

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE FERRAMENTA PARA A GERAÇÃO DE
INTERFACES GRÁFICAS ANDROID A PARTIR DE
TEMPLATES PREDEFINIDOS

BRAYAN ALISON BEDRITCHUK

BLUMENAU
2015

2015/1-06

BRAYAN ALISON BEDRITCHUK

**PROTÓTIPO DE FERRAMENTA PARA A GERAÇÃO DE
INTERFACES GRÁFICAS ANDROID A PARTIR DE
TEMPLATES PREDEFINIDOS**

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Francisco Adell Péricas, Mestre - Orientador

**BLUMENAU
2015**

2015/1-06

**PROTÓTIPO DE FERRAMENTA PARA A GERAÇÃO DE
INTERFACES GRÁFICAS ANDROID A PARTIR DE
TEMPLATES PREDEFINIDOS**

Por

BRAYAN ALISON BEDRITCHUK

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

Presidente: _____
Prof. Francisco Adell Péricas, Mestre – Orientador, FURB

Membro: _____
Prof. Marcel Hugo, Mestre – FURB

Membro: _____
Prof. Roberto Heinzle, Doutor – FURB

Blumenau, 04 de dezembro de 2015

Para meus pais, Nara Bernadete Canani e Eliseu Bedritchuk, que me ensinaram através do exemplo o significado de persistência, gratidão e humildade. Amo vocês dois e lhes devo tudo.

AGRADECIMENTOS

Aos meus pais, por toda a confiança e paciência ao longo de todos esses anos.

Ao meu orientador, professor Francisco Adell Péricas, pelo suporte na elaboração deste projeto, mostrando sempre disposição para discutir ideias e colaborar com o que fosse necessário.

À minha namorada, Else Elly de Souza, pelo incentivo e compreensão, não só durante a elaboração deste projeto, como também durante toda a graduação e que mesmo à distância, nunca se mostrou distante. Eu te amo e não teria conseguido sem você.

À Microton Informática, por ter proporcionado a minha primeira experiência profissional e pela flexibilidade com relação à elaboração deste projeto.

Ao meu amigo Cedulio Cezar e Silva, pelas discussões sobre design, melhorias de código e boas práticas de desenvolvimento de software. Por sempre incentivar o uso do Clean Code e por ter assistido à apresentação deste projeto.

À todas as pessoas que contribuíram de forma direta ou indiretamente na elaboração deste projeto.

“Dias inteiros de calma, noites de ardência, dedos no leme e olhos no horizonte, descobri a alegria de transformar distâncias em tempo. Um tempo em que aprendi a entender as coisas do mar, a conversar com as grandes ondas e não discutir com o mau tempo. A transformar o medo em respeito, o respeito em confiança. Descobri como é bom chegar quando se tem paciência. E pra se chegar, onde quer que seja, aprendi que não é preciso dominar a força, mas a razão. É preciso, antes de mais nada, querer.”

Amyr Klink

RESUMO

Este trabalho apresenta a implementação de um protótipo de ferramenta para geração de interfaces gráficas Android a partir de *templates* predefinidos. Através de uma interface web, utilizando o *framework* JQuery, os componentes Android podem ser arrastados e soltados na tela. É possível alterar os atributos presentes em cada componente disponível para ser adicionado. Ao selecionar o botão Gerar, através do *framework* FreeMaker, o protótipo realizará as devidas validações e conversões sobre os campos preenchidos para os componentes Android. Logo em seguida, é realizada a geração das telas a partir da requisição feita pelo usuário. Após o processamento dos componentes adicionados e outras informações sobre a tela a ser gerada, o protótipo fornece ao usuário o *download* dos arquivos de tela compactados e prontos para serem importados na IDE Android Studio. Por último, os arquivos gerados são executados corretamente em um *smartphone* com Android 5.0 e em um *tablet* com Android 4.1.1.

Palavras-chave: Java EE. Android. Gerador de código-fonte.

ABSTRACT

This paper presents the development of a prototype tool for generation of Android GUIs from predefined templates. Through a web interface, using the JQuery framework, Android components can be dragged and dropped on the screen. It can change the attributes present in each available component to be added. When selecting the Generate button, through the framework FreeMaker, the prototype will perform the necessary validations and conversions regarding the fields filled for Android components. After it is accomplished the generation of screens from the request made by the user. After processing of the added components and other information on the screen to be generated, the prototype provides the user download the compressed files and ready to be imported into the IDE Android Studio. Finally, the generated files run properly on a smartphone with Android 5.0 and a tablet with Android 4.1.1.

Key-words: Java EE. Android. Source code generator.

LISTA DE FIGURAS

Figura 1 - Projetos pré-construídos para iOS, Android e Windows Phone.....	16
Figura 2 - Construção de uma aplicação utilizando a ferramenta Xamarin Studio.....	17
Figura 3 - Tela em construção na ferramenta Smartface App Studio	17
Figura 4 - Aplicações iOS e Android geradas com Titanium	18
Figura 5 - Construção de um projeto utilizando o Titanium Studio.....	19
Figura 6 – Exemplo de um <code>Rectangular Button</code> e <code>Floating Action Button</code> ...22	
Figura 7 - Exemplo de uma tela Android com <code>Cards</code>	23
Figura 8 - Implementação de uma <code>RecyclerView</code>	24
Figura 9 - Exemplo de uma <code>Toolbar</code>	25
Figura 10 - Exemplo de um <code>EditText</code>	26
Figura 11 - Exemplo de um <code>TextView</code>	27
Figura 12 - Exemplo de um <code>Checkbox</code>	28
Figura 13 - Formulário desenvolvido em Delphi e resultado equivalente em Android	29
Figura 14 - Tela principal da ferramenta	30
Figura 15 - Diagrama de casos de uso	32
Figura 16 - Diagrama de classes da camada de negócio	33
Figura 17 - Diagrama de classes de funcionalidades.....	35
Figura 18 - Diagrama de atividades.....	36
Figura 19 - Protótipo sendo executado em um navegador	43
Figura 20 - Componentes adicionados sendo reordenados	44
Figura 21 - Componente adicionado com atributos preenchidos	44
Figura 22 - Formulário preenchido com as informações principais da tela	45
Figura 23 - Download do arquivo compactado contendo os arquivos gerados.....	46
Figura 24 - Arquivos gerados devidamente importados na IDE Android Studio	48
Figura 25 - Aplicativo de testes executado em um <i>tablet</i> com Android 4.1.1	48
Figura 26 - Aplicativo de testes executado em um <i>smartphone</i> com Android 5.0	49

LISTA DE QUADROS

Quadro 1 - Representação de uma atividade em Java	20
Quadro 2 - Arquivo <code>AndroidManifest.xml</code>	21
Quadro 3 - Arquivo <code>button_color.xml</code>	21
Quadro 4 - <i>Template</i> de código de um <code>Rectangular Button</code> e um <code>Floating Action Button</code>	22
Quadro 5 - <i>Template</i> de código de um <code>Card</code>	23
Quadro 6 - <i>Template</i> de código da <code>RecyclerView</code>	24
Quadro 7 - <i>Template</i> de código de uma <code>Toolbar</code>	25
Quadro 8 - <i>Template</i> de código de um <code>EditText</code>	26
Quadro 9 - <i>Template</i> de código do componente <code>TextView</code>	27
Quadro 10 - <i>Template</i> de código de um <code>Checkbox</code>	28
Quadro 11 - Implementação do método <code>service</code>	37
Quadro 12 - Implementação do método <code>converter</code>	38
Quadro 13 - Implementação do método <code>efetuarValidacoes</code>	39
Quadro 14 - Implementação do método <code>gerar</code>	39
Quadro 15 - Implementação do método <code>compactarArquivos</code>	40
Quadro 16 - Implementação do método <code>efetuarDownload</code>	40
Quadro 17 - Implementação do método <code>gerarPagina</code>	40
Quadro 18 - <i>Template</i> de tela Java	41
Quadro 19 - <i>Template</i> de tela XML para componentes com filhos.....	41
Quadro 20 - <i>Template</i> de tela XML para componentes sem filhos	41
Quadro 21 - Implementação da função JavaScript responsável pelas validações de campos ..	42
Quadro 22 – Arquivo XML de exemplo gerado a partir do protótipo	46
Quadro 23 – Arquivo Java de exemplo gerado a partir do protótipo	47
Quadro 24 - Comparativo entre ferramentas	50

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

IDE – *Integrated Development Environment*

JSON – *JavaScript Object Notation*

RF – *Requisito Funcional*

RNF – *Requisito Não Funcional*

SDK – *Software Development Kit*

UML – *Unified Modeling Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS DO TRABALHO	14
1.2 ESTRUTURA DO TRABALHO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 INTRODUÇÃO AOS GERADORES DE CÓDIGO.....	15
2.1.1 EXEMPLOS DE GERADORES DE CÓDIGO COMERCIAIS.....	16
2.2 ESTRUTURA DE PROJETOS ANDROID	19
2.2.1 Atividade (<i>activity</i>).....	19
2.2.2 Android Manifest	20
2.2.3 Recursos (<i>resources</i>).....	21
2.3 PRINCIPAIS COMPONENTES DE TELA DO ANDROID.....	22
2.3.1 Button	22
2.3.2 Card.....	23
2.3.3 RecyclerView.....	24
2.3.4 Toolbar.....	25
2.3.5 EditText.....	25
2.3.6 TextView.....	26
2.3.7 Checkbox.....	27
2.4 TRABALHOS CORRELATOS	28
2.4.1 Gerador de interfaces Android a partir do Delphi.....	28
2.4.2 Elicitar: protótipo de gerador de código a partir de especificações com padrões de requisitos	29
3 DESENVOLVIMENTO DO PROTÓTIPO	31
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	31
3.2 ESPECIFICAÇÃO	31
3.2.1 Diagrama de casos de uso	31
3.2.2 Diagrama de classes	32
3.2.3 Diagrama de atividades	35
3.3 IMPLEMENTAÇÃO	36
3.3.1 Técnicas e ferramentas utilizadas.....	36
3.3.2 Implementação do protótipo	37

3.3.2.1 Classe ConversorTelaAndroid.....	38
3.3.2.2 Classe ValidacaoComponentes	38
3.3.2.3 Classe GeradorTela.....	39
3.3.2.4 Classe ZipUtils.....	39
3.3.2.5 Classe DownloadUtils.....	40
3.3.2.6 Classe GeradorPaginaErro	40
3.3.3 <i>Templates</i> do FreeMaker.....	40
3.3.4 Validações JavaScript	42
3.3.5 Operacionalidade da implementação	42
3.4 RESULTADOS E DISCUSSÕES.....	49
4 CONCLUSÕES.....	51
4.1 EXTENSÕES	51
REFERÊNCIAS	53

1 INTRODUÇÃO

Nos dias de hoje, devido à evolução rápida e constante dos dispositivos móveis, é comum encontrar pessoas utilizando *smartphones* e *tablets*. Essa evolução abriu caminho para que desenvolvedores de software pudessem implementar as suas soluções para os dispositivos móveis, sem contar que: “Antes, o mercado de desenvolvimento para celulares era praticamente restrito aos fabricantes e operadoras que controlavam a criação e inclusão dos aplicativos em seus aparelhos. [...]” (MONTEIRO, 2013, p. 11).

O Android é o sistema operacional para dispositivos móveis mais utilizado no mundo, pois está presente em mais de um bilhão de *smartphones* e *tablets* (ANDROID DEVELOPERS, 2015). De acordo com os dados mais recentes da International Data Corporation (IDC), empresa especializada em tecnologia da informação e responsável por realizar análise de mercado, o Android vendeu mais de 255 milhões de unidades e conta com cerca de 85% do mercado somente no segundo trimestre de 2014 (INTERNATIONAL DATA CORPORATION, 2014). Uma desvantagem com relação à popularidade do Android é a falta de padronização dos aplicativos devido ao grande número de modelos existentes para esta plataforma. Em uma análise recente feita pela *OpenSignal*, foram identificados 24.093 modelos diferentes com Android (MOBILE TIME, 2015).

Com o crescimento do mercado *mobile*, cresceu o número de *frameworks* específicos para dispositivos móveis, sejam eles iOS, Android ou Windows Phone, que são as marcas dominantes do mercado *mobile* atualmente (WEB SOCIAL DEV, 2014). Em termos de compatibilidade, o Android foi projetado para funcionar em diferentes dispositivos como celulares, *tablets* e televisores. Para que o aplicativo seja bem sucedido em todos esses dispositivos, deve ser fornecida uma interface de usuário flexível, que se adapte a diferentes configurações de tela (ANDROID DEVELOPERS, 2015).

Buscando acelerar o desenvolvimento de aplicativos, o *Software Development Kit* (SDK) do Android fornece *templates* para a geração de projetos com estrutura básica ou com a adição de componentes em projetos existentes. Os *templates* de código fornecidos pelo SDK seguem as diretrizes de *design* e desenvolvimento para Android (ANDROID DEVELOPERS, 2015).

Diante do exposto, este trabalho propõe-se a criar um protótipo de ferramenta para gerar interfaces gráficas Android, contendo os principais componentes da plataforma.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo de ferramenta web para a geração de interfaces gráficas Android a partir de *templates* predefinidos.

Os objetivos específicos do trabalho são:

- a) disponibilizar uma ferramenta web que inclua opções de criação de interfaces gráficas para a plataforma Android;
- b) permitir que os usuários informem os parâmetros das interfaces a serem geradas;
- c) testar as telas geradas importando as mesmas na *Integrated Development Environment* (IDE) Android Studio e executando em dispositivos Android.

1.2 ESTRUTURA DO TRABALHO

O trabalho foi construído e estruturado em quatro capítulos. O segundo capítulo abrange a fundamentação teórica necessária para fornecer um melhor embasamento e entendimento do trabalho desenvolvido e do tema abordado.

O capítulo três apresenta o desenvolvimento do protótipo, mostrando a especificação e os principais requisitos a serem cumpridos no trabalho. Para isso, diagramas de casos de uso, classes e atividades foram criados. Ainda nesse capítulo são apresentadas as tecnologias utilizadas na implementação, a descrição da operacionalidade do protótipo desenvolvido e, por fim, os resultados alcançados.

O quarto e último capítulo trata das conclusões obtidas após o desenvolvimento do trabalho e as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os aspectos teóricos relacionados à contextualização do trabalho. Na seção 2.1 é feita uma introdução aos Geradores de Código. Na seção 2.2 é feita uma descrição da estrutura de projetos em Android. Na seção 2.3 são apresentados os principais componentes de tela do Android. Por fim, na seção 2.4 são abordados dois trabalhos correlatos ao trabalho atual.

2.1 INTRODUÇÃO AOS GERADORES DE CÓDIGO

A geração de código é uma técnica utilizada para a construção de programas. Não há nenhum estilo específico para as ferramentas de geração de código (DALGARNO, 2006 apud SILVEIRA, 2006, p. 16): 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; podem ter um número ilimitado de entradas e saídas.

Segundo Dalgarno (2006 apud SILVEIRA, 2006, p. 16), as principais vantagens das técnicas de geração de código são:

- a) qualidade: a qualidade do código construído por um gerador está relacionada diretamente à qualidade do código ou dos *templates* usados. Quando a qualidade desse código ou dos *templates* é aumentada, a qualidade do código gerado também é aumentada;
- b) consistência: o código criado por geradores de código é consistente. O nome das variáveis, dos métodos e das classes, por exemplo, é padronizado em todo o código gerado. Isso faz com que o código construído seja fácil de ser compreendido e, conseqüentemente, de ser usado. Além disso, é mais fácil de agregar funcionalidades a um código consistente;
- c) abstração: alguns geradores constroem códigos baseados em modelos abstratos do código alvo. Pode-se, por exemplo, construir uma camada de acesso a base de dados, a partir de uma definição em *eXtensible Markup Language* (XML) das tabelas, dos campos, dos relacionamentos e das consultas a uma base de dados. O valor desta abstração é que o gerador pode ser redirecionado para construir o código para outra plataforma. Com isto obtém-se portabilidade entre plataformas;
- d) produtividade: é fácil reconhecer o benefício inicial de um gerador de código. Na primeira vez que o gerador é executado, a partir de uma entrada contendo os modelos abstratos, quase instantaneamente se obtém o código de saída que implementa a especificação contida nesses modelos. Entretanto, o ganho real de

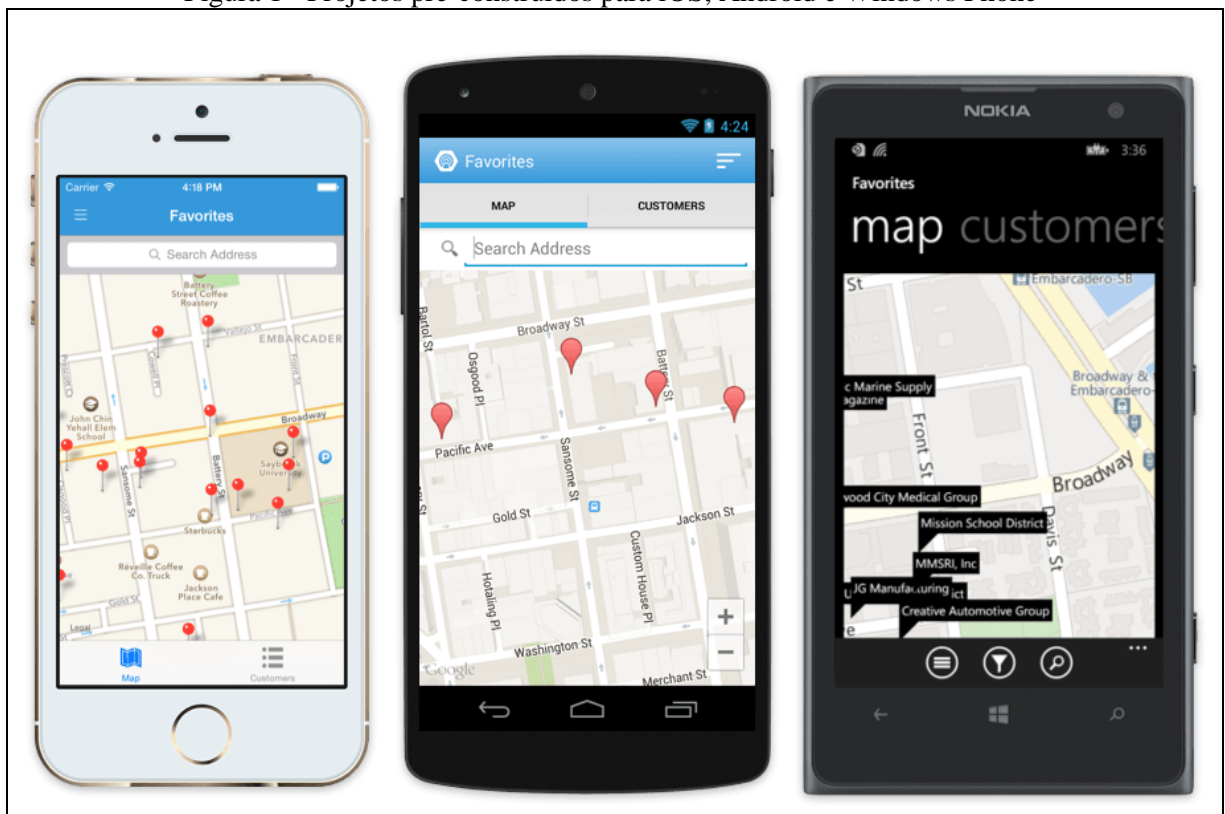
produtividade começa quando o gerador é executado várias vezes para gerar novos códigos baseados em mudanças na especificação definida.

Existem geradores para uma ampla variedade de aplicações. São comuns os geradores de interface com o usuário e os de acesso a bases de dados. A codificação da camada de interface com o usuário é razoavelmente repetitiva, então é comum a utilização de geradores para simplificar esta codificação e torná-la um trabalho menos manual. Além de reduzir erros na escrita, um gerador de interface pode também fornecer uma implementação para múltiplas linguagens ou plataformas (DALGARNO, 2006 apud SILVEIRA, 2006, p. 18). A seguir são apresentados alguns exemplos de geradores de código para dispositivos móveis disponíveis no mercado: Xamarin Studio, Smartface App Studio e Titanium Studio.

2.1.1 EXEMPLOS DE GERADORES DE CÓDIGO COMERCIAIS

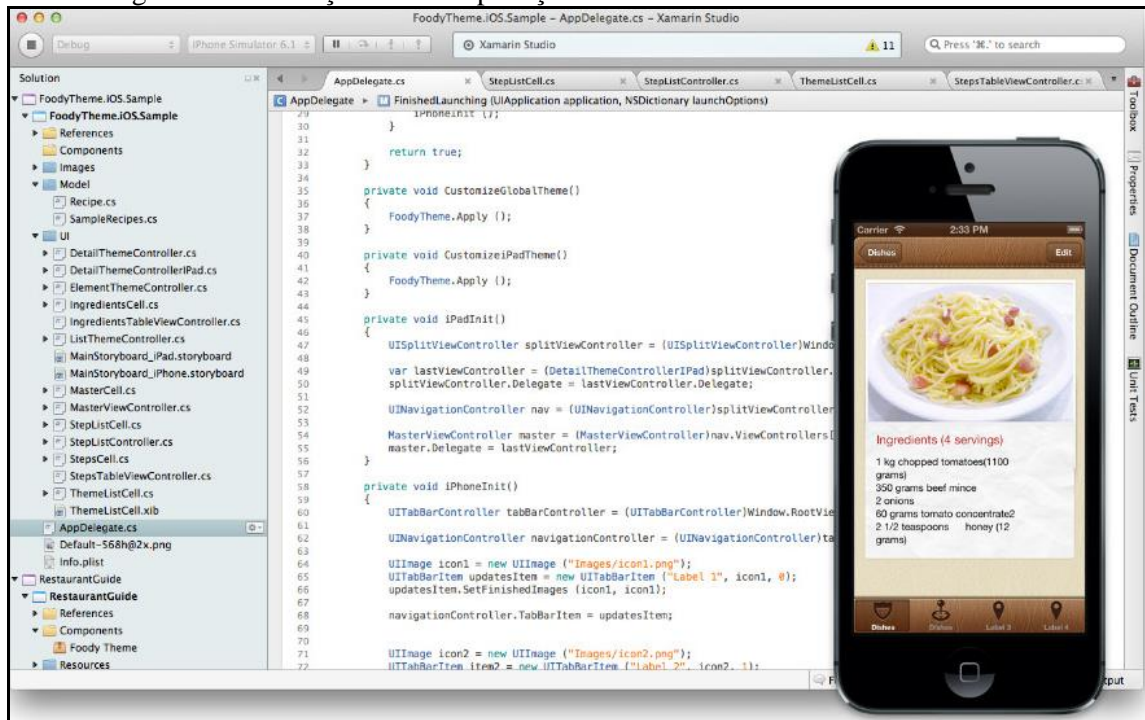
O Xamarin Studio é uma ferramenta de desenvolvimento capaz de gerar aplicativos completos com código-fonte nativo para as plataformas Android, iOS e Windows Phone a partir da linguagem C# (XAMARIN INC, 2015). A Figura 1 representa projetos pré-construídos para iOS, Android e Windows Phone disponíveis no site do fabricante. Enquanto a Figura 2 mostra a ferramenta sendo utilizada na construção de um projeto.

Figura 1 - Projetos pré-construídos para iOS, Android e Windows Phone



Fonte: Xamarin Inc (2015).

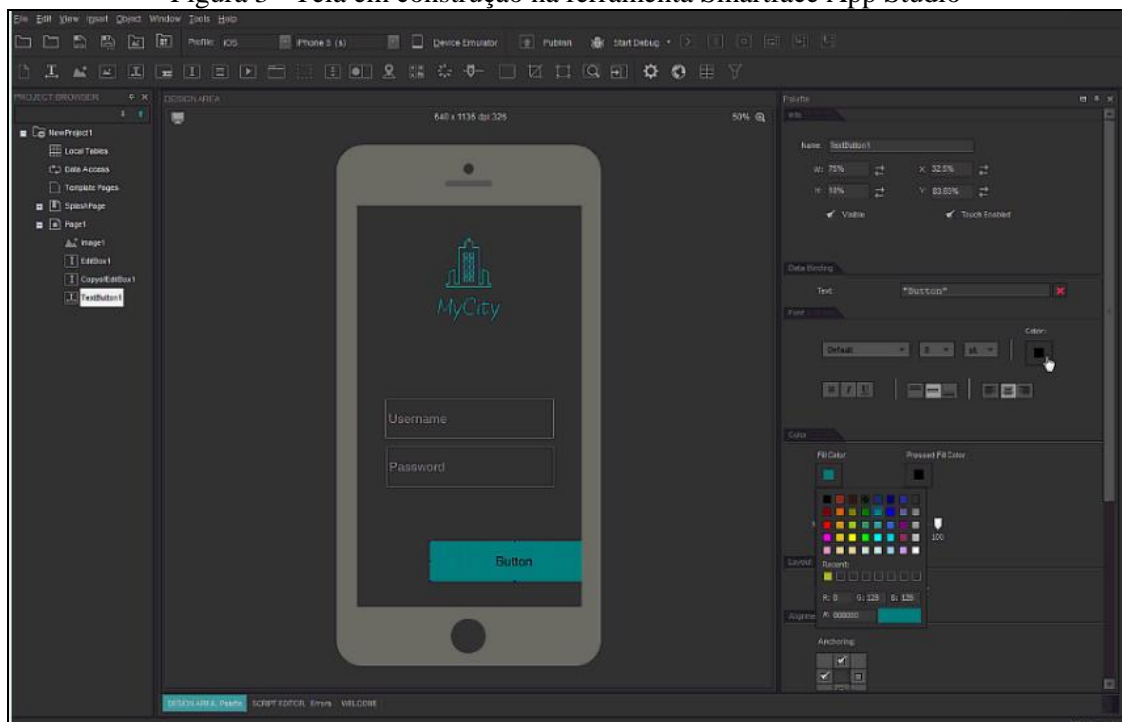
Figura 2 - Construção de uma aplicação utilizando a ferramenta Xamarin Studio



Fonte: Xamarin Inc (2015).

Smartface App Studio é uma ferramenta que permite ao desenvolvedor projetar, desenvolver e publicar aplicativos nativos para Android e iOS a partir da linguagem JavaScript (SMARTFACE, 2015). A Figura 3 mostra uma tela sendo construída na ferramenta.

Figura 3 - Tela em construção na ferramenta Smartface App Studio



Fonte: Smartface Inc (2015).

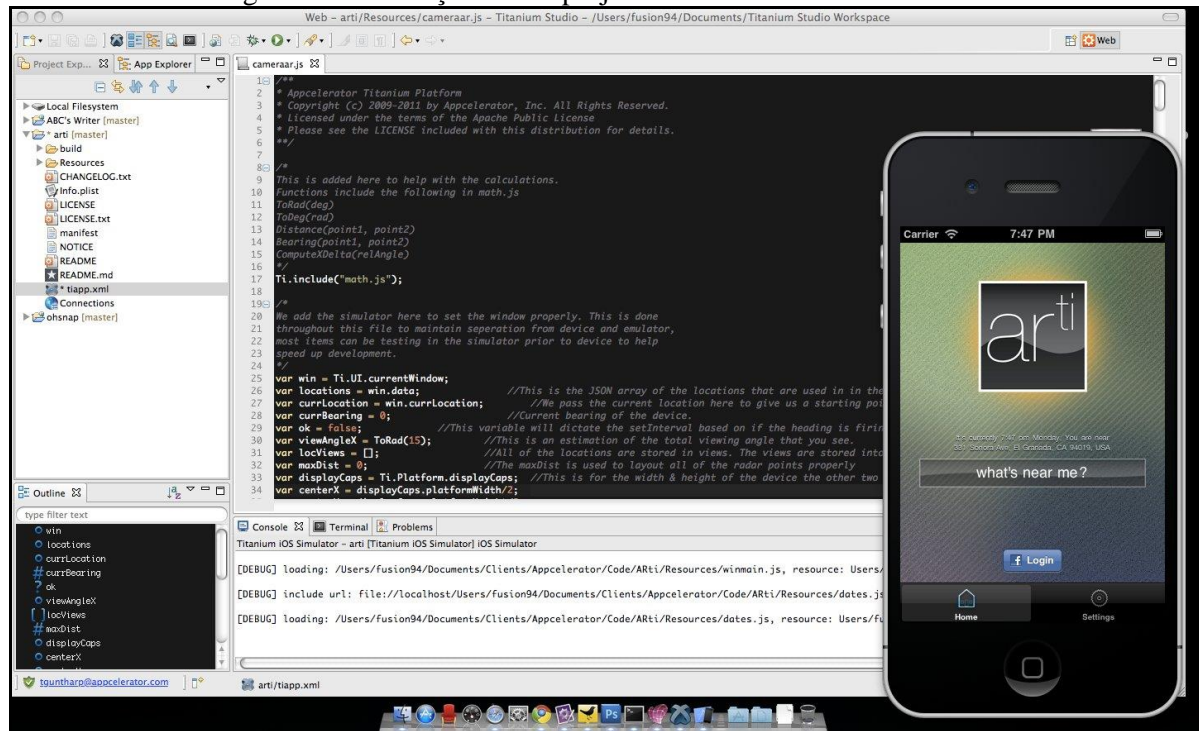
Titanium Studio é uma das ferramentas mais populares para desenvolvimento de aplicativos móveis multiplataforma. Esse *framework* permite a geração de interfaces gráficas nativas para as plataformas Android e iOS (APPCELERATOR, 2015). A Figura 4 mostra uma aplicação gerada para iOS e para Android utilizando o Titanium Studio. Enquanto a Figura 5 mostra a ferramenta sendo utilizada para construir um projeto.

Figura 4 - Aplicações iOS e Android geradas com Titanium



Fonte: Appcelerator Inc (2015).

Figura 5 - Construção de um projeto utilizando o Titanium Studio



Fonte: Appcelerator Inc (2015).

2.2 ESTRUTURA DE PROJETOS ANDROID

Um projeto Android é composto principalmente de arquivos Java e XML. Dentre as utilizações desses arquivos, as mais importantes são: a definição da interface do usuário e o mapeamento dos componentes de aplicação do sistema e dos atributos utilizados pela aplicação. Todo o comportamento de uma tela no Android é implementada em uma classe de atividade. O mapeamento dos componentes e a declaração das permissões de acesso, entre outras informações, são feitas através do arquivo `AndroidManifest.xml`, que por sua vez está localizado no diretório base do projeto Android. Por outro lado, os arquivos de interface, as listas pré-definidas e as imagens, entre outros recursos utilizados pela aplicação, são organizados em subpastas a partir do diretório `res`, destinado aos recursos do projeto Android (ANDROID DEVELOPERS, 2015). A seguir são descritos os principais componentes que formam projetos Android: Atividades, Android Manifest e recursos.

2.2.1 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. O Quadro 1 mostra uma classe Java representando uma atividade (ANDROID DEVELOPERS, 2015).

Quadro 1 - Representação de uma atividade em Java

```
public class MainActivity extends AppCompatActivity {

    private MainFragment mainFragment;

    @Override
    public void onCreate(Bundle savedInstanceState, PersistableBundle
persistentState) {
        super.onCreate(savedInstanceState, persistentState);
        setContentView(R.layout.act_single_frame);

        initFragment();
    }

    private void initFragment() {
        FragmentManager fm = getSupportFragmentManager();
        mainFragment = (MainFragment)
fm.findFragmentByTag(MainFragment.class.getName());

        if (mainFragment == null) {
            mainFragment = new MainFragment();
            addFragment(R.id.act_single_frame, mainFragment,
MainFragment.class.getName());
        }
    }

    private void addFragment(int id, Fragment fragment, String tag) {
        FragmentTransaction ft =
getSupportFragmentManager().beginTransaction();
        ft.add(id, fragment, tag);
        ft.commit();
    }
}
```

Fonte: Android Developers (2015).

2.2.2 Android Manifest

Toda aplicação Android deve possuir um arquivo de manifesto, chamado `AndroidManifest.xml`. Nele estão presentes informações essenciais sobre a aplicação e necessárias para o Sistema Operacional Android. O arquivo de manifesto organiza uma aplicação em uma estrutura bem definida, a qual é compartilhada por todas as aplicações e permite que o Sistema Operacional Android carregue-as e execute-as em um ambiente gerenciado (MEDNIEKS et al., 2011).

Entre as principais declarações do arquivo de manifesto, podem ser destacadas as permissões de usuário, declaração do nível mínimo de *Application Programming Interface* (API) necessário para rodar a aplicação, declaração dos dispositivos de hardware utilizados pela aplicação, bibliotecas em que o aplicativo precisa se conectar e, principalmente, a

declaração dos componentes de aplicação utilizados, como atividades e serviços. No Quadro 2 é exibido um arquivo exemplo de manifesto de uma aplicação (MEDNIEKS et al., 2011).

Quadro 2 - 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.ch03.app"
    android:versionCode="1"
    android:versionName="1.0">

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

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

        <activity android:name=".MainActivity"
            android:label="Main Activity">

            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>

        </activity>

        <receiver
            android:name=".MainBroadcastReceiver"
            android:label="Main Broadcast Receiver"/>
    </application>

    <uses-sdk android:minSdkVersion="7" />
</manifest>
```

Fonte: Mednieks et al. (2011).

2.2.3 Recursos (*resources*)

Além do código fonte, as aplicações Android podem demandar espaço de armazenamento para grandes quantidades de dados necessários 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, devem ser mantidas separadas do código (MEDNIEKS et al., 2011). No Quadro 3 é exibido um arquivo exemplo contendo três recursos de cores sendo aplicados.

Quadro 3 - Arquivo button_color.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:state_pressed="true"
        android:color="#ffff0000"/>

    <item android:state_focused="true"
        android:color="#ff0000ff"/>
```

```
<item android:color="#ff000000"/>
</selector>
```

Fonte: Android Developers (2015).

2.3 PRINCIPAIS COMPONENTES DE TELA DO ANDROID

O Android fornece uma variedade de componentes de interface gráfica para permitir a interação com o usuário, tais como botões, listas, campos de texto, caixas de diálogos, entre outros (ANDROID DEVELOPERS, 2015). A seguir são descritos os principais componentes de tela do Android: Button, Card, RecyclerView, Toolbar, EditText, TextView e Checkbox.

2.3.1 Button

Button é um componente responsável por efetuar ações quando selecionado. Consiste, geralmente, de um texto, uma imagem, ou ambos, concebido de acordo com a cor do aplicativo. Os dois tipos de botões mais comuns são: Rectangular Button e Floating Action Button (GOOGLE DESIGN, 2015). Na Figura 6 são apresentados um Rectangular Button e um Floating Action Button.

Figura 6 – Exemplo de um Rectangular Button e Floating Action Button



Fonte: Google Design (2015).

No Quadro 4 é apresentado o *template* de código com um Rectangular Button e um Floating Action Button contendo a definição de seus identificadores, largura, altura e outros atributos conforme apresentados na Figura 6.

Quadro 4 - *Template* de código de um Rectangular Button e um Floating Action Button

```
<Button
  style="@style/PrimaryButton"
  android:id="@+id/button1"
  android:layout_height="wrap_content"
  android:layout_width="wrap_content"
  android:text="Button"/>
```

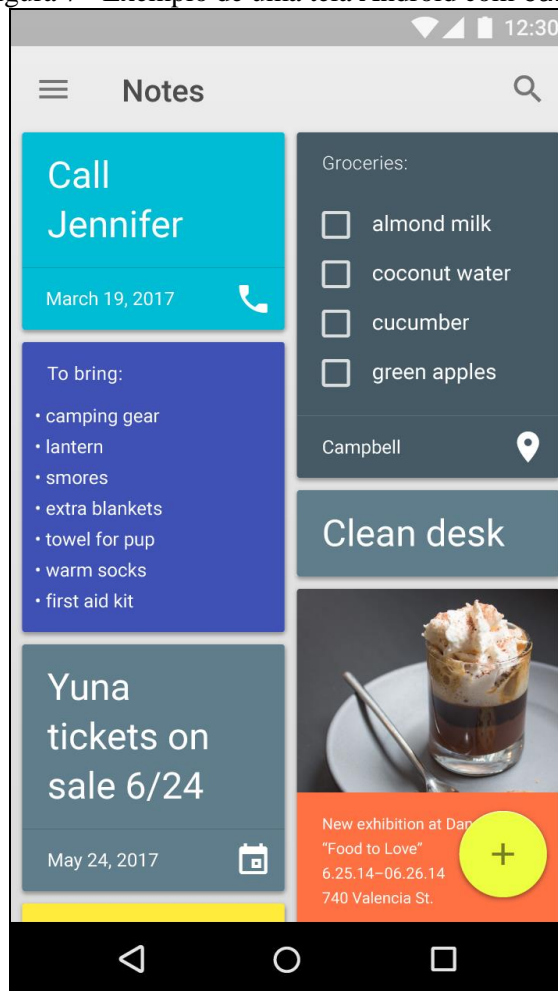
```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:clickable="true"
    android:src="@drawable/ic_add"/>
```

Fonte: Android Developers (2015).

2.3.2 Card

Cards são componentes responsáveis por agrupar informações detalhadas sobre um determinado contexto (GOOGLE DESIGN, 2015). Na Figura 7 são apresentados alguns Cards em cores diferentes contendo textos e imagens.

Figura 7 - Exemplo de uma tela Android com Cards



Fonte: Google Design (2015).

No Quadro 5 é apresentado o *template* utilizado como referência na implementação dos cards da Figura 7, contendo a definição de seu identificador, largura, altura e outros atributos.

Quadro 5 - *Template* de código de um Card

```
<android.support.v7.widget.CardView
```



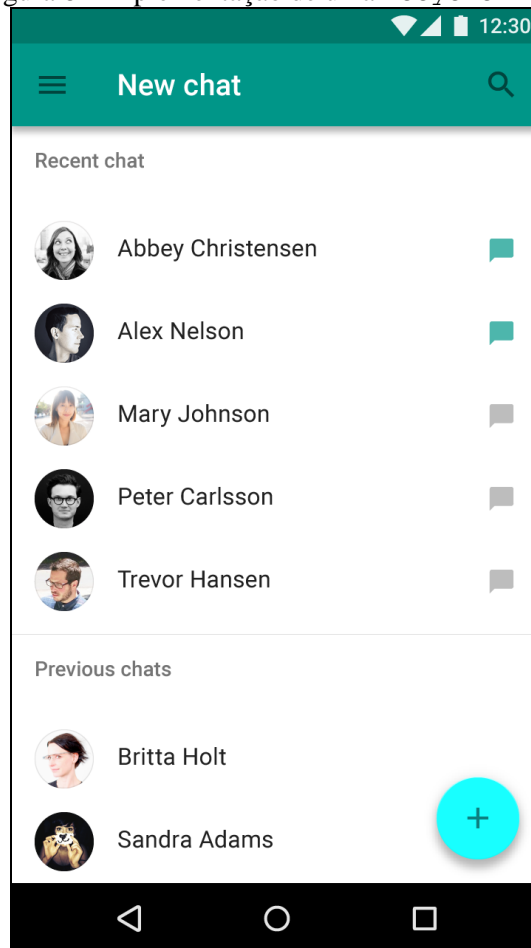
```
xmlns:card_view="http://schemas.android.com/apk/res-auto"
android:id="@+id/card_view"
android:layout_gravity="center"
android:layout_width="200dp"
android:layout_height="200dp"
card_view:cardCornerRadius="4dp" />
```

Fonte: Android Developers (2015).

2.3.3 RecyclerView

RecyclerView é responsável por apresentar vários itens em um arranjo vertical como um único elemento contínuo (GOOGLE DESIGN, 2015). Na Figura 8 é apresentada a implementação de uma RecyclerView contendo uma lista de contatos com imagens e textos.

Figura 8 - Implementação de uma RecyclerView



Fonte: Google Design (2015).

No Quadro 6 é apresentado o *template* de código da RecyclerView apresentada na Figura 8, contendo a definição de seu identificador, largura e altura.

Quadro 6 - *Template* de código da RecyclerView

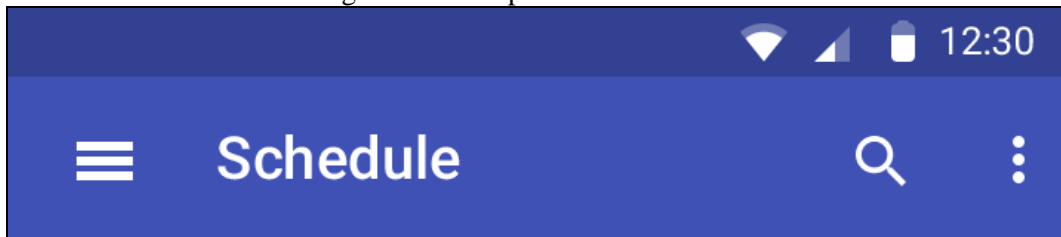
```
<android.support.v7.widget.RecyclerView
android:id="@+id/my_recycler_view"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

Fonte: Android Developers (2015).

2.3.4 Toolbar

A `Toolbar` representa uma barra de ferramentas posicionada na parte superior das telas do Android. Geralmente, contém uma descrição a respeito da tela na qual está sendo exibida e algumas opções adicionais (GOOGLE DESIGN, 2015). A Figura 9 apresenta um exemplo de uma `Toolbar` contendo um título e uma opção de pesquisa.

Figura 9 - Exemplo de uma `Toolbar`



Fonte: Google Design (2015).

No Quadro 7 é apresentado o *template* de código da `Toolbar` apresentada na Figura 9, contendo a definição de sua largura, altura e outros atributos.

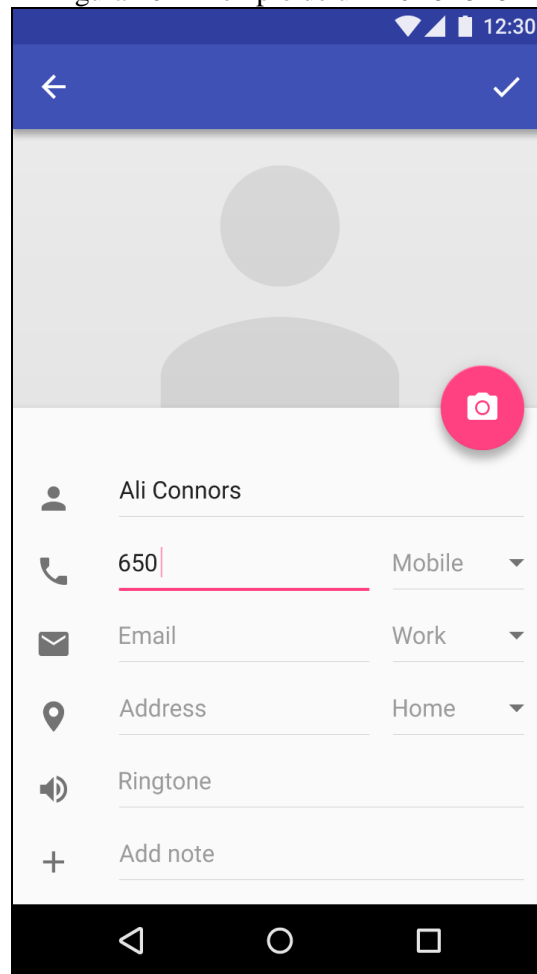
Quadro 7 - *Template* de código de uma `Toolbar`

```
<android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary" />
```

Fonte: Android Developers (2015).

2.3.5 EditText

Os componentes `EditText` permitem ao usuário inserir e editar textos. Ao tocar em um `EditText`, o sistema exibe automaticamente o teclado, que por sua vez, se adapta ao tipo preestabelecido do campo (texto ou numérico) (GOOGLE DESIGN, 2015). A Figura 10 apresenta alguns `EditTexts` utilizados na inserção de um registro.

Figura 10 - Exemplo de um `EditText`

Fonte: Google Design (2015).

No Quadro 8 é apresentado o *template* de código de um `EditText` utilizado como referência para a implementação dos componentes apresentados na Figura 10, contendo a definição de seu identificador, largura e altura.

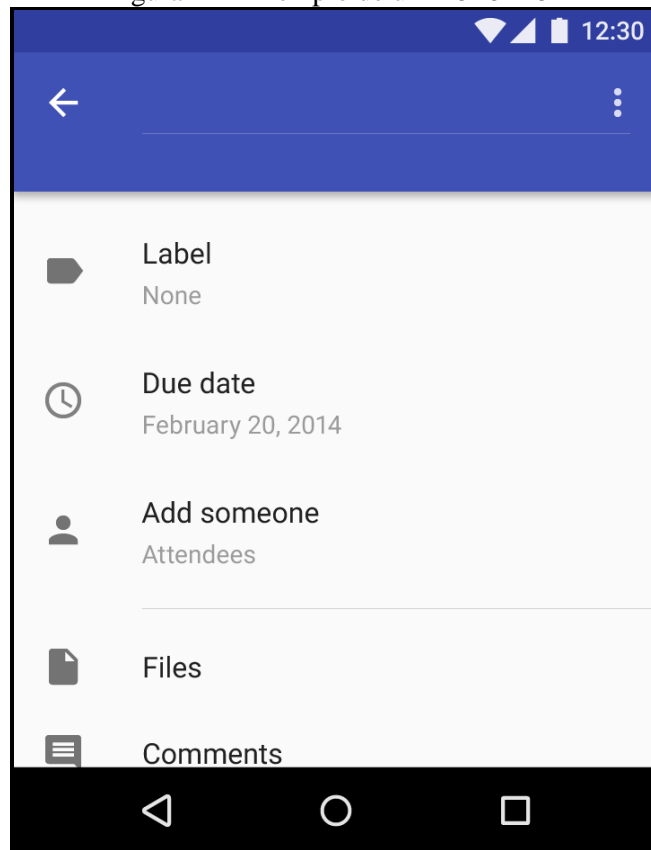
Quadro 8 - *Template* de código de um `EditText`

```
<EditText
    android:id="@+id/edit_text_form"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Fonte: Android Developers (2015).

2.3.6 `TextView`

Os componentes `TextView` são responsáveis por exibir textos para o usuário (ANDROID DEVELOPERS, 2015). A Figura 11 apresenta exemplos de `TextViews` representando legendas para campos de texto.

Figura 11 - Exemplo de um `TextView`

Fonte: Google Design (2015).

No Quadro 9 é apresentado o *template* de código de um `TextView` utilizado como referência para a implementação dos componentes apresentados na Figura 11, contendo a definição de seu identificador, texto, largura, altura e outros atributos.

Quadro 9 - *Template* de código do componente `TextView`

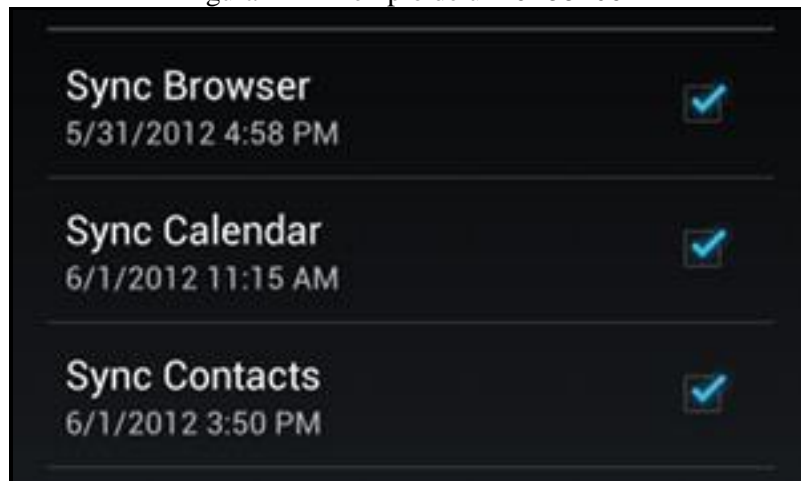
```
<TextView
    android:id="@+id/text_id"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Label"
    android:textColor="@android:color/primary_text_dark"
    android:textSize="16sp"/>
```

Fonte: Android Developers (2015).

2.3.7 Checkbox

Os componentes `Checkbox` permitem que o usuário selecione uma ou mais opções de um conjunto. Normalmente, são apresentados em uma lista vertical (ANDROID DEVELOPERS, 2015). A Figura 12 apresenta um exemplo de `Checkbox` em uma lista.

Figura 12 - Exemplo de um Checkbox



Fonte: Android Developers (2015).

No Quadro 10 é apresentado o *template* de código utilizado como referência para a implementação dos componentes `CheckBox` apresentados na Figura 12, contendo a definição de seu identificador, largura, altura e outros atributos.

Quadro 10 - *Template* de código de um `CheckBox`

```
<CheckBox
    android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

Fonte: Android Developers (2015).

2.4 TRABALHOS CORRELATOS

Nesta seção são descritos dois trabalhos correlatos ao trabalho proposto: um gerador de interfaces Android a partir do Delphi, proposto por Rezini (2013), e um gerador de código a partir de especificações com padrões de requisitos, proposto por Battisti (2014).

2.4.1 Gerador de interfaces Android a partir do Delphi

Rezini (2013) elaborou um conversor de interfaces gráficas do ambiente Delphi para Android. Além da conversão de interfaces a ferramenta faz a geração de um projeto, a compilação do mesmo e a instalação do arquivo compilado em uma máquina virtual ou dispositivo Android. A Figura 13 ilustra uma interface desenvolvida a partir do *template* em Delphi (esquerda) e o resultado obtido na conversão exibido em uma máquina virtual Android (direita).

Figura 13 - Formulário desenvolvido em Delphi e resultado equivalente em Android



Fonte: Rezini (2013, p. 62).

A ferramenta 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. Possibilita também que o desenvolvedor que utiliza a 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.

2.4.2 Elicitar: protótipo de gerador de código a partir de especificações com padrões de requisitos

Battisti (2014) elaborou um gerador de código baseado em especificações com padrões de requisitos escritos em língua portuguesa. O processo definido neste trabalho consiste em quatro etapas: cadastrar os requisitos conforme os padrões de requisitos, utilizar processamento de linguagem natural para identificar componentes de interfaces, gerar arquivo de definição e gerar o código a partir desse arquivo de definição.

Na primeira etapa do processo são utilizados os padrões de requisitos: tipo de dados, estrutura de dados e entidade ativa. Na etapa seguinte é utilizado um analisador morfológico. A próxima etapa gera arquivos JavaScript *Object Notation* (JSON) ou XML que descrevem um formulário com base nas informações obtidas na etapa anterior contendo os componentes

de interface identificados. Por último, os formulários HTML são gerados. Na Figura 14 é possível visualizar a tela principal da ferramenta.

Figura 14 - Tela principal da ferramenta



Fonte: Battisti (2014, p. 42).

Um aspecto positivo que Battisti (2014) observou na etapa de testes, foi o ganho obtido em produtividade com a especificação feita utilizando padrões de requisitos, já que os mesmos se assemelham a padrões de desenvolvimento, o que gera reusabilidade dos requisitos cadastrados na ferramenta. Um aspecto negativo foi que a ferramenta elaborada ficou restrita ao reconhecimento de um domínio pequeno de padrões de requisitos e de componentes de interfaces.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são apresentadas as etapas do desenvolvimento do protótipo. Na seção 3.1 são enumerados os principais requisitos da ferramenta a ser desenvolvida. A seção 3.2 apresenta sua especificação. A seção 3.3 mostra os detalhes da implementação e, por fim, a seção 3.4 exibe os resultados obtidos no protótipo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Para a implementação do protótipo, foram identificados os seguintes requisitos:

- a) disponibilizar uma paleta contendo os principais componentes de interface do Android (Requisito Funcional - RF);
- b) permitir adicionar, remover e alterar os componentes de tela (RF);
- c) permitir alterar os atributos dos componentes adicionados (RF);
- d) permitir o *download* dos arquivos gerados (RF);
- e) ser implementada utilizando a linguagem de programação Java 7 (Requisito Não Funcional - RNF);
- f) utilizar o ambiente de desenvolvimento Eclipse 4.5 (RNF);
- g) utilizar HTML, CSS e JavaScript para construir a interface com o usuário (RNF);
- h) utilizar o *framework* jQueryUI para permitir arrastar e soltar os componentes (RNF);
- i) utilizar o *framework* Freemarker para a geração dos arquivos (RNF);
- j) criar diagramas de especificação na ferramenta EA (RNF).

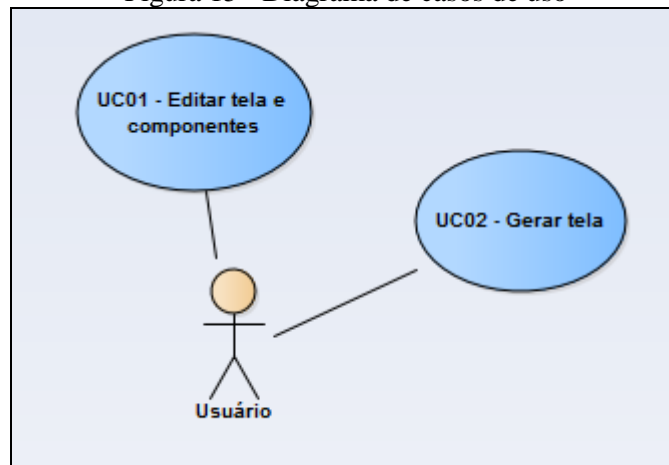
3.2 ESPECIFICAÇÃO

A seguir é apresentada a especificação do protótipo, representado por diagramas da *Unified Modeling Language* (UML) e modelada utilizando a ferramenta Enterprise Architect 11. Os diagramas utilizados foram o de casos de uso, classes e atividades.

3.2.1 Diagrama de casos de uso

Nesta seção são apresentados na Figura 15 os casos de uso da ferramenta. Identificou-se somente um ator, denominado usuário, o qual utiliza todas as funcionalidades da ferramenta.

Figura 15 - Diagrama de casos de uso



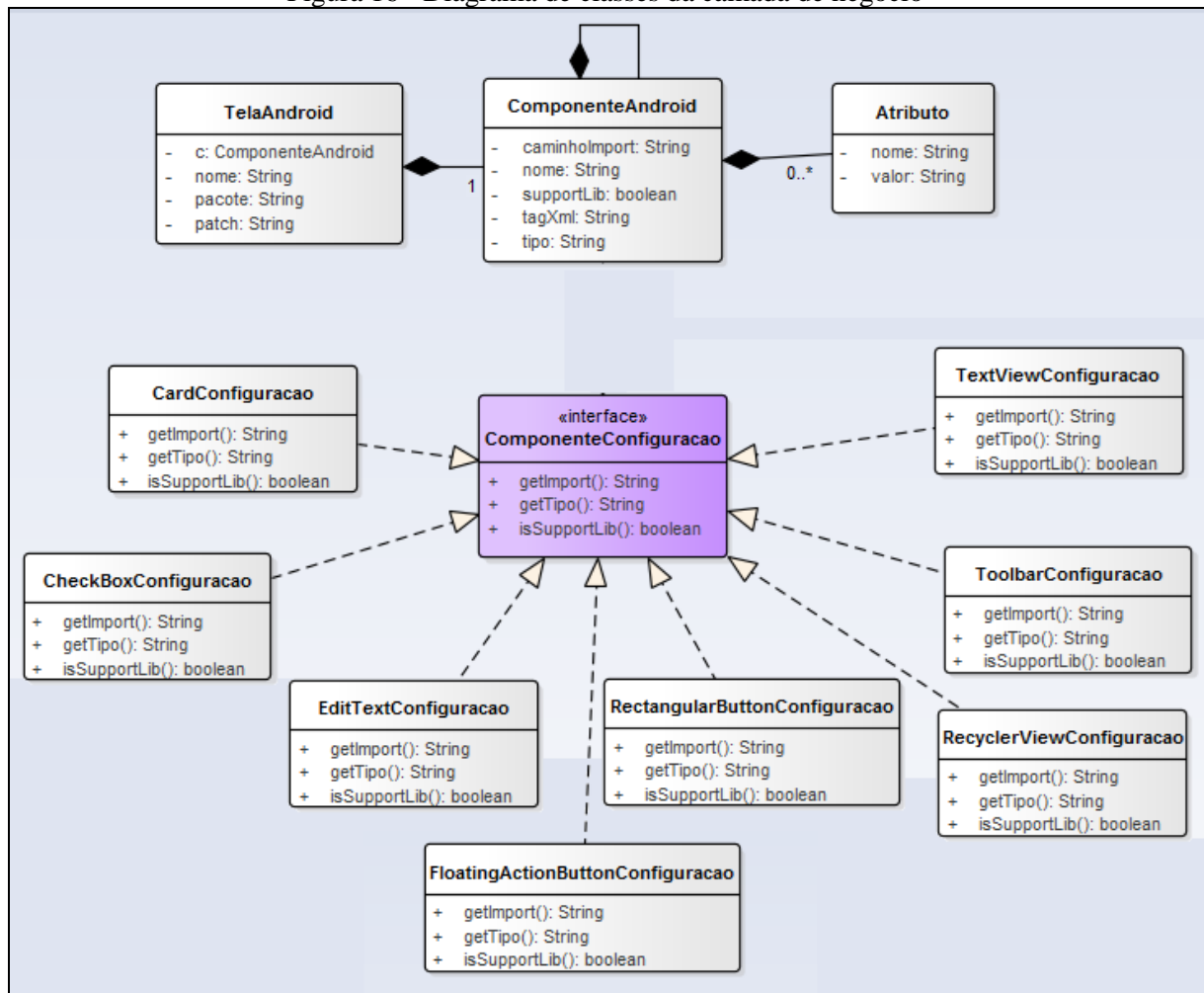
A seguir são descritos os casos de uso exibidos na Figura 15:

- a) UC01 - Editar tela e componentes: permite adicionar, remover e alterar os componentes na tela a ser gerada, tal como seus atributos específicos;
- b) UC02 - Gerar tela: permite gerar e efetuar o *download* dos arquivos de tela a partir dos parâmetros e componentes adicionados.

3.2.2 Diagrama de classes

Os diagramas apresentados nesta seção apresentam as principais classes do protótipo. Na Figura 16 podem-se observar as classes pertencentes à camada de negócio. Esta camada tem como responsabilidade transcrever os componentes apresentados.

Figura 16 - Diagrama de classes da camada de negócio



As classes apresentadas na Figura 16 são descritas a seguir:

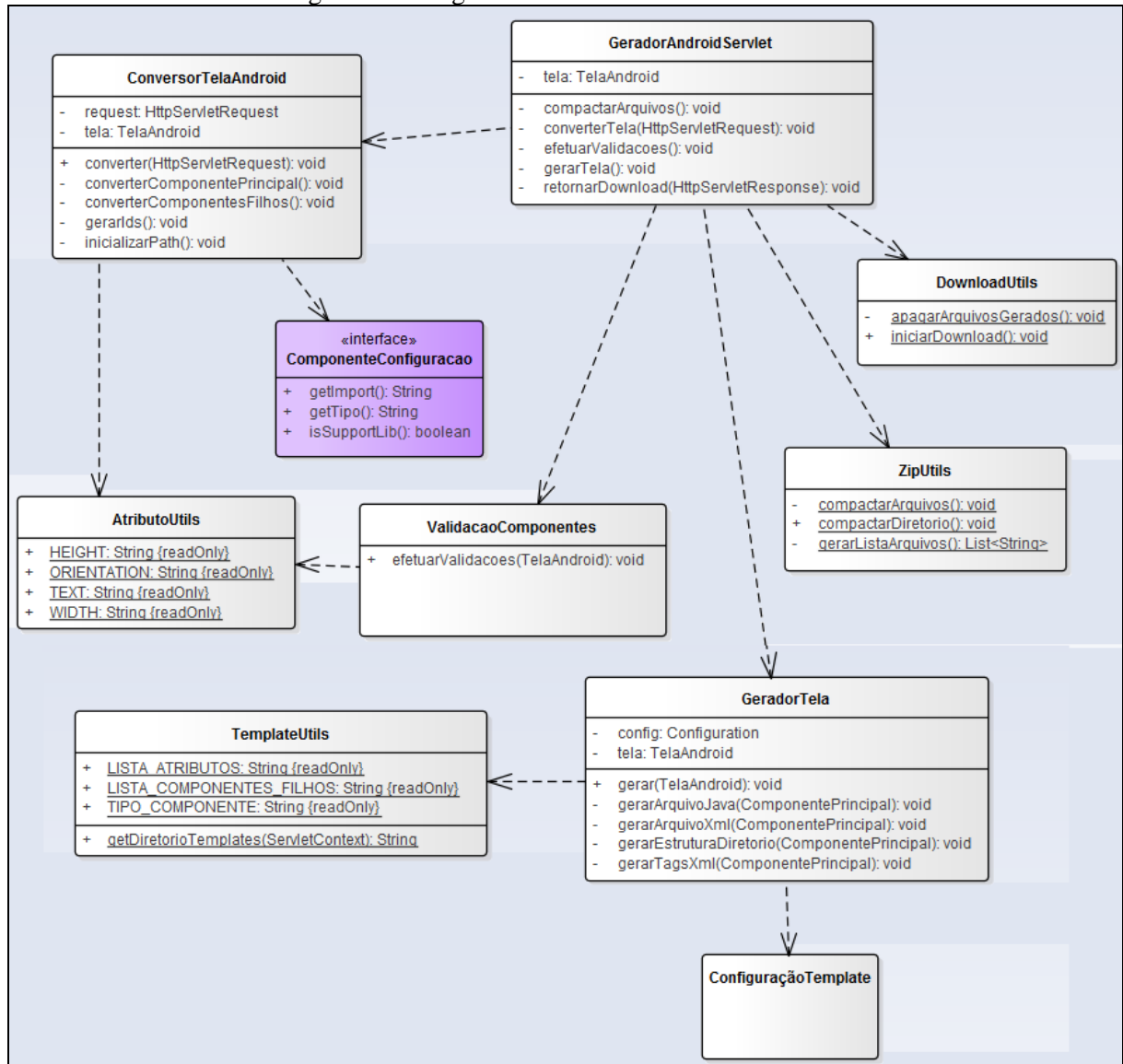
- a) **TelaAndroid**: armazena as informações referentes à estrutura da tela a ser gerada;
- b) **ComponenteAndroid**: armazena as informações referentes ao componente a ser gerado;
- c) **Atributo**: armazena as informações dos atributos a serem adicionados nos componentes;
- d) **ComponenteConfiguracao**: define a interface de implementação para a configuração dos componentes Android;
- e) **CardConfiguracao**: armazena informações referentes à configuração do componente `CardView`;
- f) **CheckBoxConfiguracao**: armazena informações referentes à configuração do componente `CheckBox`;
- g) **EditTextConfiguracao**: armazena informações referentes à configuração do componente `EditText`;

- h) `FloatingActionButtonConfiguracao`: armazena informações referentes ao componente `FloatingActionButton`;
- i) `RectangularButtonConfiguracao`: armazena informações referentes à configuração do componente `RectangularButton`;
- j) `RecyclerViewConfiguracao`: armazena informações referentes à configuração do componente `RecyclerView`;
- k) `ToolBarConfiguracao`: armazena informações referentes à configuração do componente `ToolBar`;
- l) `TextViewConfiguracao`: armazena informações referentes à configuração do componente `TextView`.

Na Figura 17 é exibido outro diagrama de classes, indicando as classes que são responsáveis por realizar as funções disponibilizadas pelo protótipo, ou seja, que fazem o intercâmbio entre as requisições do usuário com os recursos do sistema. As classes são:

- a) `GeradorAndroidServlet`: responsável por receber as requisições do cliente;
- b) `ConversorTelaAndroid`: responsável por converter a requisição do cliente em objetos Java;
- c) `ValidacaoComponentes`: responsável por validar os componentes convertidos a partir da requisição do cliente;
- d) `GeradorTela`: responsável por gerar a tela;
- e) `DownloadUtil`: responsável por fornecer o *download* dos arquivos gerados para o usuário;
- f) `ConfiguracaoTemplate`: responsável pela configuração inicial do *framework* `FreeMarker`;
- g) `AtributoUtil`: responsável por manter o nome de atributos de componentes `Android`;
- h) `TemplateUtil`: responsável pelos parâmetros padrões de *templates* do `FreeMarker`;
- i) `ZipUtils`: responsável pela compactação dos arquivos gerados.

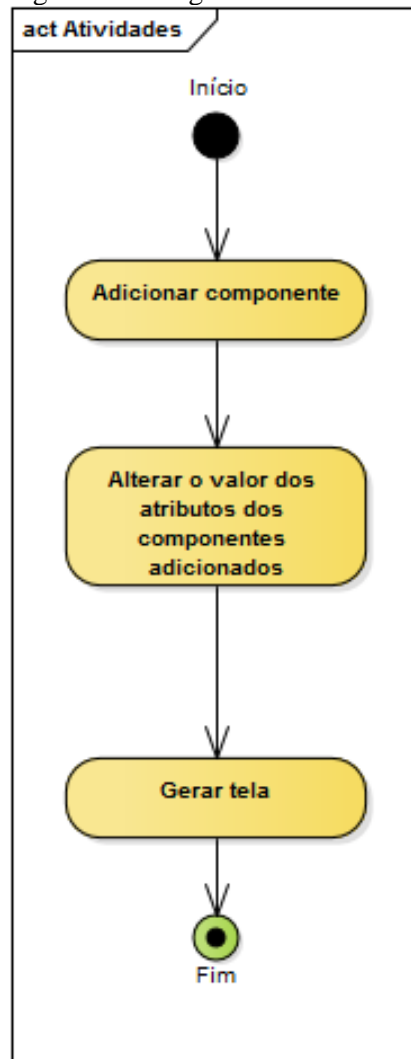
Figura 17 - Diagrama de classes de funcionalidades



3.2.3 Diagrama de atividades

Na Figura 18 tem-se o fluxo padrão do funcionamento do protótipo para a geração de tela. O processo se inicia com o usuário adicionando os componentes na tela. Em seguida o usuário deve alterar os atributos dos componentes adicionados. Por fim, ele pode gerar a tela.

Figura 18 - Diagrama de atividades



3.3 IMPLEMENTAÇÃO

A seguir estão detalhadas e elencadas as técnicas e ferramentas utilizadas na implementação, assim como as principais classes e rotinas implementadas durante o desenvolvimento da aplicação.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação do protótipo foram utilizados conceitos extraídos de Martin (2009). Em resumo, os conceitos apresentados no livro consistem em escrever métodos pequenos e bem descritivos, evitando assim a documentação excessiva dentro dos arquivos de código-fonte e permitindo o reaproveitamento de código em outros lugares. Foi optado em implementar o protótipo utilizando a língua portuguesa, pois, em ambientes corporativos, dado o contexto do Brasil, as regras de negócio têm se tornado cada vez mais complexas exigindo uma atenção constante por parte dos programadores para não cometerem erros no

código. Sendo assim, utilizar a língua portuguesa juntamente com os conceitos definidos no livro Código Limpo, proporcionam aos programadores um entendimento geral a respeito do que foi implementado.

Foi utilizado o ambiente de desenvolvimento Eclipse 4.5.1 e como linguagem de programação a linguagem Java na versão 7. As tecnologias utilizadas na implementação do protótipo foram:

- a) Apache TomCat 7: é um servidor de aplicações web de código aberto desenvolvido pela Apache Software Foundation. Ele implementa diversas especificações do Java EE incluindo Java Servlet, JavaServer Pages, entre outras;
- b) FreeMarker: é um *framework* Java para a geração de texto a partir de *templates*. Os *templates* são escritos em uma linguagem específica deste *framework* e contém a saída esperada pelo usuário. Originalmente, seu foco principal era a geração de páginas web utilizando a arquitetura MVC;
- c) jQuery: é um *framework* JavaScript de código aberto responsável por simplificar a interação entre os scripts de uma página e o seu HTML. Sua sintaxe foi desenvolvida para tornar mais simples a navegação em uma página, a seleção de elementos, criação de animações e manipulação de eventos;
- d) Genymotion: é um emulador Android desenvolvido pela Genymobile. Essa ferramenta é uma alternativa ao emulador nativo presente no SDK do Android. Ele fornece *plug-ins* para os ambientes de desenvolvimento Eclipse e Android Studio. Com ele é possível simular a localização de um GPS, variações na bateria do dispositivo, mensagens de texto, entre outras funcionalidades.

3.3.2 Implementação do protótipo

As requisições feitas pelo usuário são recebidas pela classe `GeradorAndroidServlet`. Essa classe é responsável por gerenciar as requisições feitas pelo cliente ao servidor. O Quadro 11 mostra a implementação do principal método desta classe, o método `service`.

Quadro 11 - Implementação do método `service`

```
@Override
protected void service (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    try {
        converterTela (request);
        efetuarValidacoes ();
        gerarTela ();
        compactarArquivos ();
        retornarDownload (response);
    }
}
```

```

    } catch (ValidacaoException e) {
        retornarPaginaErro(e.getMessage(), response);

    } catch (Exception e) {
        e.printStackTrace();
        retornarPaginaErro(e.getMessage(), response);
    }
}

```

Este método é responsável por receber requisições, utilizar classes específicas para efetuar as devidas conversões, validações, criação dos arquivos e retornar o resultado ao usuário. Se ocorrer alguma exceção irreversível durante a execução do método `service`, será gerada uma página ao usuário exibindo a descrição do erro. A seguir são descritas as classes utilizadas pelo método `service`: `ConversorTelaAndroid`, `ValidacaoComponentes`, `GeradorTela`, `ZipUtils`, `DownloadUtils` e `GeradorPaginaErro`.

3.3.2.1 Classe `ConversorTelaAndroid`

A classe `ConversorTelaAndroid` é responsável por realizar as conversões do conteúdo inserido pelo usuário. O Quadro 12 mostra a implementação do principal método desta classe, o método `converter`. Este método é responsável por converter para objetos Java os componentes Android recebidos pelo usuário e em seguida gerar identificadores para cada componente adicionado.

Quadro 12 - Implementação do método `converter`

```

public TelaAndroid converter(HttpServletRequest request) throws
ComponenteInvalidoException {
    this.request = request;
    this.tela = new TelaAndroid();

    converterComponentePrincipal();
    converterComponentesFilhos();
    gerarIds();

    return tela;
}

```

3.3.2.2 Classe `ValidacaoComponentes`

A classe `ValidacaoComponentes` é responsável por realizar as devidas validações para possibilitar que todos os componentes da tela sejam gerados da maneira correta. O Quadro 13 mostra a implementação do principal método desta classe, o método `efetuarValidacoes`. Este método é responsável por validar se todos os campos obrigatórios foram devidamente preenchidos. Caso alguns campos não tenham sido preenchidos, este método irá concatenar

mensagens de erro para cada campo que não foi preenchido e irá disparar uma exceção para o usuário.

Quadro 13 - Implementação do método `efetuarValidacoes`

```
public void efetuarValidacoes(TelaAndroid tela) throws ValidacaoException {
    ComponenteAndroid componente = tela.getComponentePrincipal();

    validarTela(tela);
    validarComponentePrincipal(componente);
    validarComponentesAdicionados(componente.getFilhos());

    if (!listaErros.isEmpty()) {
        String msgErro = concatenarDescricaoErros();
        throw new ValidacaoException(msgErro);
    }
}
```

3.3.2.3 Classe GeradorTela

A classe `GeradorTela` é a classe responsável por interpretar os objetos Java que representam os componentes convertidos, efetuar o devido processamento através de classes do *framework* `FreeMarker` e por fim gerar os arquivos que representam a tela. O Quadro 14 mostra a implementação do principal método desta classe, o método `gerar`. Este método é responsável pela geração da estrutura de diretórios na qual os arquivos ficarão armazenados temporariamente. Em seguida é realizada a geração das *tags* de cada componente que irão compor o arquivo XML da tela. Em seguida é feita a geração do arquivo XML da tela. Por fim, é feita a geração do arquivo Java com as devidas ligações com os componentes do arquivo XML.

Quadro 14 - Implementação do método `gerar`

```
public void gerar(TelaAndroid tela) throws TemplateNotFoundException,
    MalformedTemplateNameException, ParseException,
    IOException, TemplateException {

    this.tela = tela;
    ComponenteAndroid componente = tela.getComponentePrincipal();

    gerarEstruturaDiretorio(componente);
    gerarTagsXml(componente);
    gerarArquivoXml(componente);
    gerarArquivoJava(componente);
}
```

3.3.2.4 Classe ZipUtils

A classe `ZipUtils` é a classe responsável por compactar os arquivos gerados. O Quadro 15 mostra a implementação do principal método desta classe, o método

`compactarArquivos`. Este método é responsável por receber o caminho do diretório contendo os arquivos para serem compactados e efetuar as devidas compactações.

Quadro 15 - Implementação do método `compactarArquivos`

```
public static void compactarArquivos(String path) throws IOException {
    List<String> arquivos = new ArrayList<>();
    gerarListaArquivos(path, new File(path), arquivos);
    compactar(path, arquivos);
}
```

3.3.2.5 Classe `DownloadUtils`

A classe `DownloadUtils` é a classe responsável por possibilitar o download dos arquivos gerados ao usuário. O Quadro 16 mostra a implementação do principal método desta classe, o método `efetuarDownload`. Este método é responsável primeiramente por limpar qualquer dado que esteja armazenado em *buffer*. Em seguida é retornado o *download* dos arquivos compactados. Por fim, são apagados quaisquer arquivos gerados pela ferramenta.

Quadro 16 - Implementação do método `efetuarDownload`

```
public static void efetuarDownload(ServletContext context, TelaAndroid
tela, HttpServletResponse response) throws IOException {

    limparBuffer(response);
    iniciarDownload(context, tela, response);
    apagarArquivosGerados(tela);
}
```

3.3.2.6 Classe `GeradorPaginaErro`

A classe `GeradorPaginaErro` é a classe responsável por gerar uma página contendo mensagens de erro para o usuário no caso de ter ocorrido alguma exceção no processo de geração de tela. O Quadro 17 mostra a implementação do principal método desta classe, o método `gerarPagina`. Este método é responsável por montar todas as partes que compõem a página.

Quadro 17 - Implementação do método `gerarPagina`

```
public void gerarPagina(PrintWriter out, String mensagem) {
    out.println("<html>");
    definirConteudoHead(out);
    definirConteudoBody(out, mensagem);
    out.println("</html>");
}
```

3.3.3 *Templates* do FreeMaker

O arquivo `TemplateActivity.ftl` é o *template* responsável pela representação da tela gerada no código Java. O Quadro 18 mostra a implementação deste *template*.

Quadro 18 - *Template* de tela Java

```

<#if nomePacote??>
package ${nomePacote};

</#if>
<#list listaComponentes as componente>
import ${componente.caminhoImport};
</#list>

public class ${nomeClasse} extends AppCompatActivity {

    <#list listaComponentes as componente>
    private ${componente.tipo} ${componente.nome};
    </#list>

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.${nomeArquivoXml});

        inicializarViews();
    }

    private void inicializarViews() {
        <#list listaComponentes as componente>
        this.${componente.nome} = (${componente.tipo})
        findViewById(R.id.${componente.nome});
        </#list>
    }
}

```

O arquivo `TemplateComponenteComFilhos.ftl` é o *template* responsável pela representação XML dos componentes Android selecionados e que possuem componentes filhos. O Quadro 19 mostra a implementação deste *template*.

Quadro 19 - *Template* de tela XML para componentes com filhos

```

<?xml version="1.0" encoding="utf-8"?>
<${tipoComponente}
    <#list listaAtributos as atributo>
    ${atributo.nome}="${atributo.valor}"<#if atributo?is_last></#if>
    </#list>

    <#list listaFilhos as filho>
    ${filho.tagXml}
    </#list>

    </${tipoComponente}>

```

O arquivo `TemplateComponenteSemFilhos.ftl` é o *template* responsável pela representação XML dos componentes Android selecionados que não possuem componentes filhos. O Quadro 20 mostra a implementação deste *template*.

Quadro 20 - *Template* de tela XML para componentes sem filhos

```

<${tipoComponente}
    <#list listaAtributos as atributo>
    ${atributo.nome}="${atributo.valor}"<#if atributo?is_last>/></#if>

```

```
</#list>
```

3.3.4 Validações JavaScript

O arquivo `ValidacaoComponentes.js` contém o código JavaScript responsável pelas validações dos campos obrigatórios que o usuário deve informar para gerar as telas corretamente. O Quadro 21 mostra a implementação da principal função deste arquivo, a função `isFormComponentesValido`. Esta função é chamada quando o usuário seleciona o botão Gerar e é responsável por efetuar as validações dos componentes obrigatórios que o usuário deve preencher.

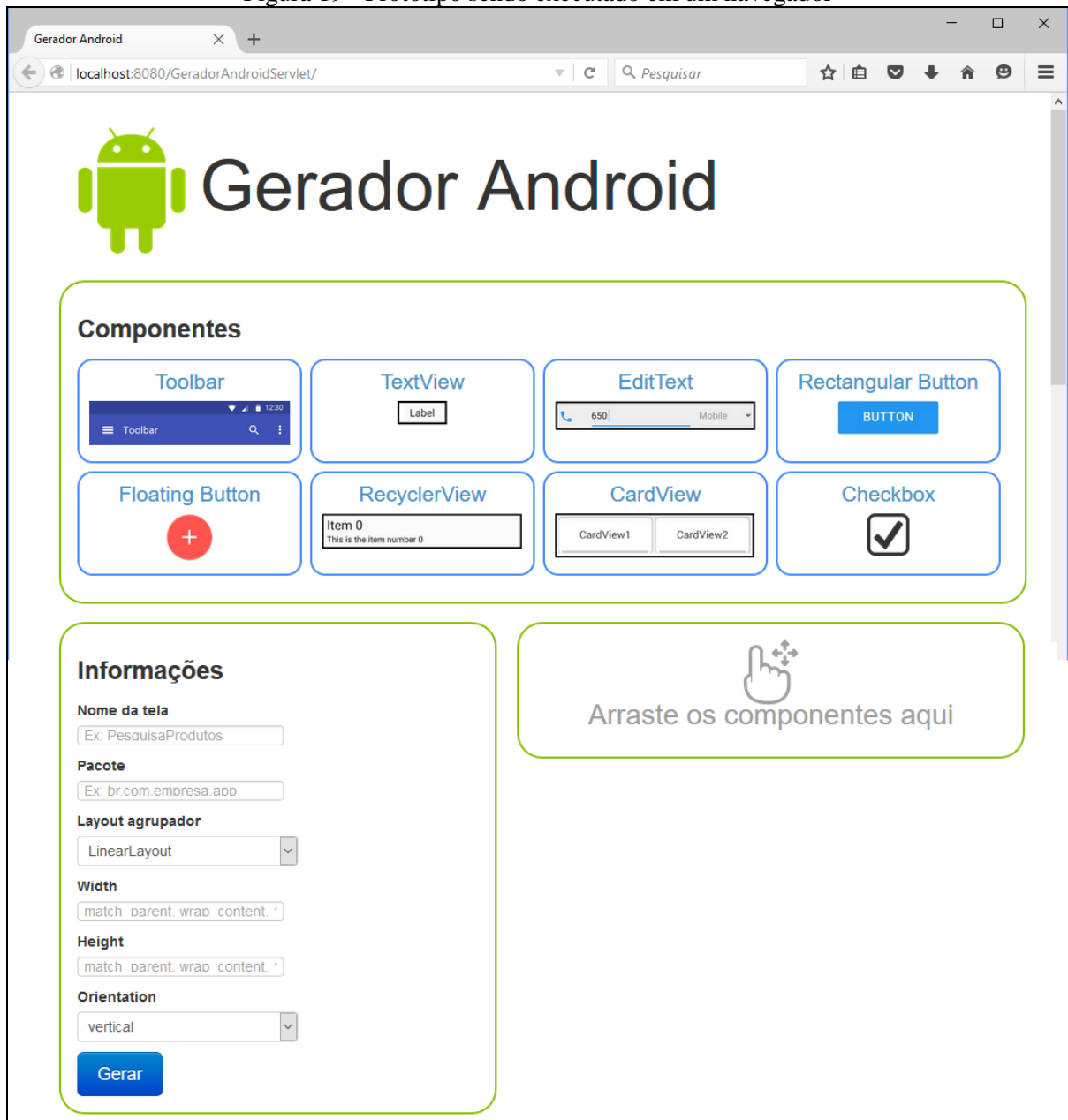
Quadro 21 - Implementação da função JavaScript responsável pelas validações de campos

```
function isFormComponentesValido() {  
    try {  
        validarNomeTela();  
        validarCamposComponentePrincipal();  
        validarSePossuiFilhosAdicionados();  
        validarCamposFilhos();  
  
        return true;  
    } catch (mensagemErro) {  
        mostrarMensagem(mensagemErro);  
        return false;  
    }  
}
```

3.3.5 Operacionalidade da implementação

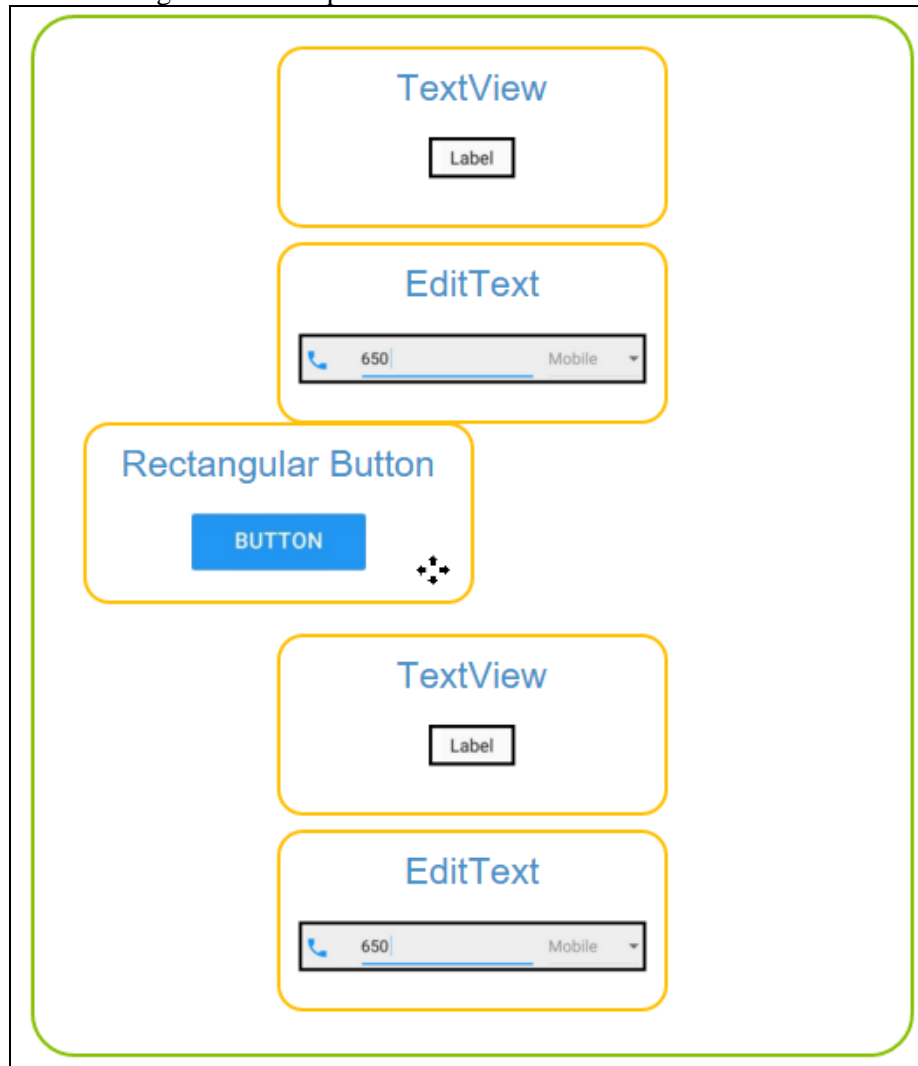
Esta seção busca apresentar a operacionalidade do protótipo por parte do usuário. A tela do protótipo possui uma paleta de componentes, um formulário para a inserção das informações principais da tela e um quadro que permite a inserção dos componentes a serem gerados. A Figura 19 mostra o protótipo sendo executado em um navegador.

Figura 19 - Protótipo sendo executado em um navegador



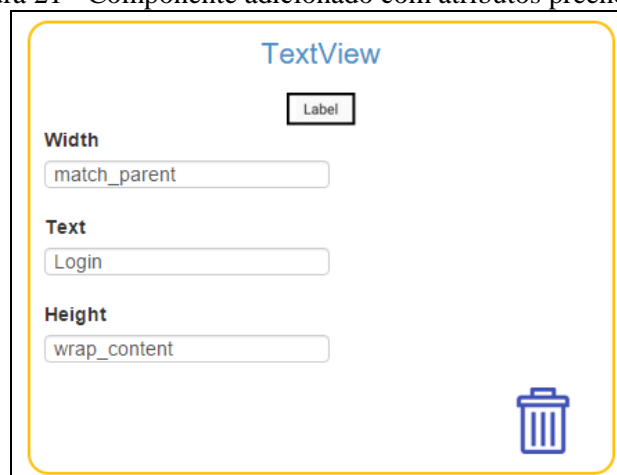
Ao adicionar os componentes na tela, é possível alterar a ordem em que eles serão gerados. A Figura 20 mostra alguns componentes adicionados sendo reordenados.

Figura 20 - Componentes adicionados sendo reordenados



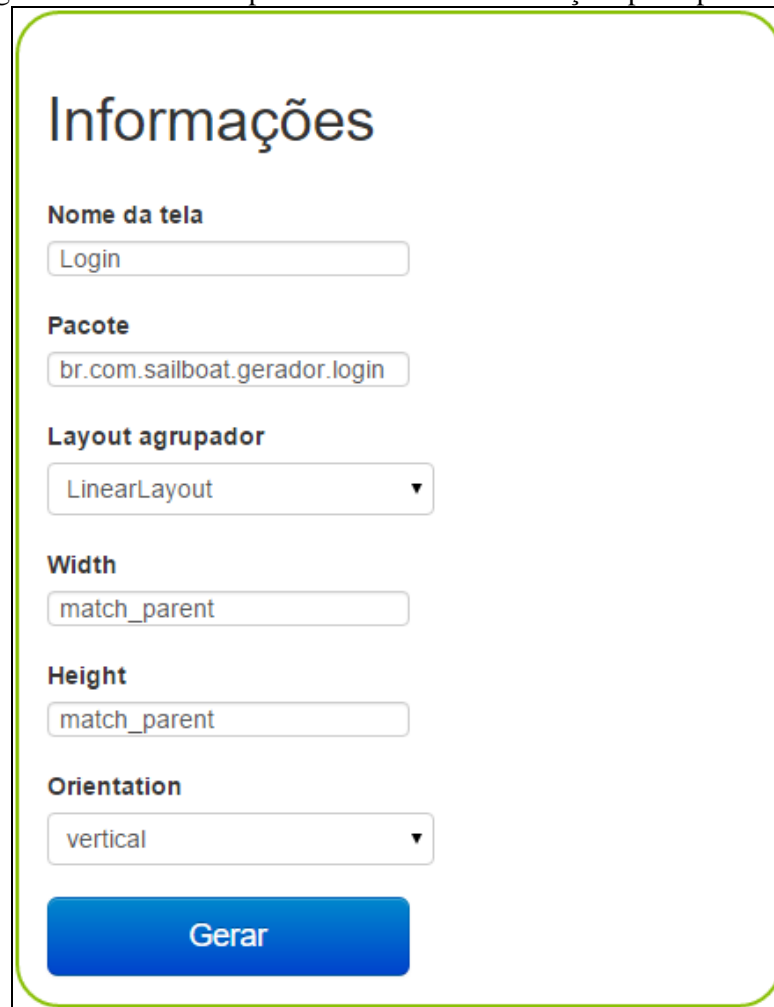
Depois de inserir os componentes a serem gerados, é necessário adicionar valores específicos referentes aos atributos de cada componente. A Figura 21 mostra um componente adicionado com seus atributos preenchidos.

Figura 21 - Componente adicionado com atributos preenchidos



Depois de adicionar os valores específicos de cada componente, é necessário preencher corretamente o formulário com as informações principais da tela a ser gerada. A Figura 22 exibe o formulário preenchido.

Figura 22 - Formulário preenchido com as informações principais da tela



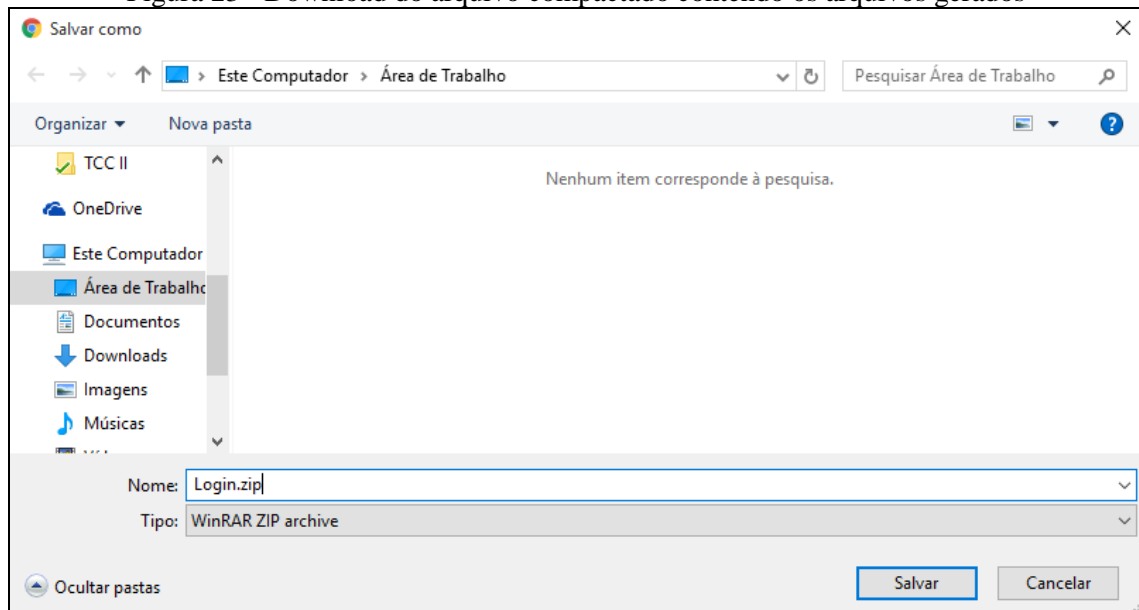
The image shows a web form titled "Informações" (Information) with the following fields and values:

- Nome da tela** (Screen Name): Login
- Pacote** (Package): br.com.sailboat.gerador.login
- Layout agrupador** (Grouping Layout): LinearLayout
- Width**: match_parent
- Height**: match_parent
- Orientation**: vertical

At the bottom of the form is a blue button labeled "Gerar" (Generate).

Depois de ter preenchido todas as informações necessárias, ao selecionar o botão Gerar, o sistema irá enviar as informações preenchidas para o servidor e irá retornar o *download* contendo os arquivos gerados. A Figura 23 mostra a tela que permite ao usuário fazer *download* do arquivo compactado contendo os arquivos gerados.

Figura 23 - Download do arquivo compactado contendo os arquivos gerados



No Quadro 22 é mostrado o conteúdo do arquivo XML de exemplo gerado a partir do protótipo.

Quadro 22 – Arquivo XML de exemplo gerado a partir do protótipo

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <TextView
    android:layout_width="match_parent"
    android:text="Login:"
    android:layout_height="wrap_content"
    android:id="@+id/textview1"/>

  <EditText
    android:layout_width="match_parent"
    android:hint="Insira um login"
    android:layout_height="wrap_content"
    android:id="@+id/edittext1"/>

  <TextView
    android:layout_width="match_parent"
    android:text="Senha:"
    android:layout_height="wrap_content"
    android:id="@+id/textview2"/>

  <EditText
    android:layout_width="match_parent"
    android:hint="Insira uma senha"
    android:layout_height="wrap_content"
    android:id="@+id/edittext2"/>

  <Button
    android:layout_width="match_parent"
    android:text="Entrar"
```

```

        android:layout_height="wrap_content"
        android:id="@+id/button1"/>
</LinearLayout>

```

No Quadro 23 é mostrado o conteúdo do arquivo Java de exemplo gerado a partir do protótipo.

Quadro 23 – Arquivo Java de exemplo gerado a partir do protótipo

```

package br.com.sailboat.gerador;

import android.widget.TextView;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.EditText;
import android.widget.Button;

public class ActivityLogin extends AppCompatActivity {

    private TextView textview1;
    private EditText edittext1;
    private TextView textview2;
    private EditText edittext2;
    private Button button1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activitylogin);

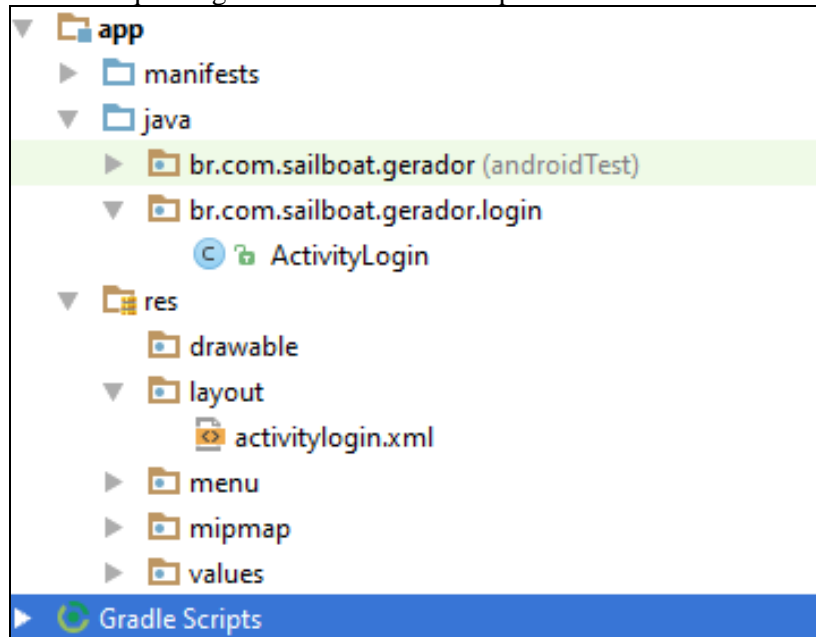
        inicializarViews();
    }

    private void inicializarViews() {
        this.textview1 = (TextView) findViewById(R.id.textview1);
        this.edittext1 = (EditText) findViewById(R.id.edittext1);
        this.textview2 = (TextView) findViewById(R.id.textview2);
        this.edittext2 = (EditText) findViewById(R.id.edittext2);
        this.button1 = (Button) findViewById(R.id.button1);
    }
}

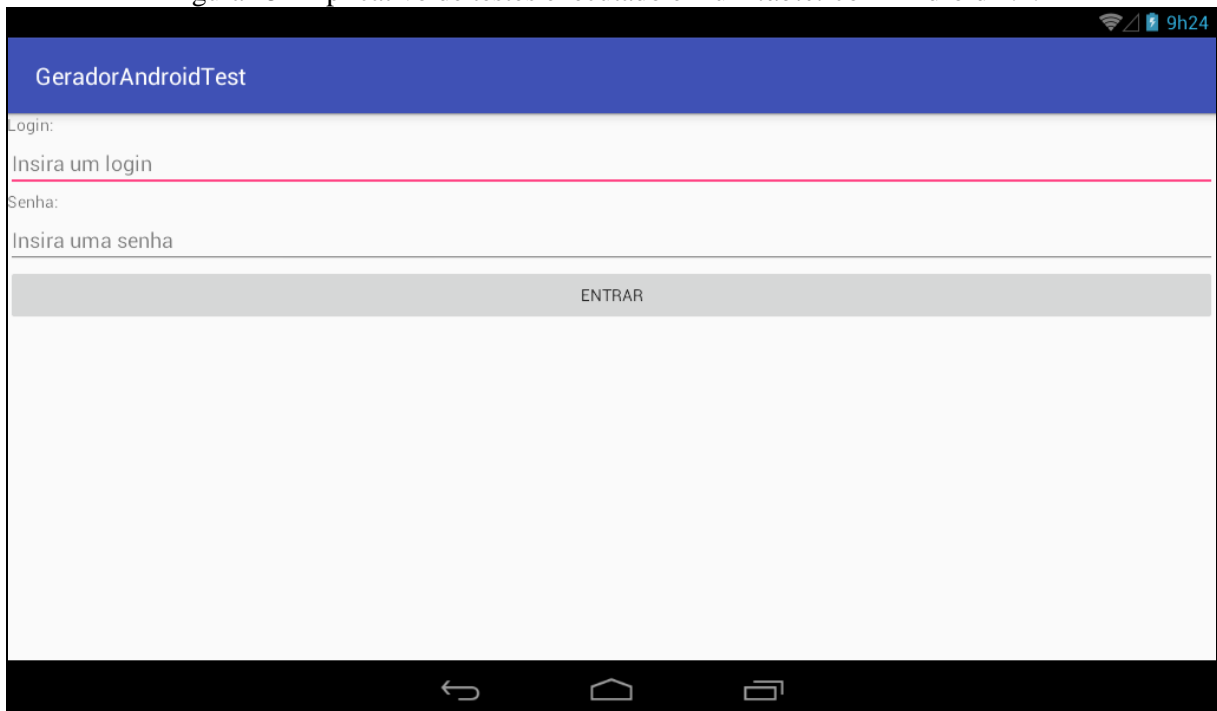
```

Depois de descompactar os arquivos, é possível copiá-los para dentro da IDE Android Studio. A Figura 24 mostra os arquivos gerados devidamente importados em um projeto de testes.

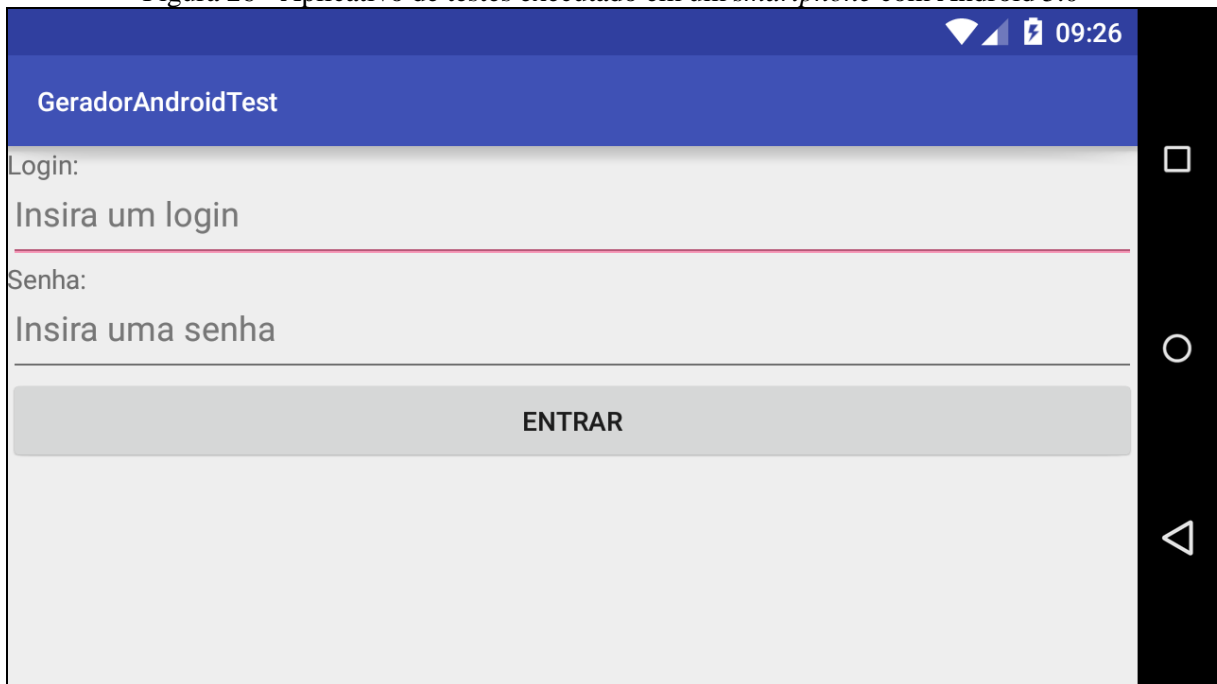
Figura 24 - Arquivos gerados devidamente importados na IDE Android Studio



A Figura 25 mostra um aplicativo de testes contendo os arquivos de tela gerados, sendo executados em um *tablet* com o sistema Android 4.1.1.

Figura 25 - Aplicativo de testes executado em um *tablet* com Android 4.1.1

A Figura 26 mostra um aplicativo de testes contendo os arquivos de tela gerados, sendo executados em um *smartphone* com o sistema Android 5.0.

Figura 26 - Aplicativo de testes executado em um *smartphone* com Android 5.0

3.4 RESULTADOS E DISCUSSÕES

O presente trabalho apresentou o desenvolvimento de um protótipo que gera arquivos de tela Android (XML e Java) a partir de *templates* do *framework* FreeMarker. Como este *framework* disponibiliza uma linguagem simples de criação de *templates*, é possível gerar arquivos de texto de forma escalável sem perder a legibilidade do código.

Os testes foram feitos importando as telas geradas na IDE Android Studio e executando o aplicativo de testes tanto em emuladores como em dispositivos reais. Para os dispositivos reais foi utilizado um *smartphone* com Android 5.0 e um *tablet* com Android 4.1.1. Nas duas versões do sistema as telas se comportaram de maneira estável. Os testes foram repetidos tanto no emulador nativo do Android, como no emulador Genymotion.

Os conceitos apresentados por Martin (2009) com relação à estruturação e organização de código, facilitaram não só os testes, como também a integração entre as classes e o entendimento da implementação do protótipo como um todo por parte de outros programadores.

Em relação aos trabalhos correlatos, Battisti (2014) utilizou as mesmas tecnologias para realizar a geração de código no seu projeto, porém Rezini (2013) foi quem mais se aproximou em termos de usabilidade, pois utilizou a função de arrastar e soltar os componentes na tela. O Quadro 24 apresenta as principais diferenças entre os trabalhos correlatos citados e o trabalho desenvolvido.

Quadro 24 - Comparativo entre ferramentas

CARACTERÍSTICA	TCC	Battisti	Rezini
Plataforma	Web	Web	Desktop
Linguagem de programação	Java	Java	Delphi
Formato de entrada	Formulário HTML	Padrões de requisitos	Formulário Delphi
Formato de saída	XML e Java	XML ou JSON	XML e Java
Realiza a construção de um projeto Android após a geração de código	Não	Não	Sim

4 CONCLUSÕES

Após o estudo realizado sobre geração de código para a plataforma Android, verificou-se que o tempo necessário para desenvolver uma tela manualmente justifica a criação de um gerador de código. Partindo do pressuposto de que as empresas de software têm muitos projetos e vários requisitos para implementar, um gerador de código direcionado ao contexto de uma empresa é capaz de proporcionar agilidade na implementação de projetos.

Este trabalho apresentou o projeto e desenvolvimento de um protótipo de geração de interfaces gráficas para a plataforma Android a partir de *templates* previamente implementados, onde o principal objetivo é agilizar o processo de desenvolvimento de telas para esta plataforma. O protótipo visa eliminar a implementação manual tanto das telas XML como da ligação entre os componentes através de sua referência correspondente no código Java.

Após o estudo e compreensão do *framework* FreeMarker, esse foi de grande utilidade no desenvolvimento do protótipo. A grande diversidade de materiais e componentes e a possibilidade de geração de arquivos de texto a partir de *templates* foram de grande facilitadora na geração das telas. Outro grande facilitador foi o *framework* JQuery que, através de ferramentas que possibilitam arrastar e soltar componentes HTML, possibilitou um aspecto dinâmico na utilização do protótipo ao adicionar os componentes Android na tela a ser gerada.

Tomando como base os trabalhos correlatos estudados, pode-se afirmar que o protótipo criado possui algumas similaridades com ambos os trabalhos. A principal contribuição e diferencial deste trabalho com relação aos demais trata-se da maneira dinâmica em que os componentes são adicionados à tela e seu potencial de escalabilidade por ser um projeto web que, por sua vez, não necessita de um processamento elevado no lado do cliente.

4.1 EXTENSÕES

Como sugestões de extensões para continuidade do presente trabalho em trabalhos têm-se:

- a) adicionar mais componentes Android na paleta de seleção;
- b) permitir combinar a orientação vertical e horizontal entre os componentes de uma mesma tela;
- c) utilizar o padrão JSON para enviar para o servidor os componentes a serem gerados;
- d) permitir adicionar componentes filhos dentro de outros componentes;
- e) permitir gerar um projeto Android completo contendo a tela gerada;

- f) permitir a persistência da tela gerada em um banco de dados, possibilitando ao usuário realizar a recuperação da tela dentro do protótipo em algum outro momento;

REFERÊNCIAS

- ANDROID DEVELOPERS. **The developer's guide**, [S. l.], 2014. Disponível em <<http://developer.android.com/guide/>>. Acesso em: 12 set. 2015.
- APPCCELERATOR INC. **Titanium**, 2015. Disponível em: <<http://www.appcelerator.com/>>. Acesso: 17 set. 2015.
- BATTISTI, Leandro V. **Elicitar**: protótipo de gerador de código a partir de especificações com padrões de requisitos. 2014. 55 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, Brasil.
- GOOGLE DESIGN. [S. l.], 2015. Disponível em <<https://www.google.com/design/spec/>>. Acesso em: 12 ago. 2015.
- INTERNATIONAL DATA CORPORATION. **Smartphone OS Market Share, Q2 2014**, [S.l.], 2014. Disponível em: <<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>>. Acesso em: 13 set. 2014.
- MARTIN, Robert C. **Código limpo**: habilidades práticas do Agile software. Rio de Janeiro: Alta Books, 2009. xv, 412 p, il.
- MEDNIEKS, Zigurd et al. **Programming Android**: second edition. Sebastopol: O'Reilly Media, 2011.
- MOBILE TIME. [S. l.], 2015. Disponível em <http://www.mobiletime.com.br/06/08/2015/fragmentacao-do-android-ha-24-mil-modelos-e-quase-1-3-mil-fabricantes-no-mundo/418325/news.aspx?__akacao=2518960/>. Acesso em: 10 ago. 2015.
- MONTEIRO, João. **Google Android – Crie aplicações para celulares e tablets**. São Paulo: Casa do Código, 2013. 317 p, il.
- REZINI, Douglas J. **Geração de interfaces Android a partir do Delphi**. 2013. 70 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, Brasil.
- SILVEIRA, Janira. **Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados**. 2006. 82 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, Brasil.
- SMARTFACE INC. **Smartface**, 2015. Disponível em: <<http://www.smartface.io/>>. Acesso: 14 set. 2015.
- WEB SOCIAL DEV. [S. l.], 2014. Disponível em <[http:// http://websocialdev.com/lista-de-frameworks-para-desenvolvimento-mobile/](http://http://websocialdev.com/lista-de-frameworks-para-desenvolvimento-mobile/)>. Acesso em: 14 set. 2014.
- XAMARIN INC. **Xamarin**, 2015. Disponível em: <<http://xamarin.com/>>. Acesso: 21 set. 2015.