neste trabalho; na Seção 6 é apresentado o experimento utilizado para avaliar a eficiência do framework; na Seção 7 são demonstrados os resultados obtidos com os experimentos, e por fim, na Seção 8 são apresentadas as conclusões e considerações finais do trabalho.

2 TRABALHOS RELACIONADOS

Fonseca (2005) desenvolveu uma ferramenta para conversão de interfaces gráficas feitas em ambiente Delphi para a linguagem de programação Java. A ferramenta Delphi2Java-II implementa: a) conversão de arquivos DFM (*Definition Form* – Definição do Formulário) para a linguagem Java; b) conversão de 25 componentes de interface; c) disponibilização no projeto Java dos seguintes eventos: "onclick", "onchange", "onkeypress", "onenter" e "onexit", além dos eventos "onCreate" e "onDestroy" pertinentes ao "TForm", "onPopup" pertinente ao componente "TPopupMenu" e "onCloseup" pertinente ao "TComboBox".

Mattos e Martins (2006) complementou o projeto Delphi2Java-II iniciado por Fonseca (2005). A ferramenta foi desenvolvida como um gerador de interface e de código para acesso a banco de dados. Esta aplicação lê o conteúdo dos arquivos Delphi (DFM) e produz classes Java que apresentam o mesmo comportamento dos formulários desenvolvidos em Delphi. 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; c) a conversão de sete componentes de visualização de dados; d) a conversão de quatro componentes de acesso a banco de dados; e)

a interligação dos componentes de acesso a banco de dados com os componentes de visualização de dados, implementada em Delphi utilizando o 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") de uma aplicação Delphi, permitindo que o código seja futuramente inserido (MATTOS; MARTINS, 2006).

Rezini (2013) desenvolveu a ferramenta DelphiToAndroid utilizando o ambiente Delphi na forma de um componente e permite realizar a conversão de interfaces gráficas do ambiente Delphi para a plataforma *Android*. 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* (REZINI, 2013).

Zimmermann (2011) desenvolveu uma ferramenta para conversão de interfaces gráficas desenvolvidas em Delphi para a biblioteca Gimp Tool Kit (GTK+). A ferramenta DelphiToGTK+, assim denominada pelo autor, 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).

3 DESCRIÇÃO DO FRAMEWORK

O *framework* DelphiToAndroidConverter, desenvolvido neste trabalho, realiza a conversão de aplicativos Delphi em aplicativos Android.

O *framework* é constituído de um conjunto de classes escritas na linguagem de programação *Object Pascal* utilizando padrões de orientação a objeto, que são responsáveis pela tradução dos códigos de um aplicativo escrito em *Object Pascal* para a linguagem Java nativa do *Android*. Além do código estruturado em procedimentos presente nas unidades de código *Units* e os objetos de tela presentes nos formulários *Forms*, o banco de dados Microsoft SQL Server 2008 do aplicativo Delphi é mapeado no banco de dados SQLite para que possa ser utilizado no *Android*.

O principal método de conversão utilizado como base para realização do trabalho foi o Fusion/RE. O método *Fusion* foi criado por Coleman (1994), desenvolvido originalmente para o projeto de sistemas baseado na abordagem orientada a objetos, e posteriormente adaptado ao uso da engenharia reversa. Três fases distintas compreendem o método: análise, projeto e implementação (PENTEADO, 1996). Baseado

nestas fases, todo código do aplicativo Delphi é avaliado e analisado, para reconhecimento das entidades, e criação do modelo de objetos e interface de acordo com o modelo de dados atual do aplicativo. O esquema mostrado na Figura 1 mostra o esquema básico de funcionamento do *framework* desenvolvido neste trabalho.

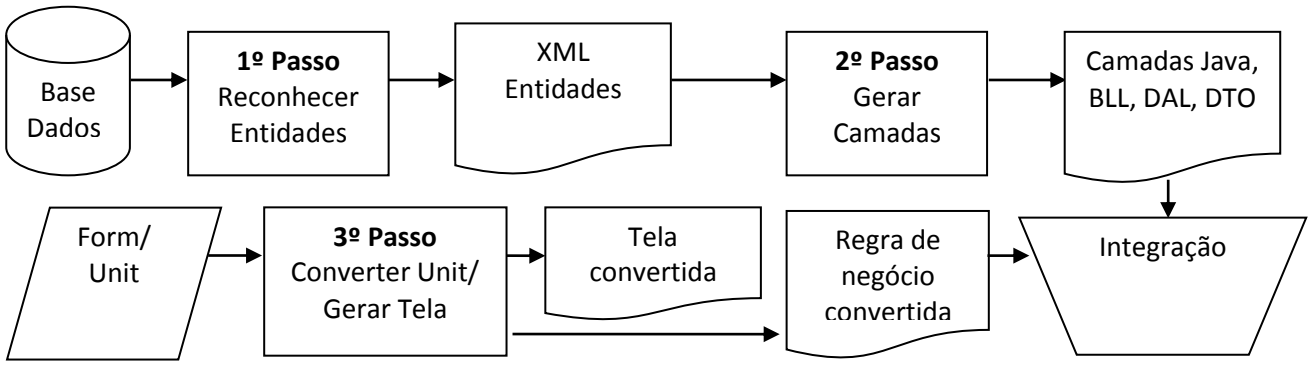


Figura 1. Esquema básico de funcionamento do *framework*.

Segundo Penteado, Germano e Masiero (2009), o método de engenharia reversa orientado a objetos é composto pelos seguintes passos: revitalizar a arquitetura do sistema com base na documentação existente; desenvolver o modelo de objetos; abstrair o modelo de análise do sistema; e elaborar o novo modelo de operações.

O 1º Passo da Figura 1 é reconhecer as entidades do aplicativo com base na documentação e codificação existente, que consiste em obter informações relacionadas à arquitetura do sistema para o seu entendimento, elaborando-se a lista de todos os procedimentos, sua descrição e relacionamentos. Ainda neste passo, o *framework* conecta-se à base de dados por meio de um conector ADO (*ActiveX Data Objects*), que possibilita conexão a vários bancos diferentes por uma *string* de conexão. Após conectar ao banco de dados é iniciado o processo conversão. Neste passo, todos os arquivos do aplicativo a ser convertido (DFM e PAS) são analisados, todas as linhas de código são percorridas a procura das entidades utilizadas pelo mesmo. Uma entidade é uma representação de um conjunto de informações sobre determinado conceito do sistema. Toda entidade possui atributos, que são as informações que referenciam a entidade (DEVMEDIA, 2014).

No fim do processo é gerado um arquivo XML (*eXtensible Markup Language*) com os nomes de todas as entidades, possibilitando obter um modelo de análise considerando somente os aspectos físicos, que são os componentes da estrutura física do banco, como tabelas, campos, tipos de valores, índices (LOPES, 2014). O *Android* tem integração com o banco de dados SQLite, que

apesar de ser simples, apresenta recursos de persistência e integridade das informações (LECHETA, 2010). No final do processo é gerada também uma classe em Java responsável pela criação do banco de dados no SQLite. O mapeamento do banco de dados Microsoft SQL Server 2008 para o SQLite é feito com auxílio da classe "uTemplateClass" desenvolvida para o *framework*. Essa classe é alimentada com as seguintes propriedades da entidade analisada: nome da entidade, campos chaves, lista de colunas e lista de *constraints*. Na classe "uTemplateClass" também está localizado os métodos para manutenção destas propriedades, como por exemplo o "ColumnCount" que resulta no número de colunas. Cada coluna na lista de colunas armazena as informações referentes a ela no banco de dados, as colunas são representas pela classe "uColumn". Além das propriedades de cada coluna, a classe também implementa alguns métodos como o "GetDataTypeJava" que retorna o tipo de dados específico da coluna para o SQLite. As classes estão representa no diagrama da Figura 2.

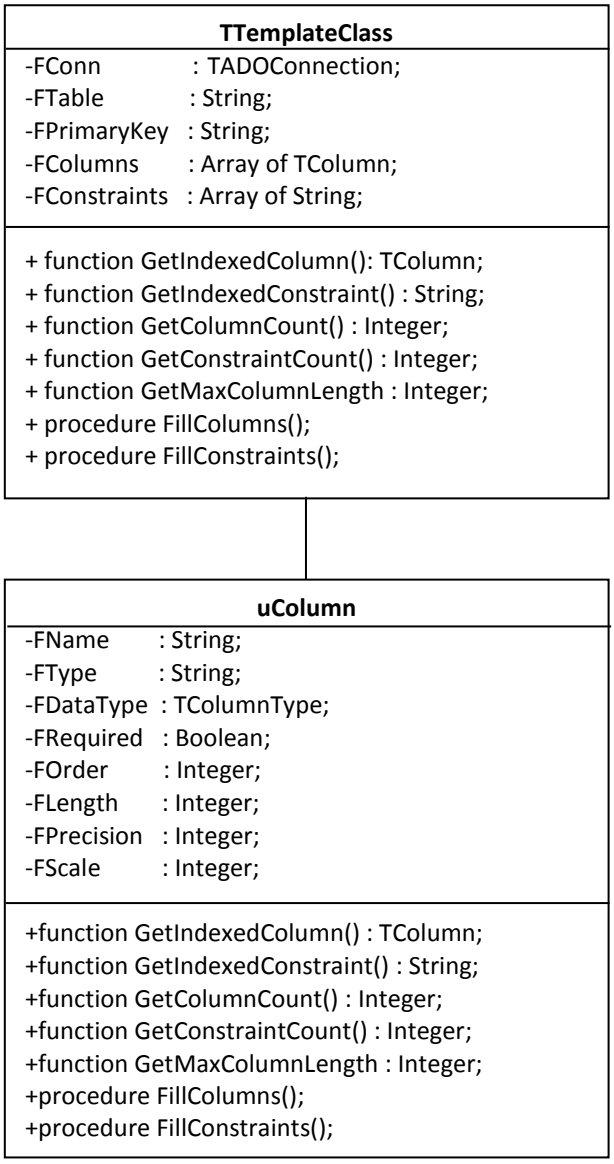


Figura 2. Diagrama de classes do mapeamento do banco.

O 2º Passo (Figura 1) é responsável por desenvolver o Modelo de Objeto com as classes e seus relacionamentos. O Modelo de Objeto é uma plataforma e linguagem de interface neutra que permite que programas e *scripts* possam acessar e atualizar dinamicamente o conteúdo, estrutura e estilo de documentos (W3C, 2014). Neste passo são criadas as três novas camadas do aplicativo. O XML gerado anteriormente (no 1º Passo) é percorrido, e cada entidade é

analisada extraindo as informações dos campos que a compõem. Para cada entidade é criada três novas camadas como mostra a Figura 3. Essas classes identificam as seguintes camadas da aplicação (MICROSOFT, 2014):

a) Camada Lógica de Negócio (*Business Logic Layer* - BLL): consiste na lógica de negócios, contido em uma base de código que é separada da camada de dados e apresentação;
b) Objeto de Transferência de Dados (*Data Transfer Object* - DTO): é um recipiente simples para um conjunto de dados agregados. Ele não deve conter lógica de negócios e limitar o seu comportamento para atividades como verificação de consistência interna e validação básica;
c) Camada de Acesso aos Dados (*Data Access Layer* - DAL): responsável em recuperar e modificar as informações localizadas no banco de dados. Todo o código que é específico para a fonte de dados subjacente - como a criação de uma conexão com o banco de dados, e os comandos para seleção, inserção, atualização e deleção - deve ser localizado na DAL. A camada de apresentação não deve conter nenhuma referência a esse código de acesso aos dados, mas deve sim fazer chamadas para o DAL para qualquer e todas as solicitações de dados. Camadas de acesso a dados normalmente contêm métodos para acessar os dados do banco de dados subjacentes.

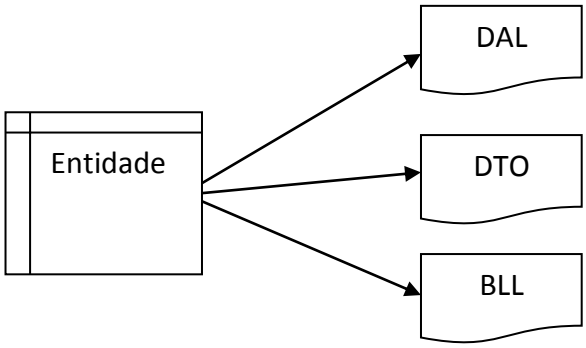


Figura 3. Camadas geradas por entidade no segundo passo.

O 3º Passo é a abstração da regra de negócio do aplicativo a ser convertido e a elaboração do novo Modelo de Operações. Nesta etapa todas as *Units* são analisadas, e o código fonte organizado em procedimento e funções é convertido para uma única classe Java. Esta classe recebe o mesmo nome da *Unit* analisada, e todos os procedimentos e funções encontrados na *Unit* são convertidos em respectivos métodos dessa classe. Além desses, são criados os seguintes métodos para gestão das informações e acesso aos componentes de tela: “iniciarControles” (inicializa os componentes de tela),

"configurarEventos" (atribui os métodos convertidos aos eventos dos compoentes), "DesabilitaTela" (desabilita todos componentes de tela), "ObjectToControl" (atribui os valores das propriedades de um objeto DTO para os componentes de tela), "ControlToObject" (atribui os valores dos componentes e carrega um objeto DTO).

Ainda nesse passo ocorre também a conversão da parte gráfica que são as telas do projeto. Todos componentes presentes nos formulários do ambiente Delphi são convertidos para um componente similar no ambiente *Android*, além do próprio formulario do Delphi que é convertido em uma tela no *Android*. Uma tela é composta de vários elementos visuais, que no caso do *Android* são representados pela classe "android.view.View". A classe "Activity" define que existe uma tela, controla o estado, passagem de parâmetros e métodos. A classe "View" tem a finalidade de desenhar os componetes na tela (LECHETA, 2010). A Figura 4 mostra um exemplo de uma tela "View" com dois componentes, um botão e um componente para exibir textos localizado acima do botão.

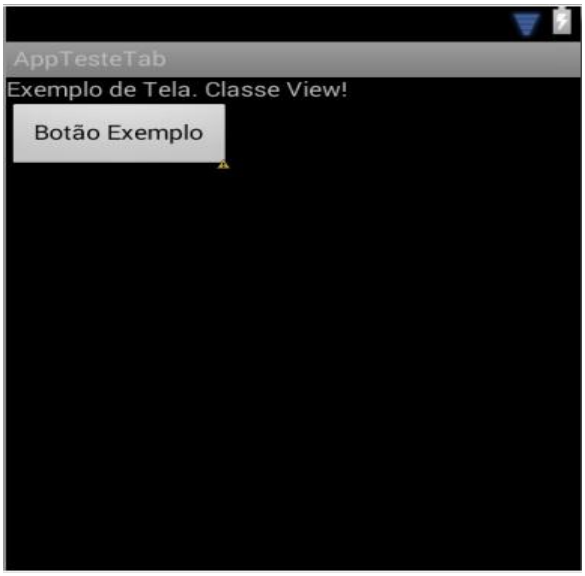


Figura 4. Exemplo de uma tela View.

Segundo Rezini (2013), para conversão das interfaces gráficas, devem-se identificar os componentes de interface do Delphi que possuam comportamento equivalente em interfaces *Android*, convertendo o formulário objeto a objeto e mapeando cada propriedade em uma classe específica. Desta forma, um componente deve ter como primeira classe ancestral, a classe "View".

Uma interface de usuário (*User Interface* - UI) em *Android* é construída por 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, denominados de *widgets,* como mostra o diagrama de classes na Figura 5.

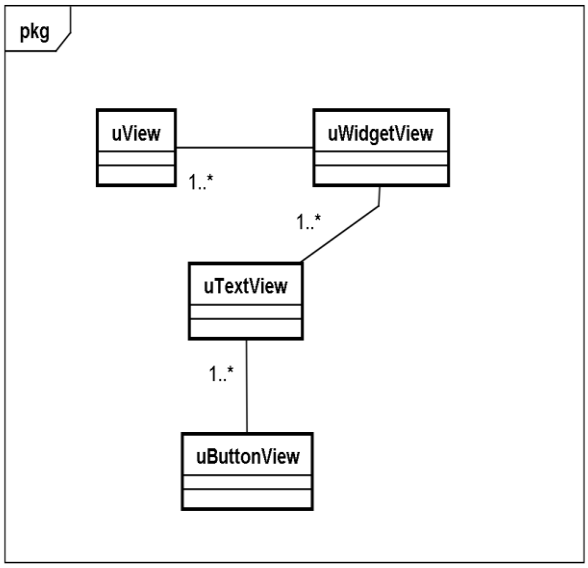


Figura 5. Diagrama de classe de um botão no ambiente *Android*.

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 6, em que cada objeto da classe "ViewGroup" contém outros objetos dependentes da classe "View" e "ViewGroup" (ANDROID DEVELOPERS, 2013).

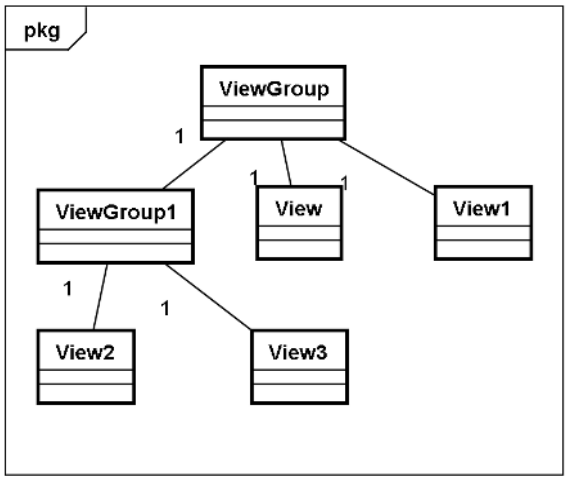


Figura 6. Exemplo de um Layout "ViewGroup".

No final deste passo (3º Passo) ocorre a integração da camada BLL criada no 2º Passo com a classe que corresponde à conversão da *Unit*. Esta integração consiste na implementação da chamada dos métodos criados na camada BLL pela classe com a regra de negócio convertida.

4 MÉTODO DE ANÁLISE DOS CÓDIGOS

Assim como mencionado na Seção 3 (Figura 1), o primeiro processo desempenhado pelo *framework DelphiToAndroidConverter* é a leitura de todas as linhas de código presentes nas *Units* e Formulários a procura de trechos que contenham possíveis entidades do banco dados.

Tabela 1. Códigos contendo entidades, reconhecidos pela ferramenta.

Códigos dos formulários	1 object Table1: TTable 2 TableName ='Empresa' 3 Left = 200 4 Top = 64 5 End 1 SQL.Strings = (2 'SELECT IdVendedor, ' 3 ' NomeVendedor, ' 4 ' DataCadastro,' 5 ' senha' 6 ' FROM Vendedor' 7 ' WHERE código=:Codigo' 8 ' AND senha = :Senha')
Códigos das Units	AdoQuery.close; AdoQuery.sql.clear; AdoQuery.sql.add('SELECT Empresa.IdEspecieDocumentoEspecieVinculaAntigo, EspecieDocumento.Descricao AS Especie FROM Empresa ' + ' LEFT JOIN EspecieDocumento ON EspecieDocumento.Idespeciedocumento = Empresa.IdEspecieDocumentoEspecieVinculaAntigo ' +' WHERE Empresa.IdEmpresa = ' + IntToStr(Empresa)); AdoQuery.open;

A Tabela 1 mostra códigos que são reconhecidos e analisados pela ferramenta.

O reconhecimento dos trechos do código é feito pela presença de palavras reservadas (SELECT, FROM, JOIN, WHERE) que são, por padrão, da linhagem SQL (*Structured Query Language*) (POSTGREESQL, 2014) e por estruturas exclusivas presentes na linguagem Object Pascal, como o "TableName" mostrado na Tabela 1.

Após o reconhecimento, os nomes das entidades são inseridos em um arquivo XML para serem utilizadas posteriormente pelo 2º Passo do *framework*. A Figura 8 mostra a estrutura do arquivo XML, em que cada entidade é colocada na *tag* "<ROW Entidade="NomeDaEntidade"/>. As primeiras linhas do arquivo servem para informar como as informações estão estruturadas.

```
<?xml version="1.0" standalone="true"?>
-<DATAPACKET Version="2.0">-
<METADATA>-<FIELDS><FIELD
WIDTH="200" fieldtype="string"
attrname="Entidade"/></FIELDS><PARAM
S/></METADATA>-<ROWDATA>
<ROW Entidade="Vendedor"/>
<ROW Entidade="Logradouro"/>
<ROW Entidade="Cliente"/>
<ROW Entidade="Empresa"/>
<ROW Entidade="PedidoVenda"/>
<ROW Entidade="Cor"/>
<ROW Entidade="Modelo"/>
<ROW Entidade="CorTamanhoFinal"/>
<ROW Entidade="Faixa"/>
<ROW Entidade="Marca"/>
<ROW Entidade="Produto"/>
<ROW Entidade="PedidoVendaItens"/>
</ROWDATA></DATAPACKET>
```

Figura 8. Arquivo XML com as entidades encontradas.

Após o reconhecimento das entidades, é realizada a leitura do XML para geração das classes em Java. Esta leitura é feita com a ajuda de um componente "TClientDataSet" do ambiente Delphi, que possibilita leitura e manutenção de arquivos XML.

A sintaxe do aplicativo a ser convertido, que é a forma como as instruções de uma linguagem são escritas, é analisada pela classe "UconvertUnit" do *framework*. Esta classe possui as principais funções do *framework*, em que todo código Delphi é convertido por um equivalente em Java. Isso é realizado por meio do reconhecimento das estruturas de código que ambas linguagem possuem. Assim para cada objeto de linguagem *Object Pascal* é mapeado um novo código correspondente em Java. A seguir, a Figura 9 mostra um exemplo da conversão de uma estrutura condicional.

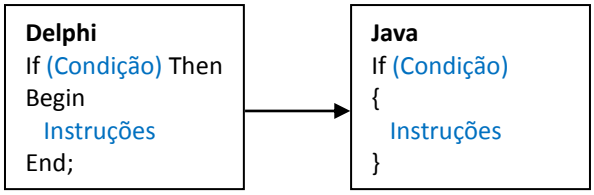


Figura 9. Sintaxe da *Unit* convertido pelo *framework*.

O Delphi utiliza a *Visual Component Library* (VCL), uma biblioteca de componentes visuais mapeada sobre a API[4] do Windows e implementada em *Object*

[4]API: é o acrônimo de *Application Programming Interface*. É o conjunto de padrões de programação que permite a construção de aplicativos e a sua utilização (TECMUNDO, 2014).

Pascal. Ela inclui controles nativos do Windows, como botões e caixas de edição, controles, além de vários controles do Delphi ligados ao conceito de janela do Windows. Os componentes são subclasses da classe "TComponent", classe base ou raiz da hierarquia de classes dos componentes Delphi. A partir dessa classe os componentes visuais herdam também da classe "TControl". Dessa última, obtêm-se atributos que determinam posição e tamanho do componente em tela, além de serem exibidos em um formulário em tempo de projeto e execução. Além da classe "TControl", um componente visual ainda pode herdar outras classes, obtendo assim outras propriedades e eventos (CANTU, 2003, p. 89).

A conversão da tela também é feita de forma sequencial por meio do arquivo DFM que representa um formulário no ambiente Delphi. Cada componente no DFM é identificado pelo termo "object" e suas propriedades terminam com o termo "end". Todas as propriedades são encontradas neste intervalo e seu valor atribuído através do sinal de igual ("=") como mostra a Figura 10.

```
object Table1: TTable
  TableName = 'fornecedor'
  Left = 144
  Top = 105
end
```

Figura 10. Exemplo de um componente *TTable* em um formulário.

Para facilitar a conversão, depois de identificados, os componentes são novamente mapeados em objetos Delphi pelo *framework.* 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 nome do componente seguido de um ponto e a propriedade, ou com o uso do método "GetPropValue", utilizando como parâmetro um objeto e o nome literal da propriedade, retornando um valor do tipo "Variant". Na Figura 11 é mostrado o fluxo da conversão do formulário.

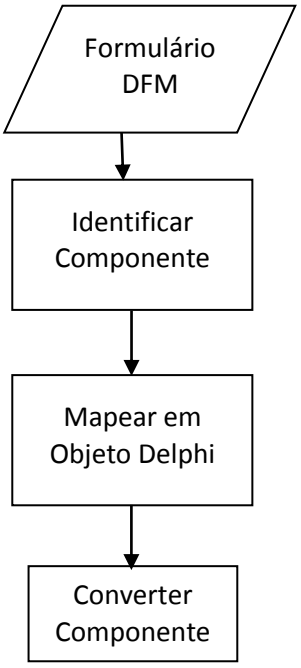


Figura 11. Sequência de conversão do Formulário (Tela).

5 SOLUÇÃO IMPLEMENTADA

Neste trabalho foi desenvolvido o *framework* DelphiToAndroidConverter contendo 42 Classes mostradas na Figura 12, que tem como função converter aplicativos desenvolvidos no ambiente Delphi em aplicativos *Android*.

Para serem convertidos, os aplicativos Delphi devem apresentar alguns requisitos:

1. Desenvolvimento utilizando a arquitetura Cliente Servidor;
2. Regra de negócio concentrada em uma mesma *Unit* desenvolvidas em procedimentos e funções para cada *Form*;
3. Componentes de interfaces gráficas contidos dentro da especificação de componentes da Tabela 3.

A Figura 12 apresenta um diagrama contendo as classes do *framework* com suas relações. As classes do pacote "Principal" possuem métodos para iniciar o processo de conversão. Na classe principal "UConvertForm" está contido os principais métodos do *framework*, são eles: "AnalisaEntidadesBancosDados", "GerarBanco", "GerarCamadas", "GerarUnit", "GerarTela", que são executados nesta ordem para conversão do aplicativo. O pacote "ConverteBancoUnit" contém as classes que implementam os métodos responsáveis pela tradução dos códigos nas *Units* e pelo mapeamento do banco de dados. O pacote "ConvertForm" é responsável pela conversão das interfaces gráficas *Forms*, este pacote contém as classes dos componentes que o *framework* está habilitado a converter podendo ser acrescido de outras classes de acordo com a necessidade.

O código Delphi deve seguir uma padronização básica, como por exemplo, a não ocorrência de *Units* sem *Forms,* caso aconteça a utilização de estruturas de códigos que não estejam mapeados nas classes do *framework* serão mantidos exatamente iguais ao Delphi e deverão ser corrigidos manualmente após a conversão.

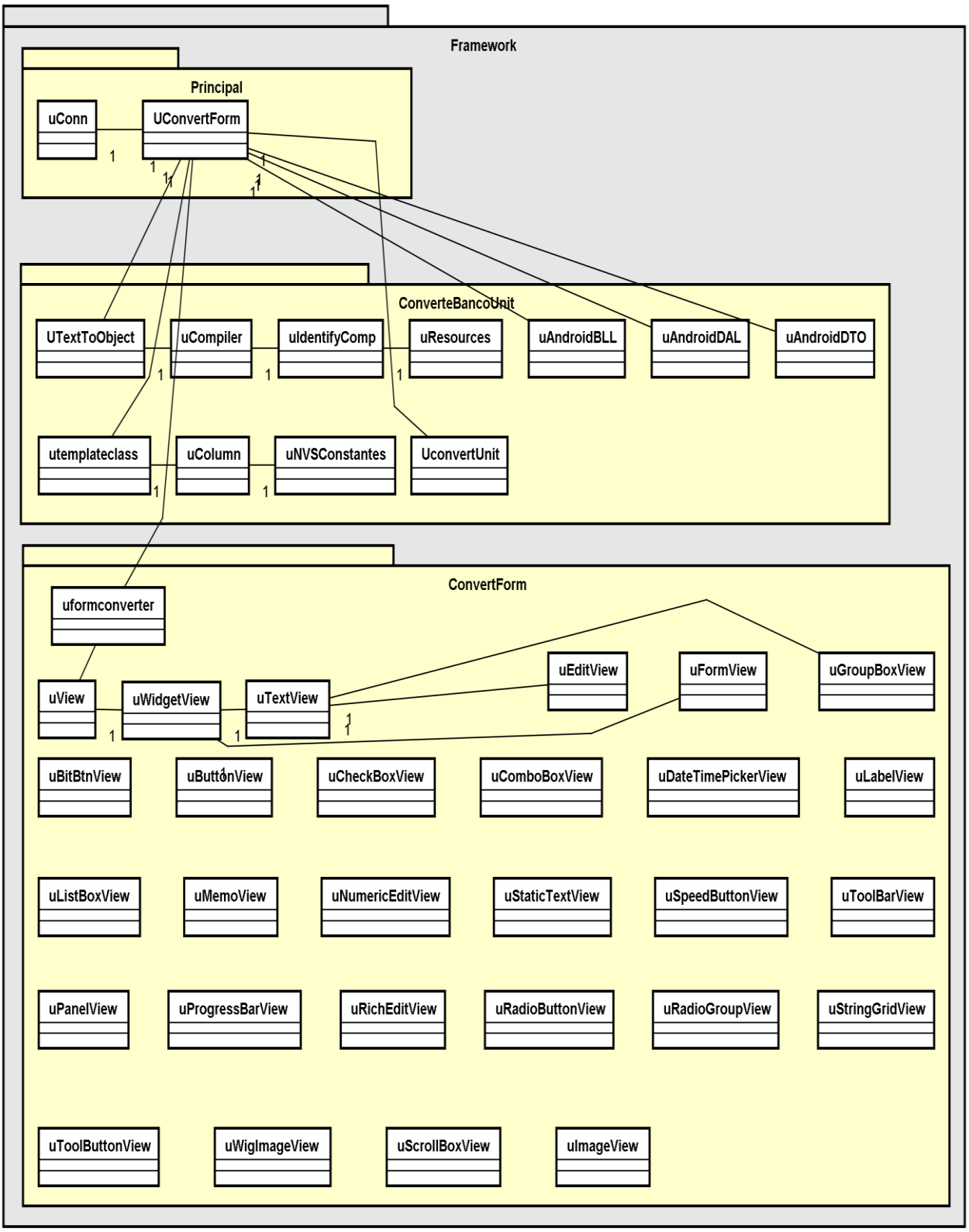


Figura 12. Diagrama de classes do *framework* DelphiToAndroidConverter.

Tabela 3. Componentes de interfaces gráficas convertidos.

Delphi	Android
TForm	RelativeLayout
TPanel	RelativeLayout, EditText
TBitBtn , TRzBitBtn	Button
TButton	Button
TSpeedButton	ImageButton
TRadioButton	RadioButton

TRadioGroup	RadioGroup, RadioButton, EditText
TCheckBox	CheckBox
TComboBox	Spinner
TGroupBox	RelativeLayout, EditText
TListBox	ListView
TLabel, TRzLabel	EditText
TStaticText	EditText
TEdit , TRzDBEdit, TRzEdit	EditText
TMemo	EditText
TRichEdit	EditText
TDateTimePicker	EditText
TStringGrid	GridView
TProgressBar	ProgressBar
TToolBar	RelativeLayout
TToolButton	Button
TScrollBox	ScrollView, RelativeLayout
TImage	ImageView

Após ser compilado, o *framework* opera na forma de um programa executável para Windows e tem apenas duas interfaces para o usuário final. A primeira para que sejam fornecidas as informações de acesso ao banco de dados do aplicativo Delphi, e a segunda com um botão para iniciar o processo de conversão e dois campos para fornecer os diretórios: o primeiro para entrada do aplicativo a ser convertido; e outro onde os arquivos convertidos pelo *framework* serão gerados.

6 EXPERIMENTOS

Para testar o funcionamento do *framework DelphiToAndroidConverter* foi realizado um experimento com um caso real. Nele foi utilizado um aplicativo já existente desenvolvido no ambiente de programação Delphi, versão 7, que utiliza banco de dados Microsoft SQL Server 2008. O aplicativo consiste em um sistema para coleta de pedidos comerciais de produtos para uma empresa de pequeno porte. Todo sistema está escrito de forma estruturada em *Object Pascal*. Na Figura 13 está a representação do fluxo de inserção de informações neste aplicativo.

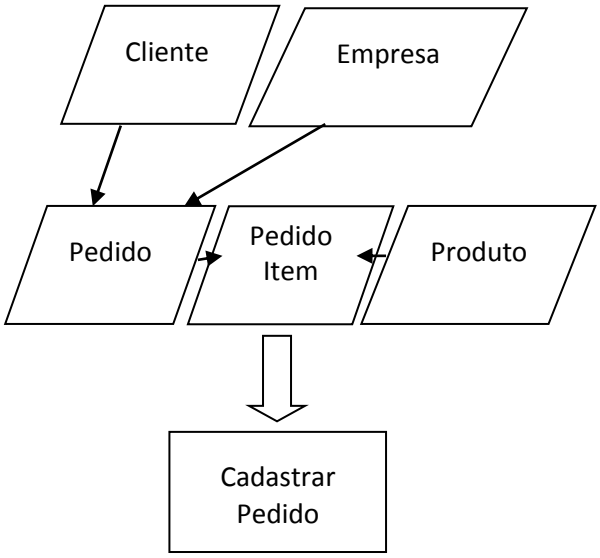


Figura 13. Fluxo da informação no aplicativo.

O aplicativo possui oito interfaces gráficas no total, sendo duas de inserção e manutenção dos dados, cinco de pesquisa, e uma de acesso ao sistema.

As interfaces gráficas de pesquisa apresentam apenas a funcionalidade de acesso `as informações contidas no banco de dados, apresentando os dados para o usuário em forma de grade. Na primeira tela de cadastro é possível inserir, editar e excluir os dados do pedido, como a data, o cliente e a empresa. A segunda tela de cadastro tem a

funcionalidade de manutenção dos itens do pedido, permitindo inserir, editar e excluir as informações relacionadas aos itens que são: produto, quantidade e preços.

Após análise do aplicativo, pelo *framework* desenvolvido neste trabalho, foram encontradas as entidades descritas na Tabela 3. Estas tabelas e seus relacionamentos foram preservados de forma idêntica ao banco de dados original como mostrados na Figura 14, que representa o Modelo Entidade Relacionamento (MER), que segundo Mello (2014) é um modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos.

Tabela 3. Entidades do aplicativo Delphi.

Vendedor
Logradouro
Cliente
Empresa
PedidoVenda
Cor
Modelo
CorTamanhoFinal
Faixa
Marca
Produto
PedidoVendaItens

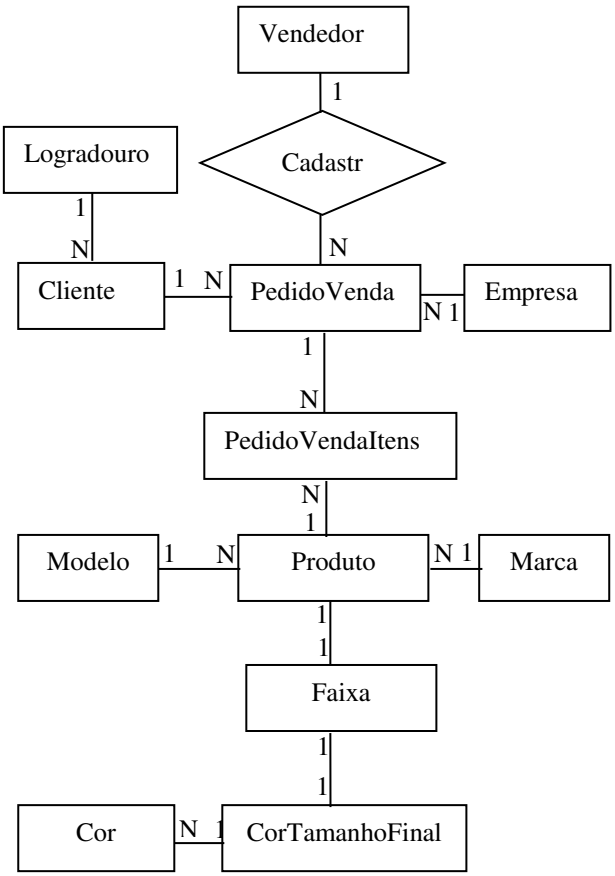
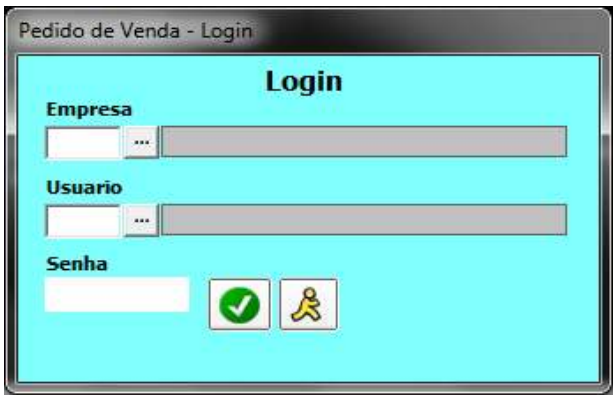


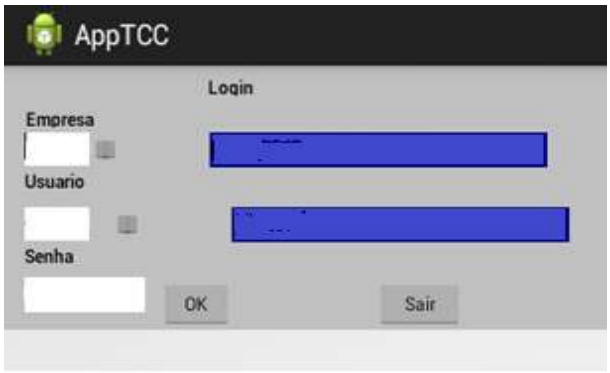
Figura 14. MER do aplicativo Delphi.

7 RESULTADOS

Toda regra de negócio foi convertida necessitando poucas alterações para instalação e utilização em um aparelho móvel. Todas as interfaces gráficas da aplicação foram convertidas e se mantiveram inalteradas com relação aos componentes e suas disposições. A Figura 15 mostra a tela de Acesso ao sistema como é no aplicativo Delphi e no aplicativo *Android*.



(a)



(b)

Figura 15. a) Tela de acesso do aplicativo Delphi; b) Tela do aplicativo convertida para *Android*.

O banco de dados foi mapeado corretamente e todas as entidades e relações mostradas na Figura 14 foram mantidas. As classes contendo as regras foram as que apresentaram maiores problemas e precisaram de algumas adaptações como: Criação métodos auxiliares para execução de consultas SQL ao banco de dados, e auxílio na referência dos métodos que representam os eventos das interfaces gráficas. Houve também intervenção de codificação manual na integração da regra de negócio com as classes geradas no 2º Passo (Figura 1). Isso se deu principalmente pelo fato de alguns métodos terem sidos criados automaticamente na conversão, como mostrado no 3º Passo da Figura 1.

A Figura 16 (ilustrada na próxima página), apresenta como ficou o digrama de classes após a conversão do aplicativo. As classes dos pacotes “BLL”, ”DAL”, ”DTO”, “UnitJava” e as classes “DataBaseHelper”, “msUtil” foram convertidas pelo *framework*. A classe “BuildConfig” e “R” são geradas por padrão em um projeto Android.

8 CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi a conversão de aplicativos Delphi em aplicativos *Android* nativos, analisando os resultados obtidos com o experimento. Este trabalho mostrou como é possível reutilização de boa parte do código e praticamente todas as interfaces gráficas do aplicativo Delphi, preservando principalmente toda regra de negócio.

Com a utilização do *framework* todo tempo e recurso que seria despendido para análise e criação de um novo aplicativo foi economizado, permitindo aos desenvolvedores se preocuparem exclusivamente com possíveis melhorias no aplicativo.

O uso de técnicas de engenharia reversa foi de grande valia e possibilitou que os *framework* chegasse aos resultados esperados.

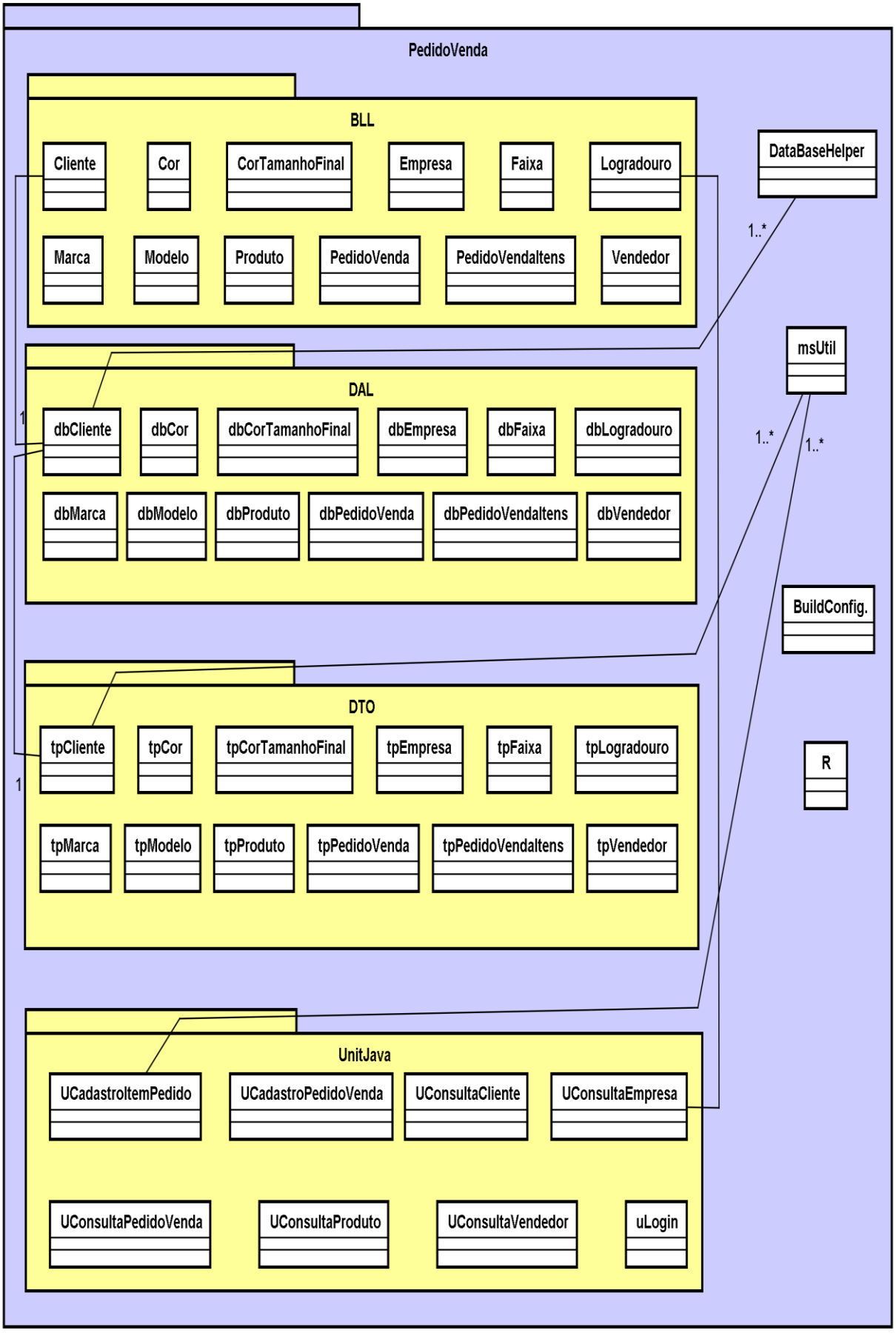


Figura 16. Diagrama de classes do aplicativo usado no estudo caso convertido para *Android*.

REFERÊNCIAS

ANDROID DEVELOPERS. **The developer's guide**. 2013. Disponível em: <http://developer.android.com/guide/>. Acesso em: 30 mar. 2014.

CANTU, M. **Dominando o Delphi 7**: a bíblia. São Paulo: Pearson Education do Brasil, 2003.

COLEMAN, D. **Object-Oriented development:** the fusion method. 1. ed. Englewood Cliffs: Prentice Hall, 1994.

DEVMEDIA. **Modelagem de Dados 1:** entidades. 2014. Disponível em: <http://www.devmedia.com.br/modelagem-de-dados-1-entidades/4140>. Acesso em: 14 jan. 2014.

FONSECA, F. **Ferramenta conversora de interfaces gráficas:** Delphi2Java-II. 2005. 59 f. Trabalho (Conclusão de Curso) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2005-I/2005-1fabriciofonsecavf.pdf>. Acesso em: 08 jun. 2014.

GRAHL, E. A. **Reutilização de software**. 1997. Disponível em: <http://www.inf.furb.br/~egrahl/disciplinas/engeharia/material/reuso.doc>. Acesso em: 04 jun. 2014.

INSPIREIDEIAS. **Por que sua empresa deve investir em aplicativos mobile?** 2014. Disponível em: <http://inspireideias.ag/por-que-sua-empresa-deve-investir-em-aplicativos-mobile/>. Acesso em: 01 jun. 2014.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. **Journal of Object-Oriented Programming**, v. 1, n. 2, june 1988.

KALAKOTA, R.; ROBINSON, M. **M-business:** tecnologia movel e estrategia. 1. ed. Porto Alegre: Bookman, 2002.

LARANJEIRA, L. **O Delphi morreu?** 2011. Disponível em: <http://www.lucianolaranjeira.com.br/o-delphi-morreu/>. Acesso em: 15 abr. 2014.

LECHETA, R. R. **Google Android**. 2. ed. São Paulo: Novatec, 2010.

LEMOS, G. S. et al. **Padrões de reengenharia auxiliados por diretrizes de qualidade de software**. 2003. Disponível em: <http://www.cin.ufpe.br/~sugarloafplop/final_articles/09_PREOO.pdf >. Acesso em: 05 fev. 2014.

LOPES, A. **Modelo físico (Sgbd).** 2014. Disponível em: <http://docente.ifrn.edu.br/abrahaolopes/semestre-2013. 1/3.2401.1m-prog-bd/11-16-modelo-conceitual-fisico-logico-e-entidade-relacionamento>. Acesso em: 08 mar. 2014.

MARTINS, C. S.; ANTONIO A. L. T.; OLIVEIRA C. A. **Os desafios para a mobilização de aplicações baseadas em plataforma Web**. 2013. Disponível em: <https://projetos.extras.ufg.br/enacomp/anais/pdf/39.pdf>. Acesso em: 02 mai. 2013.

MATTOS, M. M.; MARTINS, J. 2006. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbes/2001/015.pdf>. Acesso em: 05 ago. 2013.

MELLO, M. **O Modelo Entidade-Relacionamento (MER).** 2014. Disponível em: <http://www.las.pucpr.br/mcfmello/BD/BD-Aula02-MER.pdf>. Acesso em: 01 jun. 2014.

MICROSOFT**. Tutorial 1:** Creating a Data Access Layer; Tutorial 2: Creating a Business Logic Layer; Data Transfer Object; Data Transfer Object. 2014. Disponível em: http://msdn.microsoft.com/enus/library/aa581778.aspx; http://msdn.microsoft.com/en-us/library/aa581776.aspx;http://msdn.microsoft.com/en-us/library/ff649585.aspx. Acesso em: 01 jun. 2014.

NOVAIS, E. R. A.; PRADO, A. F. **Reengenharia de software orientada a componentes distribuídos**. 2013. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbes/2001/015.pdf>. Acesso em: 05 ago. 2013.

PENTEADO, R. A. D. **Um Método para engenharia reversa orientada a objetos**. 1996. 237 f. Tese (Doutorado) – Instituto de Física de São Carlos da Universidade de São Paulo, São Carlos, SP.

PENTEADO, R. A. D.; GERMANO F. S. R.; MASIERO P. C. **Engenharia Reversa Orientada a Objetos do Ambiente StatSim**. 2009. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=16961>. Acesso em: 06 jan. 2014.

POSTGREESQL. **Apêndice C. Palavras chave do SQL.** 2014. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/sql-keywords-appendix.html>. Acesso em: 02 jun. 2014.

REZINI, D. J. **Geração de interfaces android a partir do delphi**. 2013. Disponível em: <http://www.bc.furb.br/docs/mo/2013/353395_1_1.pdf>. Acesso em: 06 jan. 2014.

SEBESTA, R. W. **Conceitos de linguagens de programação**. Porto Alegre: Bookman, 2002.

SIGNIFICADOS. **O que é Google**. 2014. Disponível em: <http://www.significados.com.br/google/>. Acesso em: 01 jun. 2014.

SOUZA, C. **Orientação a objetos**. 2014. Disponível em: <http://cultura.ufpa.br/cdesouza/teaching/labes/OO-basic-concepts.pdf>. Acesso em: 15 mai. 2014.

TECMUNDO. **O que é API?** 2014. Disponível em: <http://www.tecmundo.com.br/programacao/1807-o-que-e-api-.htm>. Acesso em: 03 jun. 2014.

TOLEDO, J. M.; DEUS, J. D. **Planejamento de iniciativas de adoção de tecnologias móveis**. 2013. Disponível em: <http://revista.feb.unesp.br/index.php/gepros/article/viewArticle/738>. Acesso em: 11 out. 2013.

W3C. **Extensible Markup Language (XML)**. 2014. Disponível em: <http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf>. Acesso em: 15 mai. 2014.

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