UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

GERADOR DE INTERFACES GRÁFICAS PARA IOS

GABRIEL SEBASTIAN RAMIREZ

BLUMENAU
2013

2013/2-05

GABRIEL SEBASTIAN RAMIREZ

GERADOR DE INTERFACES GRÁFICAS PARA IOS

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

Profa. Joyce Martins, Mestre - Orientadora

BLUMENAU
2013

2013/2-05

GERADOR DE INTERFACES GRÁFICAS PARA IOS

Por

GABRIEL SEBASTIAN RAMIREZ

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

Presidente:	Profa. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB

Blumenau, 09 de dezembro de 2013

Dedico este trabalho aos meus pais, à minha namorada, à minha família e aos meus amigos, especialmente àqueles que me incentivaram durante a realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

Aos meus pais, que sempre incentivaram minha educação.

Aos meus amigos, por compartilhar suas experiências em outros trabalhos.

À minha orientadora, Joyce Martins, por ter acreditado desde o início na conclusão deste trabalho.

Retroceder sim. Render-se jamais.

Ernesto Che Guevara

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta para o iPad que permite a prototipação de interfaces gráficas para a plataforma iOS (iPhone e iPad). Suporta os principais elementos gráficos e respectivas propriedades que o iOS disponibiliza para criar as interfaces. Foi desenvolvida no *framework* Titanium SDK e permite gerar código XML e JavaScript correspondente às interfaces gráficas criadas.

Palavras-chave: iOS. Prototipação. Interfaces gráficas. Gerador de código. Titanium SDK.

ABSTRACT

This work presents the development of a tool for the iPad that allows prototyping GUIs for the iOS platforms (iPhone and iPad). Supports major graphic elements and their properties provides for the iOS to create interfaces. It was developed in the framework Titanium SDK and allows generate the XML and Javascript code corresponding to the graphical interfaces created.

Keywords: iOS. Prototyping. GUIs. Code generator. Titanium SDK.

LISTA DE ILUSTRAÇÕES

Figura 1 - Camadas do iOS 15
Quadro 1 – Comparativos entre os dispositivos iOS 16
Figura 2 – Camadas do ambiente de desenvolvimento 18
Figura 3 – Fluxo do desenvolvimento no Titanium Studio 19
Quadro 2 – Elementos gráficos suportados 20
Figura 4 – Aplicativo do KhanApp, usando o jQuery Mobile 22
Figura 5 - Ambiente do Titanium Studio utilizando o ForgedUI 23
Quadro 3 – Elementos gráficos suportados 25
Quadro 3 – Elementos gráficos suportados (continuação) 26
Quadro 4 – Propriedades gráficas suportadas 26
Figura 6 – Diagrama de casos de uso 27
Quadro 5 – Caso de uso: `Selecionar dispositivo` 28
Quadro 6 – Caso de uso: `Incluir elemento gráfico na interface` 28
Quadro 7 – Caso de uso: `Editar as propriedades dos elementos gráficos` 29
Quadro 8 – Caso de uso: `Visualizar o XML da interface` 29
Quadro 9 – Caso de uso: `Exportar o XML da interface` 30
Quadro 10 – Caso de uso: `Visualizar o código JavaScript da interface` 30
Quadro 11 – Caso de uso: `Exportar o código JavaScript da interface` 31
Quadro 12 – Caso de uso: `Salvar a interface` 31
Quadro 13 – Caso de uso: `Carregar interfaces` 32
Figura 7 – Diagrama de classes do pacote `App.Controller` 32
Figura 8 – Diagrama de classes do pacote `App.Model` 34
Figura 9 – Diagrama de classes do pacote `App.View` 35
Figura 10 – Diagrama de sequência: `Visualizar o código JavaScritp da interface` 38
Figura 11 – Diagrama de sequência: `Carregar interface` 39
Figura 12 – MER da ferramenta 39
Quadro 14 – Arquivo `tiapp.xml` 41
Figura 13 – Estrutura de diretórios do projeto 42
Quadro 15 – Método `criarElemento` 42
Quadro 16 – Método `criarPropriedades` 43
Quadro 17 – Método `mostrarCodigo` 44

Quadro 18 – Trecho do método `gerarElemento` 45
Figura 14 – Tela principal da ferramenta 46
Figura 15 – Dispositivos 47
Figura 16 – Elementos suportados 47
Figura 17 – Propriedades dos elementos 48
Figura 18 – Visualizar e exportar interfaces 48
Figura 19 – Salvar e carregar interfaces 48
Figura 20 – Interface padrão iPad 49
Figura 21 – Interface no iPhone 4 50
Figura 22 – Interface no iPhone 5 51
Figura 23 – Interface no iPad 52
Quadro 19 – Comparativo de funcionalidades entre os trabalhos correlatos 53
Quadro 20 – Código JavaScript resultante da interface no iPhone 4 58
Quadro 21 – Código XML resultante da interface no iPhone 4 60
Quadro 22 – Código JavaScript resultante da interface no iPhone 5 61
Quadro 23 – Código JavaScript resultante da interface no iPad 63

LISTA DE SIGLAS

API – *Application Programming Interface*

CSS – *Cascading Style Sheets*

GPS - *Global Positioning System*

GUI - *Graphical User Interface*

HIG - *Human Interface Guidelines*

HTML – *Hypertext Markup Language*

IDE – *Integrated Development Environment*

iOS – *iPhone Operating System*

JSON – *JavaScript Object Notation*

MER – Modelos de Entidades e Relacionamentos

MVC – *Model-View-Controller*

PHP – *Hypertext Preprocessor*

RCP - *Rich Client Platafrom*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SDK - *Software Development Kit*

TSS - *Titanium Style Sheets*

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 PLATAFORMA IOS 15
2.1.1 Dispositivos iOS 16
2.2 HIG 16
2.3 AMBIENTE TITANIUM STUDIO 17
2.4 FRAMEWORK ALLOY 19
2.5 TRABALHOS CORRELATOS 20
2.5.1 Interface 2 20
2.5.2 jQuery Mobile 21
2.5.3 ForgedUI 22
3 DESENVOLVIMENTO 24
3.1 REQUISITOS PRINCIPAIS DA FERRAMENTA 24
3.2 ESPECIFICAÇÃO 24
3.2.1 Elementos gráficos suportados 25
3.2.2 Propriedades gráficas suportadas 26
3.2.3 Casos de uso 27
3.2.4 Classes da ferramenta 32
3.2.5 Diagramas de sequência 36
3.2.6 MER 39
3.3 IMPLEMENTAÇÃO 40
3.3.1 Técnicas e ferramentas utilizadas 40
3.3.2 Arquivo `tiapp.xml` 40
3.3.3 Organização dos arquivos no projeto 41
3.3.4 Principais métodos implementados 42
3.3.5 Operacionalidade da implementação 45
3.4 RESULTADOS E DISCUSSÃO 49
3.4.1 Comparativo dos trabalhos correlatos 52
4 CONCLUSÕES 54

4.1 EXTENSÕES 54
REFERÊNCIAS BIBLIOGRÁFICAS 56
APÊNDICE A – Código JavaScript resultante da interface no iPhone 4 58
APÊNDICE B – Código XML resultante da interface no iPhone 4..... 60
APÊNDICE C – Código JavaScript resultante da interface no iPhone 5 61
APÊNDICE D – Código JavaScript resultante da interface no iPad 63

1 INTRODUÇÃO

Quando os primeiros dispositivos denominados *tablets* chegaram ao mercado, estes apontavam para um incremento no uso da computação móvel, mas não obtiveram muito sucesso comercial. Existiam motivos de sobra para isso, desde as limitações de recursos de processamento e quantidade de memória até a imprecisão da entrada de dados para o dispositivo. O cenário mudou consideravelmente em 2010, ano em que a Apple anunciou oficialmente o lançamento do seu modelo de *tablet*: o iPad (GOLDSTEIN; BOVE, 2011, p. 1).

A partir de então, os dispositivos móveis tornaram-se populares nas mais diversas áreas devido à simplicidade, à funcionalidade, à portabilidade e à facilidade de utilização (MYERS et at., 2004, p. 36). O mercado destes aparelhos proporcionou uma revolução quando introduziu o *smartphone*, no sentido de que alterou o paradigma da mobilidade. Paes e Moreira (2007, p. 50) afirmam que não só era possível transportar informação relevante para o dia a dia, como também passava a ser possível enviá-la e recebê-la em tempo real, bem como fazer chamadas telefônicas tradicionais.

O mercado atual de *tablets* apresenta números expressivos. Segundo pesquisa da International Data Corporation Brasil (2012), foram vendidos mais de 769 mil *tablets* somente no terceiro trimestre de 2012 no Brasil e a previsão era alcançar cerca de 2,9 milhões de aparelhos até o final deste ano. Isso significa um aumento de mais de 200% comparados ao ano anterior de 2011, onde foram vendidas 800 mil unidades. Já a expectativa para o ano de 2013 é de alcançar 4 milhões de unidades. Segundo esta mesma pesquisa, o mercado de *tablets* passará o de computadores convencionais nos próximos três anos.

No entanto, o desenvolvimento de aplicativos para estas plataformas, tanto para *tablets* como para *smartphones*, requer do desenvolvedor conhecimento e experiência. Considerando a plataforma do *iPhone Operating System* (iOS), este problema agrava-se ainda mais, pois a linguagem geralmente utilizada é o Objective C, que demanda um certo investimento na aprendizagem. Existem, porém, ambientes de desenvolvimento e linguagens de programação alternativas ao Objective C. Um deles é o Titanium Studio (APPCELERATOR, 2013b), que permite desenvolver aplicativos utilizando JavaScript, uma linguagem mais prática e de fácil aprendizagem. O Titanium Studio processa o código JavaScript e gera o aplicativo nativo, tanto para o iOS quanto para o Android. No entanto, é um ambiente *desktop*.

Assim, diante do exposto, a partir dos recursos disponibilizados pelo Titanium Studio, desenvolveu-se uma ferramenta para o iPad que permita a prototipação da interface gráfica de aplicativos móveis iOS utilizando o próprio dispositivo móvel no desenvolvimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta para gerar protótipos de interfaces gráficas de aplicativos móveis na plataforma iOS.

Os objetivos específicos podem ser descritos da seguinte maneira:

a) disponibilizar uma ferramenta para iPad, na qual seja possível criar interfaces gráficas tanto para o iPhone como para o iPad, a partir de determinados elementos e propriedades;
b) gerar *eXtensible Markup Language* (XML) e código JavaScript correspondente às interfaces criadas;
c) exportar XML e código JavaScript para o ambiente do Titanium Studio.

1.2 ESTRUTURA DO TRABALHO

O texto está estruturado em quatro capítulos. O próximo capítulo contém a fundamentação teórica necessária para permitir um melhor entendimento sobre o trabalho desenvolvido.

O capítulo três apresenta o desenvolvimento da ferramenta, contemplando os principais requisitos do problema e a especificação, com os casos de uso, os diagramas de classes e os diagramas de sequência. Neste mesmo capítulo são apresentadas as ferramentas utilizadas na especificação e na implementação, além da operacionalidade da ferramenta desenvolvida e os resultados obtidos.

O capítulo quatro refere-se a conclusões do presente trabalho e a sugestões para trabalhos futuros.

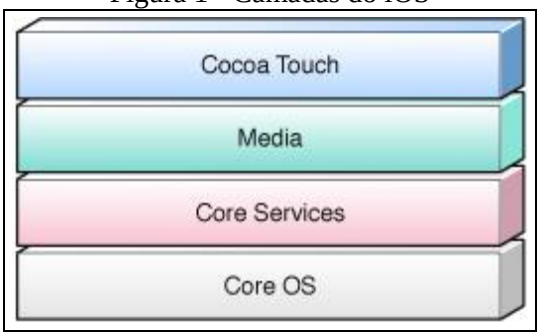
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em cinco seções, sendo que a seção 2.1 trata da plataforma iOS, a seção 2.2 explica brevemente o *Human Interface Guidelines* (HIG), a seção 2.3 introduz o ambiente de desenvolvimento Titanium Studio, enquanto a seção 2.4 apresenta o *framework* Alloy, e, por fim, na seção 2.5 são apresentados três trabalhos que abordam assuntos relacionados ao trabalho proposto.

2.1 PLATAFORMA IOS

O iOS é o sistema operacional utilizado pela Apple nos seus dispositivos móveis. Desde o seu lançamento, junto do primeiro iPhone, o sistema foi um sucesso entre o seus usuários e ainda continua como um dos principais ditadores de tendências no mercado (RIBEIRO, 2011). A Figura 1 mostra uma visão geral da arquitetura em camadas do iOS.

Figura 1 - Camadas do iOS



Fonte: Apple (2012a).

A camada *Core OS* é a camada mais baixa do sistema operacional do iPhone. Ela contém serviços para gerenciamento de energia, segurança e *Bonjour*, que é um sistema sem configuração de descoberta de dispositivos e serviços em redes IP (APPLE, 2012a).

A camada *Core Services* fornece os serviços de sistema fundamentais que todos os aplicativos fazem uso. Um deles é o *Core Foundation*, que inclui uma interface baseada em C para coleções compostas de listas e dicionários, bem como `strings` e `strings` mutáveis. Segundo Apple (2012a), a camada *Core Services* também contém *frameworks*, incluindo o *CoreData* para gerenciar os modelos de dados em um aplicativo e o *CoreLocation* para gerenciar os dados de localização disponíveis a partir de *Global Positioning System* (GPS), triangulação e WiFi. Ele possui também a interface de baixo nível para o SQLite e para o *parser* do XML (APPLE, 2012a).

A camada *Media* contém gráficos, áudio, vídeo e tecnologias, cujo objetivo é fornecer uma interface de alto nível para configurar animações e efeitos (APPLE, 2012a).

A camada *Cocoa Touch* contém os principais *frameworks* para a construção de aplicativos iOS. É dividida em duas sub-camadas. O nível mais baixo consiste das partes que não são interface do usuário. Este é o *Foundation Framework* que contém alguns serviços do sistema incluindo acesso ao sistema de arquivos e *Application Programming Interface* (API) de rede. O nível superior é o *framework* UIKit, que contém toda a infraestrutura de aplicativos e todos os componentes gráficos. Conforme afirma Apple (2012a), o *framework* UIKit inclui manipulação de eventos, gráficos, janelas, texto e gerenciamento de web, além de permitir ao desenvolvedor acesso a algumas das interfaces de hardware, incluindo a câmera, o acelerômetro e outros sensores do iPhone.

2.1.1 Dispositivos iOS

Existem atualmente diversos modelos de iPhones e iPad cada um com a suas caracteristicas. No Quadro 1 podem ser visualizados os principais dispositivos da atualidade.

Quadro 1 – Comparativos entre os dispositivos iOS

modelo	dimensões	tamanho da tela	resolução
iPhone 3G/3GS	115,2 x 62,1 mm x 12,3 mm	3,5 polegadas	320 x 480 pixeis em 163 ppi
iPhone 4/4S	115,2 mm x 58,6 mm x 9,3 mm	3,5 polegadas	640 x 960 pixeis em 326 ppi
iPhone 5/5S	123,8 mm x 58,6 mm x 7,6 mm	4 polegadas	640 x 1136 pixeis em 326 ppi
iPad 2	241,2 mm x 185,7 mm x 8,8 mm	9,7 polegadas	768 x 1024 pixeis em 132 ppi
iPad 3/4	241,2 mm x 185,7 mm x 8,8 mm	9.7 polegadas	1536 x 2048 pixeis em 264 ppi

2.2 HIG

Segundo Charland e LeRoux (2011), cada plataforma tem suas próprias convenções para as interfaces de usuário, geralmente descritas em um documento conhecido como HIG e evidenciadas na interface do sistema operacional correspondente. O HIG do iOS descreve diretrizes e princípios que ajudam os desenvolvedores na elaboração de interfaces de usuário (APPLE, 2012b).

As principais recomendações do HIG são (APPLE, 2012b):

a) utilizar componentes de interface com um tamanho mínimo de 44x44 pontos para não dificultar a manipulação dos mesmos;
b) atentar para a rotação do dispositivo, já que as pessoas esperam poder rotacionar o aparelho a qualquer momento;
c) desenvolver o aplicativo da maneira mais intuitiva possível;
d) manter a consistência no aplicativo para que seja previsível o comportamento do mesmo;
e) procurar manipular os objetos diretamente ao invés de usar componentes separados;
f) considerar a customização do aplicativo no início do processo, entre outras recomendações.

Além disso, existem alguns componentes de interface que são específicos no iOS, como o `Picker` que apresenta um conjunto de valores entre os quais o usuário pode escolher um; o `Page View Controller` que gerencia o conteúdo de várias páginas usando rolagem para transição entre elas; o `Split View`, exclusivo do iPad, que divide a exibição da tela em dois painéis lado a lado; e o `Popover`, também exclusivo do iPad, que permite a exibição de certo conteúdo numa visão transitória; entre outros (APPLE, 2012b).

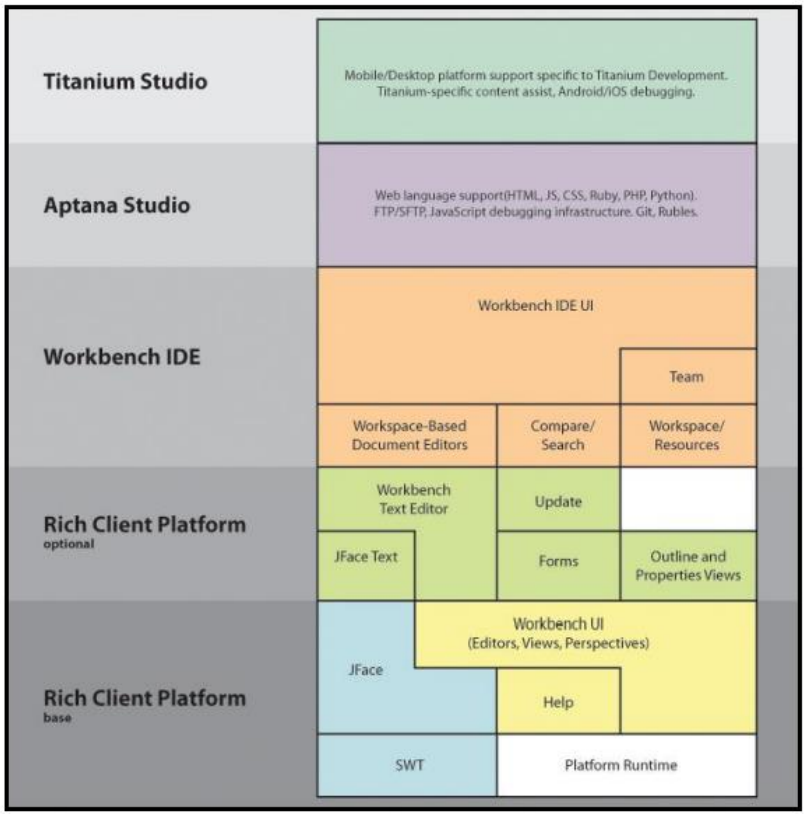
2.3 AMBIENTE TITANIUM STUDIO

O Titanium Studio, segundo AppCelerator (2013b), é uma plataforma *desktop* para desenvolvimento web e *mobile*. Com mais de 50 mil aplicativos instalados em 125 milhões de aparelhos, o Titanium Studio utiliza mais de 5000 APIs para criar aplicativos nativos tanto para iOS quanto para o Android (APPCELERATOR, 2013b). Taft (2011) afirma que é possível testar aplicativos móveis no simulador ou no dispositivo e ainda publicá-los na App Store ou no Google Play. O *Software Development Kit* (SDK) está disponível sob a licença Apache 2 (APPCELERATOR, 2012a). Isto significa que é uma licença perpétua, mundial, não exclusiva e gratuita. É uma licença de direitos autorais irrevogável para produzir ou reproduzir trabalhos derivados, exibir ou executar publicamente, sublicenciar e distribuir o trabalho em código fonte ou objeto de formulário (APACHE SOFTWARE FOUNDATION, 2004).

Para desenvolver aplicativos para a plataforma iOS no Titanium Studio é preciso ter uma máquina Apple (iMac ou Macbook) com o Xcode instalado, já que o ambiente precisa do SDK do iOS. As camadas desse ambiente podem ser visualizadas na Figura 2.

O ambiente do Titanium Studio consiste em uma camada de *plug-ins* e empacotadores que, juntamente com o Aptana Studio 3.2.2, trabalha sobre o *core* do Eclipse versão 3.7.2 (ANDERSON, 2013, p. 1). O Titanium Studio dá suporte ao desenvolvimento de aplicativos móveis específicos do *framework* Titanium. Também permite a depuração dos aplicativos desenvolvidos para as plataformas iOS e Android. O Aptana Studio dá suporte ao desenvolvimento web para aplicações clientes, utilizando *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript, e para aplicações servidoras, utilizando programação Ruby, PHP e Phyton. A Workbench IDE representa toda a infra-estrutura que o Eclipse oferece para o Titanium Studio. O *Rich Client Platafrom* (RCP) consiste de uma série de componentes já prontos e testados, como manipuladores de texto, editores de texto, *buffers* de arquivo, gerenciador de atualizações, entre outros, que o *framework* disponibiliza com a finalidade de facilitar a integração e o desenvolvimento de aplicações.

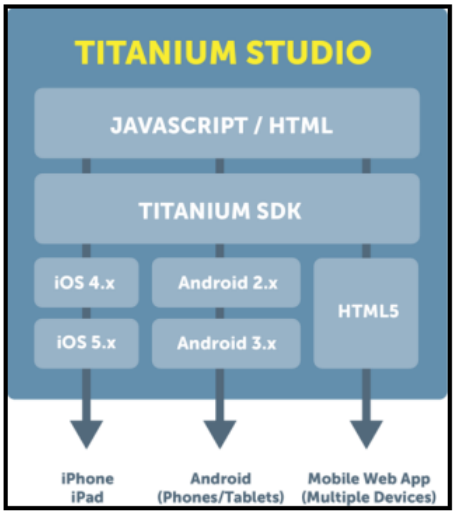
Figura 2 – Camadas do ambiente de desenvolvimento



Fonte: AppCelerator (2012a).

A Figura 3 apresenta fluxo do desenvolvimento utilizando o Titanium Studio. A implementação pode ser realizada em JavaScript ou HTML, a partir da qual é gerado código objeto para as plataformas iOS e Android, além de aplicações web em HTML5 para diversos dispositivos móveis.

Figura 3 – Fluxo do desenvolvimento no Titanium Studio



Fonte: AppCelerator (2012b).

No Titanium Studio não existe até o momento uma ferramenta para o desenvolvimento de *Graphical User Interface* (GUI), ou seja, todas as interfaces dos aplicativos são programadas. Existem alguns *plug-ins* de terceiros que permitem contornar esta situação, porém são pagos e possuem certas limitações (ANDERSON, 2013, p. 11).

2.4 FRAMEWORK ALLOY

Alloy é um *framework* da AppCelerator desenvolvido para o Titanium Studio com o objetivo de facilitar o desenvolvimento de aplicativos móveis. Utiliza a arquitetura *Model View Controller* (MVC), a partir de XML e CSS, para separar a interface do usuário da lógica de negócio e do modelo de dados. Com o Alloy é possível diminuir significativamente a quantidade de tempo e de código implementado, tornando o código mais fácil de ler, gerenciar e reutilizar (APPCELERATOR, 2013a).

As três camadas do Alloy são:

a) modelo: implementa a lógica de negócio e é desenvolvida a partir dos arquivos JavaScript;
b) visão: é codificada através de uma combinação de XML e de um estilo conhecido como *Titanium Style Sheets* (TSS);
c) controle: é responsável por emparelhar um arquivo JavaScript com um componente XML/TSS para responder à iteração do usuário, manipular eventos e interagir com a camada do modelo para executar a lógica de negócio e atualizar a visão do componente.

O Quadro 2 contém alguns elementos gráficos que são suportados, incluindo o código JavaScript e o XML correspondentes no formato do Alloy.

Quadro 2 – Elementos gráficos suportados

elemento	código Javascript	XML Alloy
botão	`button = Titanium.UI.createButton ({title: 'Botão', width: 100, height: 50 })`	`<Button title="Botão" width="100" height="50" />`
rótulo	`label = Ti.UI.createLabel ({text: 'Rótulo', width: 100, height: 50 })`	`<Label width="100" height="50"> Rótulo </Label>`
campo	`textField = Ti.UI.createTextField ({width: 250, height: 60 })`	`<TextField width="250" height="60" />`

2.5 TRABALHOS CORRELATOS

É possível encontrar alguns trabalhos relacionados ao tema proposto. Foram estudados o Interface 2 (LESS CODE LIMITED, 2013), que é um aplicativo para o iOS onde é possível fazer protótipos de interfaces gráficas e gerar o código Objective C diretamente no dispositivo móvel; o jQuery Mobile (TAFT, 2012), que é uma ferramenta com a qual é possível criar protótipos de interfaces para aplicativos móveis; e o ForgedUI (STOWELL, 2011), que é uma ferramenta para gerar interfaces gráficas para iOS e Android no ambiente do Titanium Studio.

2.5.1 Interface 2

Less Code Limited (2013) descreve o Interface 2 como um aplicativo pago para iOS que permite criar protótipos de interfaces gráficas usando *widgets* nativos do iOS seguindo as rígidas exigências da Apple para ajudar o usuário a construir aplicativos de qualidade. Nele é possível ligar as telas entre si com animações e visualizar o fluxo do aplicativo como um todo. O Interface 2 permite também transformar todas as telas criadas em código nativo para o iOS, facilitando o desenvolvimento no Xcode.

O Interface 2 foi lançado no mercado no primeiro semestre de 2012 e atualmente se encontra na versão 2.0.6. Ele é compatível tanto com o iPhone quanto com o iPad e precisa de pelo menos a versão 4.3 do iOS. Atualmente tem suporte para gerar interfaces somente até a versão 5.0 do iOS e não existe previsão de suporte aos novos recursos do iOS 6.0 (LESS CODE LIMITED, 2013).

Less Code Limited (2013) afirma ainda que na versão para o iPad é possível criar as telas tanto para o iPhone quanto para o iPad. Já no iPhone somente é permitido criar telas para

este mesmo dispositivo já que a tela do iPhone não é grande o suficiente para exibir certos componentes gráficos do iPad. O Interface 2 tem suporte a vários componentes gráficos, como por exemplo `TableView`, `Container`, `Image`, `Button`, `Text`, `Picker`, entre outros.

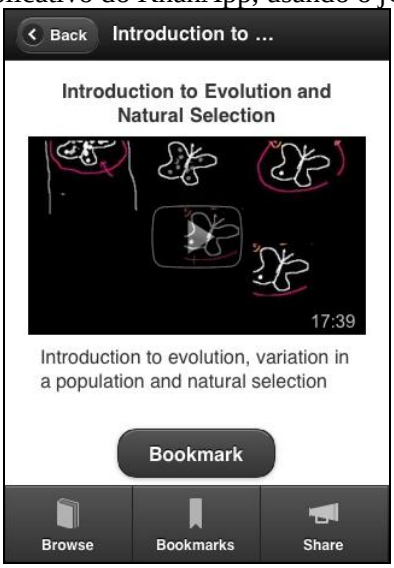
2.5.2 jQuery Mobile

Quando um programador começa a olhar para o desenvolvimento de aplicativos móveis, uma das primeiras dúvidas é se ele deve utilizar a tecnologia nativa da plataforma ou a web. Apesar do Objective-C ser fundamental para a construção nativa de aplicativos para o iOS, do Java para o Android e o BlackBerry, e do C# e do Visual Studio Tools para o Windows Phone, a oportunidade de construir aplicativos móveis com a tecnologia padrão da web está fazendo muitos desenvolvedores pensar na web como uma primeira opção (TAFT, 2012).

Seguindo esta linha, tem-se o jQuery Mobile, que é um sistema de interface unificada de usuário que funciona em todas as plataformas populares de dispositivos móveis. A diferença fundamental desta tecnologia é a grande variedade de plataformas móveis suportadas. Para desenvolvedores mais avançados, há uma API de opções de configuração global, eventos e métodos para aplicar o *script*, gerar páginas dinâmicas e construir aplicativos nativos com ferramentas como o PhoneGap. Todas as páginas em jQuery Mobile são construídas sobre uma base de HTML semântico para garantir a compatibilidade com praticamente qualquer dispositivo habilitado para web. Em dispositivos que interpretam CSS e JavaScript, jQuery Mobile aplica técnicas de aprimoramento para discretamente transformar a página semântica em uma experiência rica e interativa, que aproveita o poder do jQuery e CSS (TAFT, 2010).

Em vez de escrever aplicativos exclusivos para cada dispositivo móvel ou sistema operacional, o jQuery Mobile permite a criação de uma aplicação web única e altamente personalizada que funciona em todas as plataformas populares de *smartphones* e *tablets* (TAFT, 2010). A Figura 4 mostra um aplicativo desenvolvido usando o jQuery Mobile, chamado KhanApp, que permite aos usuários aperfeiçoarem-se em diversas áreas do conhecimento.

Figura 4 – Aplicativo do KhanApp, usando o jQuery Mobile



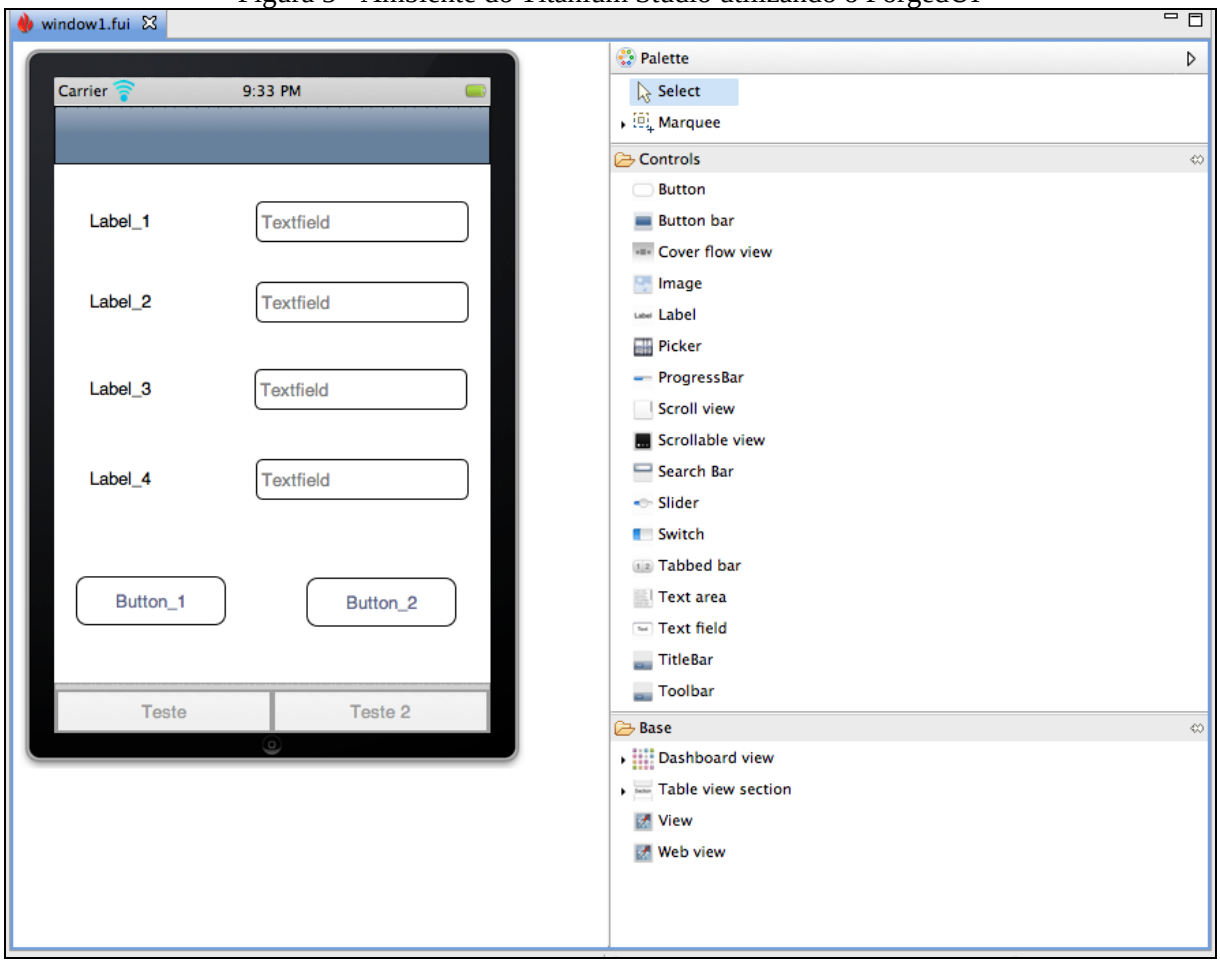
Fonte: KhanApp (2010).

2.5.3 ForgedUI

ForgedUI é, segundo Stowell (2011), um *plug-in* pago para o Titanium Studio baseado no Eclipse que simplifica as tarefas básicas de criar a parte visual de aplicativos para plataformas móveis. Foi lançado no final do ano de 2011 e atualmente está na versão 2.0. A ferramenta permite aos desenvolvedores do Titanium Studio desenhar e gerar código de interfaces gráficas tanto para a plataforma iOS como para a do Android.

Stowell (2011) afirma também que muitos desenvolvedores optaram pela ferramenta Titanium Studio para criar aplicações móveis tendo em vista a baixa curva de aprendizagem necessária e a facilidade de que uma única base de código pode ser usada para gerenciar e manter aplicativos em múltiplas plataformas. No entanto, o *design* de interfaces gráficas continua sendo uma tarefa complexa. A intenção do ForgedUI é livrar o desenvolvedor deste obstáculo e agilizar o processo de desenvolvimento de aplicativos móveis. Com o ForgedUI é possível arrastar e soltar componentes de interfaces gráficas sem a complexidade de escrever o código JavaScript necessário, já que o código correspondente é gerado pela ferramenta. Com isso o desenvolvedor ganha mais tempo para focar na funcionalidade do aplicativo e minimizar o tempo gasto ajustando *pixeis* (STOWELL, 2011). A Figura 5 apresenta um aplicativo sendo desenvolvido no Titanium Studio com ajuda do ForgedUI.

Figura 5 - Ambiente do Titanium Studio utilizando o ForgedUI



3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas de desenvolvimento da ferramenta. A primeira seção apresenta os principais requisitos do problema trabalhado. A segunda seção descreve os elementos gráficos suportados, bem como a especificação da solução através de diagramas da *Unified Modeling Language* (UML) e do Modelo de Entidades e Relacionamento (MER). A terceira seção apresenta a implementação da ferramenta. Por fim, na quarta seção são abordados os resultados deste trabalho.

3.1 REQUISITOS PRINCIPAIS DA FERRAMENTA

Os Requisitos Funcionais (RF) da ferramenta são:

a) RF01: permitir selecionar para qual dispositivo a interface será criada;
b) RF02: exibir os elementos gráficos suportados;
c) RF03: permitir escolher os elementos gráficos desejados;
d) RF04: permitir informar as propriedades de cada elemento selecionado;
e) RF05: salvar localmente as interfaces gráficas criadas;
f) RF06: gerar XML e código JavaScript correspondente à interface gráfica criada;
g) RF07: exportar XML e código JavaScript gerados para uso no Titanium Studio.

Os Requisitos Não Funcionais (RNF) da ferramenta são:

a) RNF01: ser desenvolvida no ambiente do Titanium Studio;
b) RNF02: ser desenvolvida para a plataforma iOS (iPad);
c) RNF03: ser implementada utilizando a linguagem de programação JavaScript;
d) RNF04: utilizar XML para armazenamento dos elementos gráficos suportados;
e) RNF05: utilizar o banco de dados SQLite para armazenar as interfaces criadas.

3.2 ESPECIFICAÇÃO

O desenvolvimento da ferramenta iniciou com a definição dos elementos e propriedades gráficas suportadas. Em seguida, a ferramenta foi especificada utilizando diagramas da UML aliados ao MER[1]. Sendo assim, os casos de uso são apresentados através de um diagrama de casos de uso, seguido da descrição do cenário de cada um. As classes são apresentadas através de diagramas de classes separados em três pacotes, um de modelo, um de visão e outro de controle. Também são apresentados dois diagramas de sequência.

[1] Na especificação dos diagramas da UML foi utilizada a Astah Community versão 6.7.0 para Mac/OS, enquanto do diagrama MER foi desenvolvido utilizando a ferramenta DB Designer na versão 4.0.5.4 para Mac/OS.

Complementando a especificação, as entidades do banco de dados são mostradas através de um diagrama MER.

3.2.1 Elementos gráficos suportados

Foram levantados os principais elementos gráficos utilizados nas interfaces dos aplicativos iOS. Para tanto, foram consultadas as diretrizes do HIG e, com base na experiência de usabilidade de diversos aplicativos, foram selecionados 13 elementos gráficos que a ferramenta desenvolvida suporta. O Quadro 3 exibe os elementos suportados, assim como também, o código JavaScript e XML correspondente ao elemento.

Quadro 3 – Elementos gráficos suportados

elemento	código Javascript	XML Alloy
área de texto	`textArea = Ti.UI.createTextArea ({value: 'Texto' })`	`<TextArea value='Texto'> </TextArea>`
barra de pesquisa	`searchBar = Ti.UI.createSearchBar ({barColor: '#ccc' })`	`<SearchBar barColor='#ccc'> </SearchBar>`
barra de progresso	`progressBar = Ti.UI.createProgressBar ({width: 250, min: 0, max: 100 })`	`<ProgressBar width='250' min='0' max='100' </ProgressBar>`
botão	`button = Titanium.UI.createButton ({title: 'Botão', width: 100, height: 50 })`	`<Button title='Botão' width='100' height='50' </Button>`
campo	`textField = Ti.UI.createTextField ({width: 250, height: 60 })`	`<TextField width='250' height='60' </TextField>`
data	`pickerDate = Ti.UI.createPicker ({type: Ti.UI.PICKER_TYPE_DATE, locale: 'pt' })`	`<Picker type='Ti.UI_PICKER_TYPE_DATE' locale='pt'> </Picker>`
deslizante	`slider = Ti.UI.createSlider ({min: 0, max: 100, value: 50 })`	`<Slider min='0' max='100' value='50' </Slider>`
hora	`pickerTime = Ti.UI.createPicker ({type: Ti.UI.PICKER_TYPE_TIME, locale: 'pt' })`	`<Picker type='Ti.UI_PICKER_TYPE_TIME' locale='pt'> </Picker>`
imagem	`image = Ti.UI.createImageView ({backgroundImage:'imagens/tasks.png' })`	`<ImageView image='imagens/tasks.png'> </ImageView>`
interruptor	`switch = Ti.UI.createSwitch ({value: false })`	`<Switch value='false'> </Switch>`

Quadro 3 – Elementos gráficos suportados (continuação)

elemento	código Javascript	XML Alloy
navegação	`navigation = Ti.UI.iPhone.createNavigationGroup ({window : win })`	`<NavigationGroup> <Window id='win'> </Window> </NavigationGroup>`
rótulo	`label = Ti.UI.createLabel ({text: 'Rótulo', width: 100, height: 50 })`	`<Label width='100' height='50'> Rótulo </Label>`
visão web	`webView = Ti.UI.createWebView ({html: '<b> Visão web </b>' })`	`<WebView html='<b> Visão web </b>' />`

3.2.2 Propriedades gráficas suportadas

Foram levandas as principais propriedades gráficas utilizadas no desenvolvimento de aplicativos para a plataforma iOS considerando os elementos suportados pela ferramenta.

O Quadro 4 exibe as 17 propriedades gráficas que a ferramenta possibilita editar, assim como também, quais elementos as suportam.

Quadro 4 – Propriedades gráficas suportadas

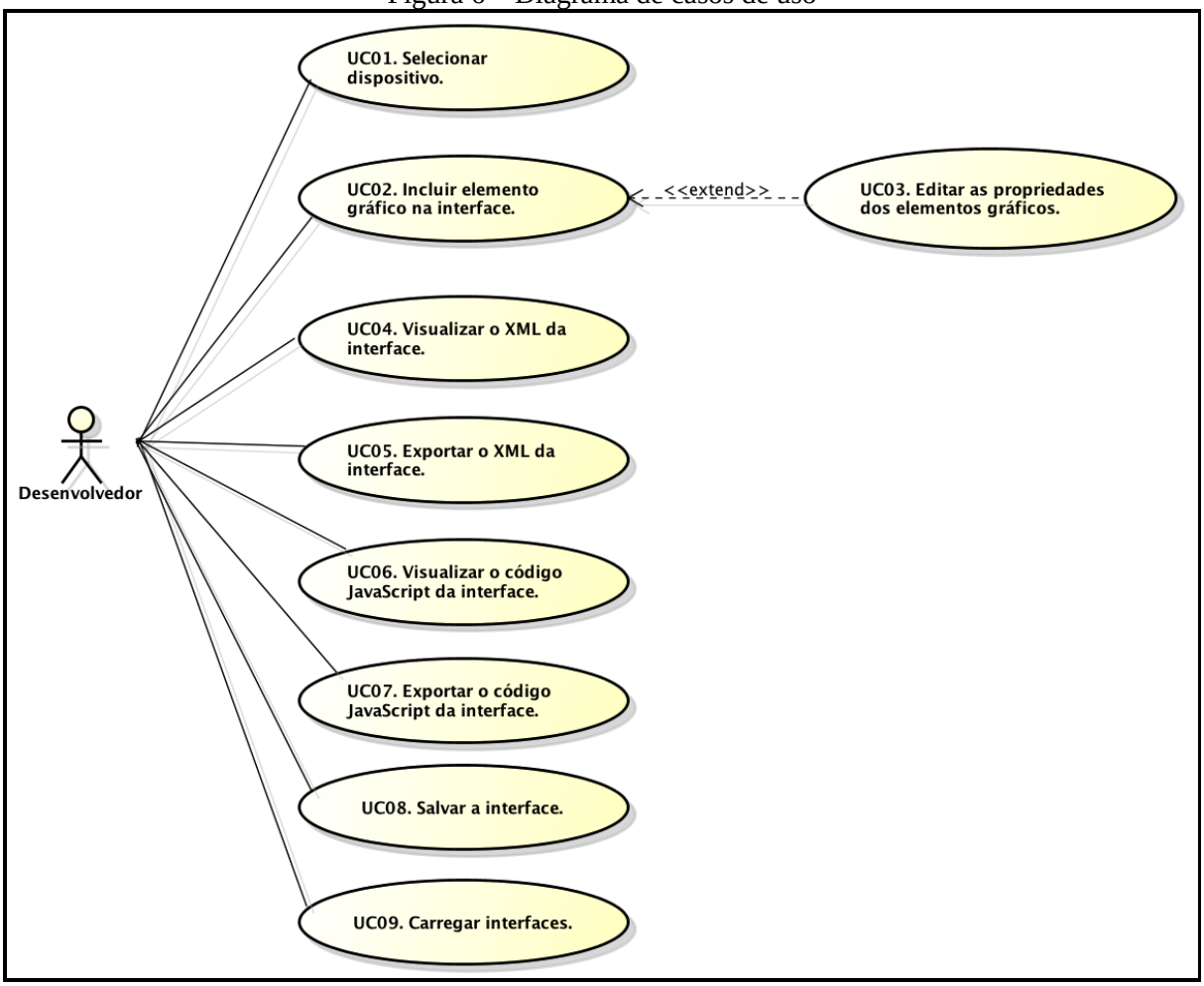
propriedade	descrição	elementos suportados
topo	margem superior do elemento	todos.
esquerda	margem esquerda do elemento.	todos.
altura	altura do elemento.	area de texto, barra de pesquisa, botão, campo, imagem, rótulo e visão web.
largura	largura do elemento.	todos, exceto o interruptor.
cor	cor da letra que aparece no elemento.	area de texto, botão, campo e rótulo.
cor de fundo	cor de fundo do elemento.	area de texto, campo e imagem, rótulo e visão web.
titulo	titulo do elemento.	botão.
texto	texto no elemento.	rótulo.
valor	texto/valor no elemento.	area de texto, barra de pesquisa e campo.
cor da barra	cor de fundo das barras.	barra de progresso, barra de pesquisa e navegação.
imagem de fundo	imagem a ser exibida no fundo do elemento.	botão, imagem, rótulo, barra de pesquisa, deslizante e campo.
HTML	código HTML para exibir.	visão web.
idioma	idioma para exibir os meses/hora correspondentes.	data e hora.
mínimo	valor mínimo permitido.	deslizante.
máximo	valor máximo permitido.	deslizante.
data mínima	menor data permitida.	data.
data máxima	maior data permitida.	data.

3.2.3 Casos de uso

A partir da elicitação dos requisitos da ferramenta, foram desenvolvidos nove casos de uso. Esses casos de uso objetivam organizar os requisitos em funcionalidades que possam ser executadas de forma simples pelo usuário. Cada caso de uso foi detalhado em cenários e vinculado a pelo menos um RF. Essa vinculação objetiva facilitar a identificação do propósito do caso de uso e justificar sua existência.

Os casos de uso desenvolvidos são desempenhados exclusivamente por um ator, o `desenvolvedor`. O `desenvolvedor` representa o único usuário do sistema, sendo capaz de interagir através de todos os casos de uso. A Figura 6 contém o diagrama de casos de uso.

Figura 6 – Diagrama de casos de uso



Inicialmente, deve ser selecionado o dispositivo para o qual se deseja criar a interface gráfica, sendo que entre as opções estão: iPhone 4, iPhone 5 e iPad. O caso de uso `Selecionar dispositivo` tem o objetivo de desenhar na tela da ferramenta o dispositivo selecionado e estabelecer os limites para a movimentação dos elementos gráficos respeitando

as dimensões do dispositivo. O Quadro 5 contém os cenários do caso de uso `Selecionar dispositivo`.

Quadro 5 – Caso de uso: `Selecionar dispositivo`

UC01. Selecionar dispositivo: possibilita selecionar para qual dispositivo a interface gráfica será criada.	
Requisitos atendidos	RF01.
Pré-condições	Nenhuma.
Cenário principal	1) O sistema exibe a tela principal contendo o dispositivo padrão (iPhone 5) e as opções de selecionar o dispositivo, inserir elementos, salvar a interface, exportar a interface e visualizar as interfaces salvas. 2) O desenvolvedor clica no botão `Devices`. 3) O sistema abre uma janela contendo as opções disponíveis, incluindo o iPhone 4, iPhone 5 e iPad. 4) O desenvolvedor seleciona o dispositivo desejado. 5) O sistema desenha o dispositivo selecionado e estabelece os limites para movimentação dos elementos gráficos.
Fluxo alternativo 01	No passo 4, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Dispositivo selecionado apresentado na tela.

Em seguida, deve ser selecionado um determinado elemento gráfico para incluir na interface que está sendo criada. O caso de uso `Incluir elemento gráfico na interface` tem o objetivo de apresentar ao desenvolvedor os elementos gráficos que a ferramenta suporta e permitir a seleção de um dado elemento para a inclusão do mesmo na interface. O Quadro 6 contém os cenários do caso de uso `Incluir elemento gráfico na interface`.

Quadro 6 – Caso de uso: `Incluir elemento gráfico na interface`

UC02. Incluir elemento gráfico na interface: possibilita visualizar e selecionar qual elemento será incluído na interface.	
Requisitos atendidos	RF02, RF03.
Pré-condições	Um dispositivo deve estar apresentado na tela.
Cenário principal	1) O desenvolvedor clica no botão `Elements`. 2) O sistema abre uma janela contendo todos os elementos gráficos suportados. 3) O desenvolvedor seleciona o elemento desejado. 4) O sistema inclui o elemento selecionado dentro da área do dispositivo.
Fluxo alternativo 01	No passo 4, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Elemento incluído e exibido na interface.

Para editar as propriedades de um determinado elemento gráfico, o mesmo deve ser selecionado em um primeiro momento. Após a seleção do elemento e da abertura da tela de propriedades, o desenvolvedor consegue visualizar quais propriedades o elemento possui assim como seus respectivos valores. O Quadro 7 contém os cenários do casos de uso `Editar as propriedades dos elementos gráficos`.

Quadro 7 – Caso de uso: `Editar as propriedades dos elementos gráficos`

UC03. Editar as propriedades dos elementos gráficos: possibilita visualizar e editar as propriedades de determinado elemento gráfico da interface.	
Requisitos atendidos	RF04.
Pré-condições	Um elemento deve estar incluído na interface.
Cenário principal	1) O desenvolvedor seleciona o elemento que deseja editar. 2) O sistema abre uma janela contendo todas as propriedades que o elemento suporta. 3) O desenvolvedor edita as propriedades. 4) O sistema altera as propriedades do elemento.
Fluxo alternativo 01	No passo 4, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Propriedades do elemento alteradas.

O objetivo do caso de uso `Visualizar o XML da interface` (Quadro 8) é permitir ao desenvolvedor visualizar na própria tela do iPad o XML da interface que ele está desenvolvendo. O XML segue o padrão do *framework* Alloy do Titanium Studio e serve como base para a geração do código JavaScript.

Quadro 8 – Caso de uso: `Visualizar o XML da interface`

UC04. Visualizar o XML da interface: permite visualizar no iPad o XML da interface.	
Requisitos atendidos	RF06.
Pré-condições	Uma interface gráfica deve estar exibida na tela.
Cenário principal	1) O desenvolvedor clica no botão `Export`. 2) O sistema apresenta as opções: `See the XML`, `See the code`, `Export the XML` e `Export the code`. 3) O desenvolvedor seleciona `See the XML`. 4) O sistema exibe o XML correspondente à interface.
Fluxo alternativo 01	No passo 5, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	XML exibido na tela.

O objetivo do caso de uso `Exportar o XML da interface`, cujos cenários encontram-se no Quadro 9, é permitir ao desenvolvedor exportar via e-mail o XML da interface.

Quadro 9 – Caso de uso: `Exportar o XML da interface`

UC05. Exportar o XML da interface: permite exportar o XML da interface via e-mail.	
Requisitos atendidos	RF06, RF07.
Pré-condições	Uma interface gráfica deve estar exibida na tela. Uma conexão com a internet deve estar disponível.
Cenário principal	1) O desenvolvedor clica no botão `Export`. 2) O sistema apresenta as opções: `See the XML`, `See the code`, `Export the XML` e `Export the code`. 3) O desenvolvedor seleciona `Export the XML`. 4) O sistema abre o aplicativo de correio eletrônico com o corpo do e-mail contendo o XML da interface. 5) O desenvolvedor informa o destinatário e confirma o envio do e-mail.
Fluxo alternativo 01	No passo 5, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	XML exportado via e-mail.

O objetivo do caso de uso `Visualizar o código JavaScript da interface` é permitir ao desenvolvedor visualizar na própria tela do iPad o código JavaScript da interface que ele está desenvolvendo. Os cenários desse caso de uso encontram-se no Quadro 10.

Quadro 10 – Caso de uso: `Visualizar o código JavaScript da interface`

UC06. Visualizar o código JavaScript da interface: permite visualizar no iPad o código JavaScript da interface.	
Requisitos atendidos	RF06.
Pré-condições	Uma interface gráfica deve estar exibida na tela.
Cenário principal	1) O desenvolvedor clica no botão `Export`. 2) O sistema apresenta as opções: `See the XML`, `See the code`, `Export the XML` e `Export the code`. 3) O desenvolvedor seleciona `See the code`. 4) O sistema exibe o código JavaScript correspondente à interface.
Fluxo alternativo 01	No passo 5, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Código JavaScript exibido na tela.

O objetivo do próximo caso de uso é permitir ao desenvolvedor exportar via e-mail o código JavaScript da interface. O Quadro 11 contém os cenários do caso de uso `Exportar o código JavaScript da interface`.

Quadro 11 – Caso de uso: `Exportar o código JavaScript da interface`

UC07. Exportar o código JavaScript da interface: permite exportar via e-mail o código JavaScript da interface.	
Requisitos atendidos	RF06, RF07.
Pré-condições	Uma interface gráfica deve estar exibida na tela. Uma conexão com a internet deve estar disponível.
Cenário principal	1) O desenvolvedor clica no botão `Export`. 2) O sistema apresenta as opções: `See the XML`, `See the code`, `Export the XML` e `Export the code`. 3) O desenvolvedor seleciona `Export the code`. 4) O sistema abre o aplicativo de correio eletrônico com o corpo do e-mail contendo o código JavaScript da interface. 5) O desenvolvedor informa o destinatário e confirma o envio do e-mail.
Fluxo alternativo 01	No passo 5, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Código JavaScript exportado via e-mail.

Para visualizar e editar uma determinada interface futuramente, a ferramenta deve permitir salvar as interfaces criadas localmente num banco de dados. O caso de uso `Salvar a interface` permite ao desenvolvedor atribuir um nome a uma interface para facilitar a busca da mesma quando for necessário. O Quadro 12 contém os cenários do caso de uso `Salvar a interface`.

Quadro 12 – Caso de uso: `Salvar a interface`

UC08. Salvar a interface: permite salvar as interfaces num banco de dados local.	
Requisitos atendidos	RF05.
Pré-condições	Uma interface gráfica deve estar exibida na tela. Uma conexão com o banco de dados local deve estar disponível.
Cenário principal	1) O desenvolvedor clica no botão `Save`. 2) O sistema abre uma janela para inserção do nome da nova interface. 3) O desenvolvedor digita o nome da interface. 4) O desenvolvedor clica em `Ok`. 5) O sistema salva a interface no banco de dados local.
Fluxo alternativo 01	No passo 5, caso a interface já exista: 1) O sistema atualiza automaticamente a interface.
Fluxo alternativo 02	No passo 6, caso o desenvolvedor cancele a operação: 1) O sistema exibe a tela principal da ferramenta.
Pós-condições	Interface salva no banco de dados local.

As interfaces criadas pelo desenvolvedor podem ser armazenadas localmente num banco de dados. O objetivo do caso de uso `Carregar interfaces` é permitir ao desenvolvedor visualizar e carregar, para posteriormente editar, as interfaces que já foram criadas e encontram-se armazenadas neste banco de dados. O Quadro 13 contém os cenários do caso de uso.

Quadro 13 – Caso de uso: `Carregar interfaces`

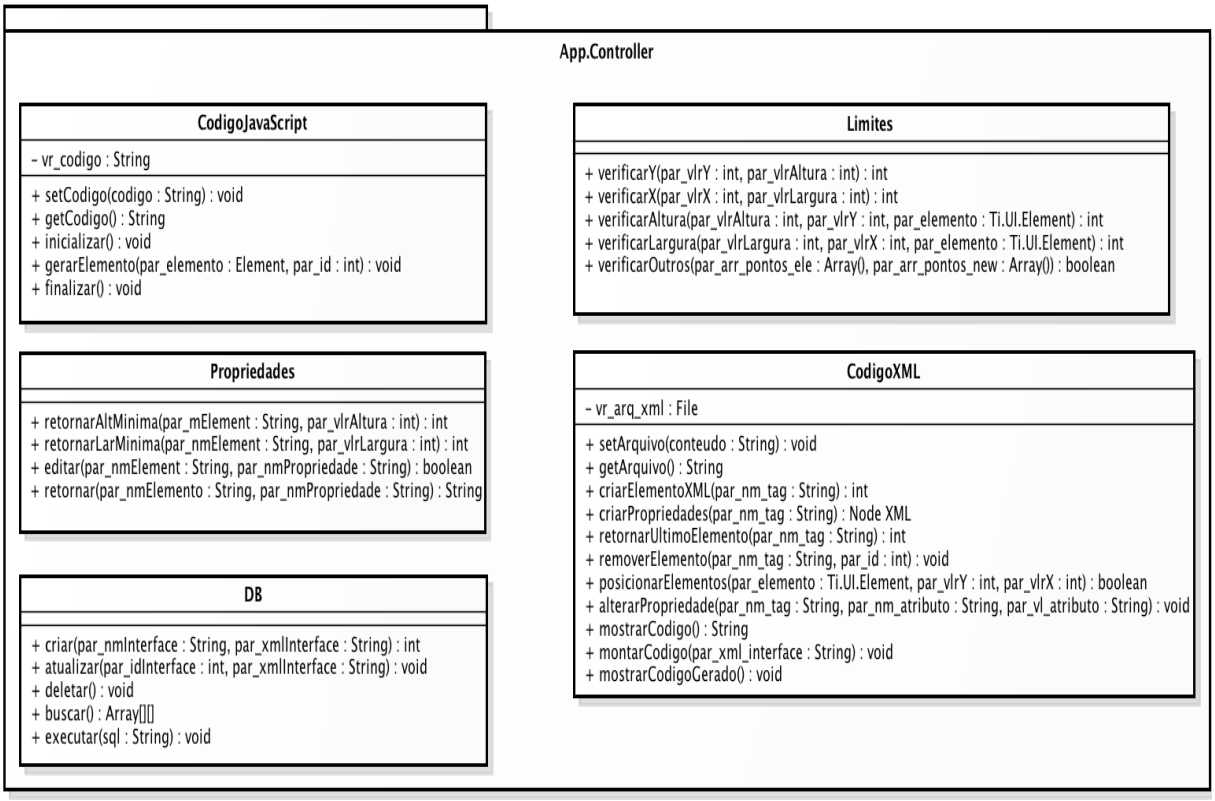
UC09. Carregar interfaces: permite carregar as interfaces de um banco de dados local.	
Requisitos atendidos	RF05.
Pré-condições	Uma conexão com o banco de dados local deve estar disponível. Pelo menos uma interface gráfica deve estar salva no banco de dados.
Cenário principal	1) O desenvolvedor clica no botão `Interfaces`. 2) O sistema abre uma janela exibindo as interfaces já criadas. 3) O desenvolvedor navega entre as interfaces criadas e seleciona a que deseja carregar. 4) O sistema carrega a interface na tela principal.
Pós-condições	Interface exibida na tela.

3.2.4 Classes da ferramenta

As classes da ferramenta estão agrupadas em pacotes, respeitando o padrão de projeto MVC. Os pacotes são `App.Controller`, `App.Model` e `App.View`.

O pacote `App.Controller` (Figura 7) contém as classes de controle da ferramenta.

Figura 7 – Diagrama de classes do pacote `App.Controller`



A classe `CodigoJavaScript` é responsável por toda a geração do código JavaScript das interfaces. Essa classe contém um único atributo chamado `vr_codigo`, que contém o código gerado. O método `inicializar` é executado no início do processo e tem como objetivo gerar o código da janela que conterá todos os elementos adicionados na interface. O

método `gerarElemento` é executado para gerar o código JavaScript de cada elemento da interface. O método `finalizar` gera o código para a abertura da janela.

A classe `Propriedades` tem como objetivo verificar certas características dos elementos, como por exemplo, altura e largura mínimas (métodos `retornarAltMinima` e `retornarLarMinima`, respectivamente). O método `editar` verifica se um determinado elemento pode ou não editar uma propriedade. Já o método `retornar` retorna o valor padrão de uma determinada propriedade para um elemento.

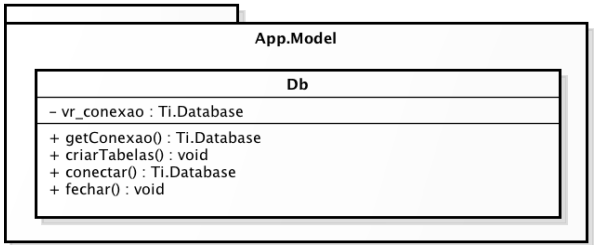
A classe `DB` possui as funções de acesso ao banco de dados local: o método `criar` é responsável por criar uma nova interface no banco de dados, com o nome e o XML da interface passados como parâmetro; o método `atualizar` atualiza o XML de uma dada interface; `deletar` remove todas as interfaces salvas no banco; `buscar` retorna todas as interfaces salvas e o método `executar` permite realizar uma consulta no banco.

A classe `Limites` contém as funções para controle de movimentação dos elementos, através dos métodos `verificarY`, que verifica o valor máximo no eixo Y, e `verificarX`, que verifica o valor máximo no eixo X. Possui também métodos para verificar a altura e a largura máxima permitidas (`verificarAltura` e `verificarLargura`, respectivamente). Já o método `verificarOutros` é responsável por impedir que um elemento sobreponha outro na interface.

A classe `CodigoXML` gerencia o XML que serve como base para a geração de código. Os métodos `criarElementoXML` e `criarPropriedades` são responsáveis por criar um novo elemento no XML e suas principais propriedades. O método `retornarUltimoElemento` retorna um inteiro contendo o identificador do último elemento criado. O método `removerElemento` remove um determinado elemento do XML. O método `posicionarElementos` é acionado cada vez que um elemento é reposicionado e o mesmo ativa a execução do já comentado método `verificarOutros`. O método `alterarPropriedade` é responsavel por criar ou alterar uma determinada propriedade de um elemento. O método `mostrarCodigo` percorre todos os elementos do XML para gerar o codigo JavaScript da interface. Enquanto o método `montarCodigo` é similar ao anterior com a diferença que monta o código a partir de uma interface que já está salva no banco de dados. Por fim, o método `mostrarCodigoGerado`, chamado logo após o `mostrarCodigo`, exibe na tela do desenvolvedor o código JavaScript.

O pacote `App.Model` contém a classe que define o modelo de objetos utilizados na aplicação e funções que agem sobre os mesmos. A Figura 8 contém o diagrama de classes do pacote `App.Model`.

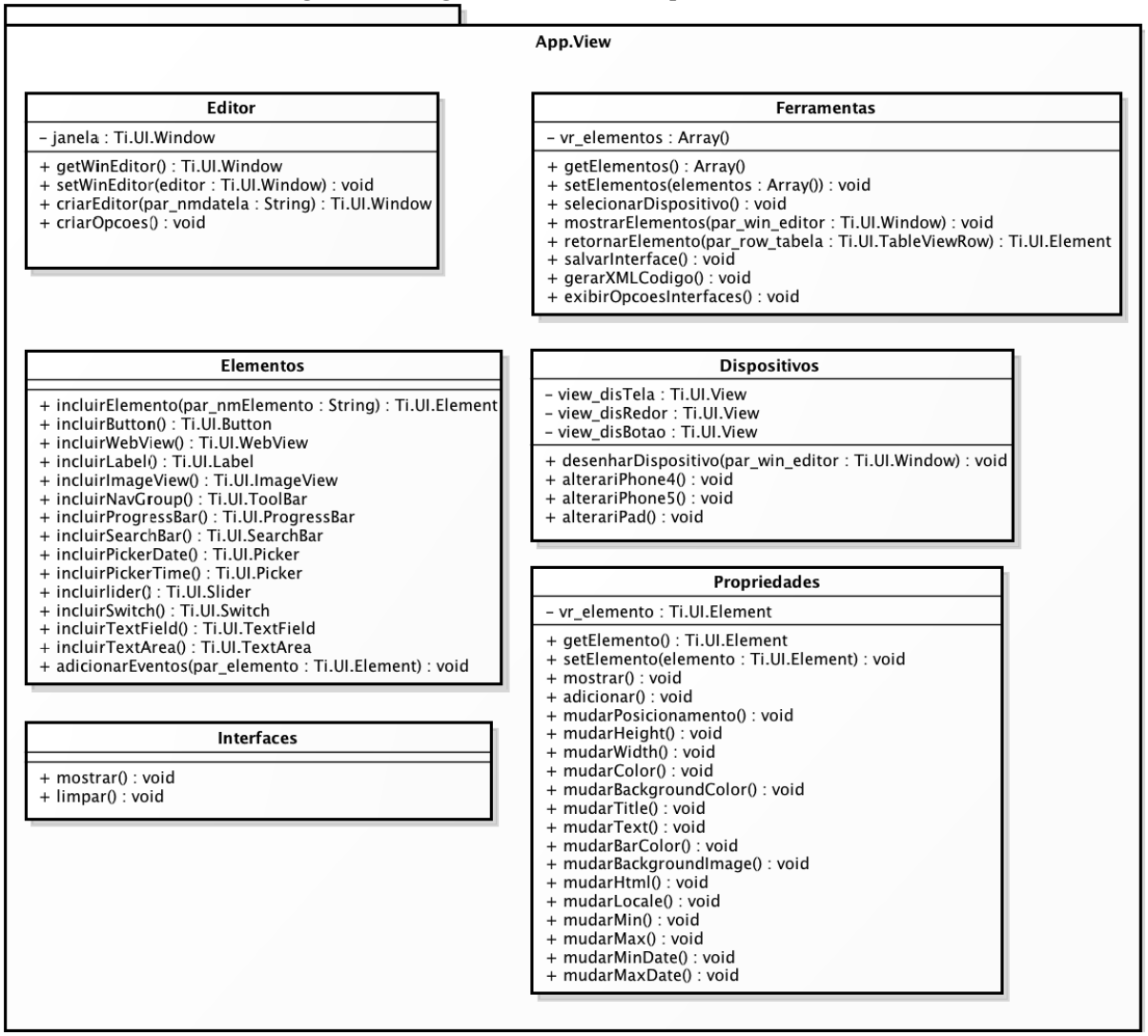
Figura 8 – Diagrama de classes do pacote `App.Model`



A classe `DB` contém um único atributo chamado de `vr_conexao` que contém a conexão com o banco de dados local. O método `criarTabelas` é ativado somente a primeira vez que a ferramenta for executada e o objetivo é criar a estrutura das tabelas necessárias à aplicação. O método `conectar` faz a conexão com o banco de dados e o método `fechar` fecha a conexão com o banco de dados.

O pacote `App.View` (Figura 9) contém as classes que representam os elementos gráficos da aplicação. Essas classes inicializam rotinas implementadas no pacote `App.Controller`.

Figura 9 – Diagrama de classes do pacote `App.View`



A classe `Editor` é a tela principal da ferramenta. Possui somente um atributo chamado `janela` que é a visão que exibe o dispositivo para qual a interface será criada. Esta visão possui uma barra superior na qual estão as opções disponíveis da ferramenta. A visão é criada pelo método `criarEditor` e as opções da ferramenta pelo método `criarOpcoes`.

A classe `Elementos` cria os elementos adicionados na interface que está sendo elaborada. Cada vez que é criado um elemento, o método `incluirElemento` é acionado, sendo que o mesmo delega para um método específico dependendo o tipo de elemento criado. Por exemplo, se é criado um elemento do tipo `Button`, o método acionado é `incluirButton`, se o elemento for do tipo `WebView`, o método correspondente é o `incluirWebView`, e assim por adiante para os demais elementos. O método `adicionarEventos` é ativado cada vez que algum elemento sofre movimentação ou é selecionado para editar suas propriedades.

Na classe `Interfaces`, o método `mostrar` exibe todas as interfaces salvas no banco de dados. O método `limpar` é acionado cada vez que uma interface é selecionada para edição e

possui a função de remover os elementos da interface que está exibida na tela para posteriormente carregar e exibir os elementos da interface selecionada. Este metodo também é utilizado quando se cria uma nova interface.

A classe `Ferramentas` exibe e controla os eventos das diversas opções que a ferramenta disponibiliza. O método `selecionarDispositivo` é encarregado de exibir os diferentes tipos de dispositivos na qual a interface pode ser criada. O método `mostrarElementos` relaciona os elementos que podem ser utilizados para a criação da interface, acionando, para cada elemento contemplado, o `retornarElemento` para exibir o elemento em questão. O método `salvarInterface` ativa a chamada dos métodos `criar` ou `atualizar` da classe `DB` do pacote `APP.Controller`, dependendo a situação da interface. O método `gerarXMLCodigo` exibe a tela com as opções de visualizar/exportar o XML e visualizar/exportar o código JavaScript. O método `exibirOpcoesInterfaces` exibe a tela contendo as opções de criar uma nova interface, visualizar as interfaces já criadas e deletar todas as interfaces.

A classe `Dispositivos` possui três atributos para desenhar os dispositivos na tela do `Editor`, são eles: `view_disTela, view_disRedor e view_disBotao`. Cada vez que for alterado o dispositivo, o método chamado é o `desenharDispositivo` e dependendo o dispositivo que for selecionado, um dos seguintes métodos é acionado para criar o desenho: `alterariPhone4`, `alterariPhone5` e `alterariPad`.

A classe `Propriedades` tem por função exibir e tratar as propriedades de cada elemento. O método `mostrar` exibe as propriedades de um dado elemento. Este método aciona o método `adicionar`, o qual delega para cada propriedade a chamada do seu método correspondente, como por exemplo, `mudarPosicionamento`, `mudarHeight`, `mudarWidth`, entre outros.

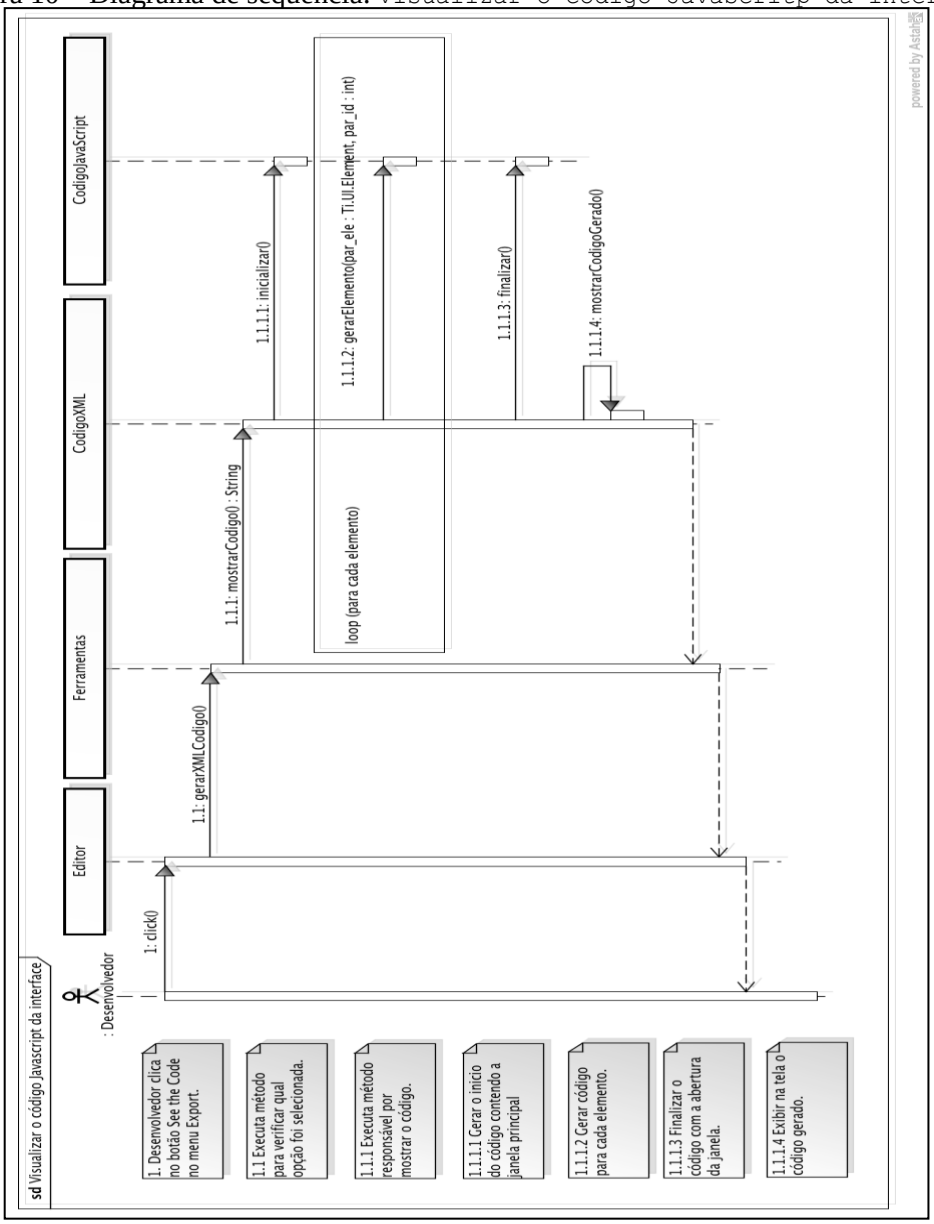
3.2.5 Diagramas de sequência

Os diagramas de sequência têm como finalidade mostrar a troca de mensagens entre as classes criadas, facilitando a implementação dos casos de uso. Nesta seção são apresentados os diagramas de sequência para os casos de uso `Visualizar o código JavaScript da interface` e `Carregar interfaces`.

O caso de uso `Visualizar o código JavaScritp da interface` parte da ação do desenvolvedor ao clicar no botão `See the Code` na opção `Export`, o qual aciona o método `gerarXMLCodigo` da classe `Ferramentas`, que por sua vez verifica qual opção foi selecionada.

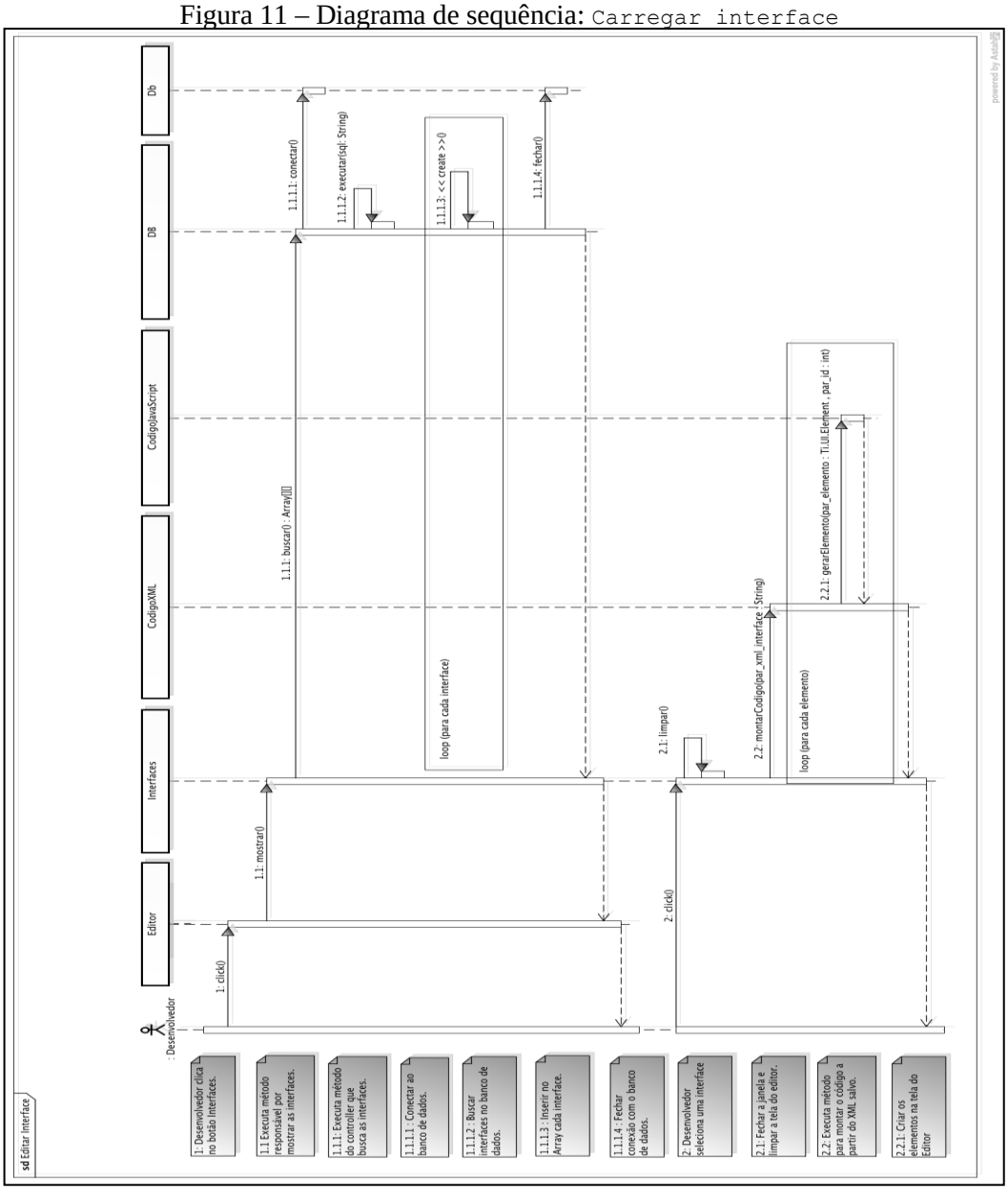
Nesta situação, o método a ser chamado é o `mostrarCodigo` da classe `CodigoXML` que retorna um `string` contendo o código gerado. Na geração do código a partir da classe `CodigoJavaScript`, primeiramente é chamado o método `inicializar` que gera o código para criar a janela que conterá os elementos incluídos na interface. Em seguida, todos os elementos da interface são percorridos e, em cada iteração, é executado o método `gerarElemento`. Por fim, é acionado o método `finalizar`. Após a finalização deste processo, o código é mostrado na tela do dispositivo através do método `mostrarCodigoGerado` da classe `CodigoXML`. A Figura 10 apresenta o diagrama de sequência do caso de uso `Visualizar o código JavaScript da interface`.

Figura 10 – Diagrama de sequência: `Visualizar o código JavaScritp da interface`



O caso de uso `Carregar interface` parte da ação do desenvolvedor ao clicar no botão `Interfaces`. Em seguida é executado o método `mostrar` da classe `Interfaces`. Este método é o responsável por exibir as interfaces salvas no banco de dados local. Nele é acionado o método `buscar` da classe `DB` que retorna a listagem de todas as interfaces salvas no banco de dados local. Após a visualização e a seleção da interface desejada pelo desenvolvedor, é executado o método `limpar` da classe `Interfaces` que efetua a remoção de qualquer elemento já presente na tela da ferramenta. Em seguida, a partir do XML da interface, é montado e executado o código JavaScript para inserir todos os elementos desta interface na tela do `Editor`, permitindo a edição da mesma. A Figura 11 apresenta o diagrama de sequência `Carregar interface`.

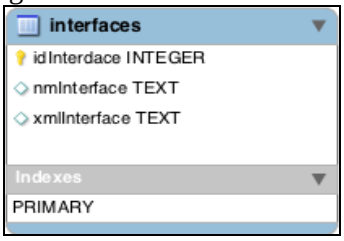
Figura 11 – Diagrama de sequência: `Carregar interface`



3.2.6 MER

Figura 12 apresenta o MER do banco de dados da ferramenta.

Figura 12 – MER da ferramenta



A tabela `Interfaces` contém as informações das interfaces salvas pelo desenvolvedor. Esta tabela possui três campos: `idInterface` é chave primária da tabela, `nmInterface` é o

nome dado à interface pelo desenvolvedor quando da criação e `xmlInterface` é o XML que contém toda a estrutura da interface, ou seja, os elementos e suas propriedades.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da ferramenta desenvolvida.

3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da ferramenta foi utilizada a linguagem de programação JavaScript no ambiente do Titanium Studio, versão 3.1.1. Para possibilitar os testes da ferramenta nos dispositivos físicos foi utilizado o Xcode SDK, versão 5.0. Os testes da ferramenta foram realizados em um dispositivo iPad 4 com 16GB e no emulador do iPad disponibilizado pelo Xcode SDK. Os testes das interfaces geradas pela ferramenta foram realizados em um iPhone 4 com 16GB e no mesmo iPad citado anteriormente, além dos emuladores do Xcode para ambas as plataformas.

Para armazenamento das interfaces gráficas criadas foi utilizado o banco de dados SQLite, o qual pode ser criado e manipulado diretamente via programação na aplicação desenvovida no ambiente do Titanium Studio.

3.3.2 Arquivo `tiapp.xml`

O arquivo `tiapp.xml` é um arquivo no formato XML que contém informações para configurações básicas das aplicações desenvolvidas no ambiente Titanium Studio. Certas informações são obrigatórias, como o nome e a versão da aplicação, a versão do ambiente e as plataformas que serão utilizadas. Além das citadas, existem outras informações que dependem de cada aplicação e são incluídas conforme a necessidade. O Quadro 14 apresenta o arquivo `tiapp.xml` da ferramenta desenvolvida.

Quadro 14 – Arquivo `tiapp.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
    <id>1</id>
    <name>Gerador_de_Interfaces</name>
    <version>1.0</version>
    <publisher>Gabriel</publisher>
    <description>not specified</description>
    <copyright>2013 by Gabriel</copyright>
    <icon>appicon.png</icon>
    <persistent-wifi>false</persistent-wifi>
    <prerendered-icon>false</prerendered-icon>
    <statusbar-style>default</statusbar-style>
    <statusbar-hidden>false</statusbar-hidden>
    <fullscreen>false</fullscreen>
    <navbar-hidden>false</navbar-hidden>
    <analytics>true</analytics>
    <guid>246b5479-3bf8-4bc0-be3c-06fd719bcdd1</guid>
    <property name="ti.ui.defaultunit" type="string">system</property>
    <iphone>
        <orientations device="ipad">
            <orientation>Ti.UI.PORTRAIT</orientation>
        </orientations>
    </iphone>
    <modules/>
    <deployment-targets>
        <target device="tizen">false</target>
        <target device="blackberry">false</target>
        <target device="android">false</target>
        <target device="ipad">true</target>
        <target device="iphone">false</target>
        <target device="mobileweb">false</target>
    </deployment-targets>
    <sdk-version>3.1.1.GA</sdk-version>
</ti:app>
```

Tem-se que:

a) *tag* `<name>`: define o nome da ferramenta que, entre outros locais, será exibido junto ao ícone da mesma;

b) *tag* `<icon>`: indica o arquivo contendo o ícone que será utilizado para representar a ferramenta;

c) *tag* `<orientation>`: define em que sentido será executada a ferramenta, no caso, será em retrato;

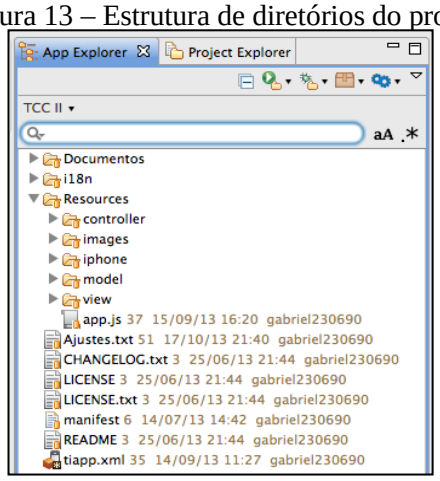
d) *tags* da seção `<deployment-targets>`: correspondem às plataformas suportadas, atribuindo o valor `true` as que são utilizadas pela ferramenta;

e) *tag* `<sdk-version>`: define que versão do Titanium Studio é utilizado no desenvolvimento da ferramenta.

3.3.3 Organização dos arquivos no projeto

O ambiente do Titanium Studio segue um padrão na estrutura dos arquivos de cada projeto. Os arquivos estão organizados em diretórios, formando uma hierarquia, conforme a Figura 13.

Figura 13 – Estrutura de diretórios do projeto



O diretório `Resources` armazena os arquivos JavaScript. Dentro dele são criados diretórios específicos para cada plataforma utilizada no projeto. Nesses diretórios são armazenados arquivos correspondentes a componentes específicos utilizados em cada plataforma. Os outros diretórios contêm os componentes principais do projeto. O diretório `controller` armazena as classes do pacote `App.Controller`. O diretório `images` contém as imagens utilizadas na interface gráfica da ferramenta. O diretório `model` contém os arquivos das classes do pacote `App.Model` e o diretório `view` contém os arquivos das classes do pacote `App.View`. O arquivo `app.js` é o arquivo principal da aplicação, é o primeiro a executar e é responsável por iniciar os componentes necessários para a execução da ferramenta.

3.3.4 Principais métodos implementados

Cada vez que é criado um novo elemento na interface, o mesmo deve ser armazenado no XML que contém todas as informações sobre esta interface. O método responsável por isto é o `criarElemento` da classe `CodigoXML`, exibido no Quadro 15.

Quadro 15 – Método `criarElemento`

```
1   function criarElemento(par_nm_tag) {
2     try {
3       var new_ele = criarPropriedades (par_nm_tag);
4       var no_window =
5       glb_arq_xml.documentElement.getElementsByTagName("Window").item(0);
6       no_window.appendChild(new_ele);
7     } catch (Exception) {
8       return;
9     }
10
11    // obter o ID criado e retornar o mesmo
12    var new_id = retornarUltimoElemento(par_nm_tag);
13    return new_id;
14  }
15
```

O método `criarElemento` recebe como parâmetro o tipo de elemento que está sendo criado, ou seja, se é um `Button`, um `WebView`, um `TextField`, e assim por adiante, que é

utilizado na criação da *tag* no XML no método `criarPropriedades` na mesma classe (linha 4), o qual pode ser visto no Quadro 16. O elemento criado é inserido no XML da interface (linha 3). O método `criarElemento` retorna o `id` do elemento criado (linha 14).

Quadro 16 – Método `criarPropriedades`

```
1   function criarPropriedades(par_nm_tag) {
2     // criar uma tag
3     glb_new_ele = glb_arq_xml.createElement(par_nm_tag);
4     var new_id = 0;
5     try {
6       // obter altura e largura padrão do elemento
7       var alt_padrao = retornar(par_nm_tag, 'altura_padrao');
8       var lar_padrao = retornar(par_nm_tag, 'largura_padrao');
9
10      new_id = parseInt(retornarUltimoElemento(par_nm_tag)) + 1;
11
12      glb_new_ele.setAttribute('height', alt_padrao);
13      glb_new_ele.setAttribute('width', lar_padrao);
14      glb_new_ele.setAttribute('id', new_id.toString());
15      glb_new_ele.setAttribute('top',
16      (verificaY(100, alt_padrao) - Aplicacao.difValorY).toString());
17      glb_new_ele.setAttribute('left',
18      (verificaX(100, lar_padrao) - Aplicacao.difValorX).toString());
19
20      // chamar método de cada elemento especifico
21      eval("novo" + par_nm_tag + "('" + par_nm_tag + "');");
22    } catch (Exception) {
23    } finally {
24      return glb_new_ele;
25    }
26  }
```

O método `criarPropriedades` é o encarregado por criar o elemento XML com certas propriedades básicas, como tamanho, posicionamento e identificador único (linhas 7 a 18). Além disso, ele chama um novo método dependendo o tipo de elemento através da função `eval` (linha 21). O parâmetro passado para a função `eval` é quem vai determinar qual método será acionado. Se for um elemento do tipo `Button`, o método a ser chamado é `novoButton`, que irá criar o título e a cor da letra. Se for um `TextField`, o método acionado é `novoTextField`, que criará as propriedades texto e cor de borda. E assim sucessivamente para cada elemento que a ferramenta suporta.

A geração do código JavaScript é feita através das classes `CodigoXML` e `CodigoJavaScript`. O processo inicia-se no método `mostrarCodigo` da classe `CodigoXML`, que obtém o XML que contém todos os elementos e propriedades da interface. Após isto, é gerado o código que cria uma janela padrão para a interface, através do método `inicializar` da classe `CodigoJavaScript`. O Quadro 17 apresenta o método `mostrarCodigo`.

Quadro 17 – Método `mostrarCodigo`

```
1   function mostrarCodigo(par_flgExporta) {
2     var elementos =
3     glb_arq_xml.getElementsByTagName("Window").item(0).getChildNodes();
4
5     Ti.include("/controller/gera_codigo.js");
6
7     inicializar();
8
9     for (var i = 1; i < elementos.length; i++) {
10      var elemento = elementos.item(i);
11      var nmElemento = elemento.getNodeName();
12      var idElemento = 1;
13      try {
14        idElemento =
15        elemento.getAttributes().getNamedItem("id").getNodeValue();
16      } catch (Exception) {
17        continue;
18      }
19      gerarElemento(elemento, nmElemento, idElemento, true);
20    }
21
22    finalizar();
23
24    if (par_flgExporta) {
25      geraEmailCodigo();
26    } else {
27      mostraCodigoGerado();
28    }
29  }
```

Depois da inicialização do código (linha 7), um *loop* percorre todos os elementos criados na interface (linhas 9 a 20). Para cada um deles, obtém o nome e o `id` e chama o método `gerarElemento` da classe `CodigoJavaScript` para gerar o código JavaScript correspondente a cada elemento da interface. Uma vez finalizada a geração do código dos elementos, é acionado o método `finalizar` que irá fazer a abertura da janela criada para conter todos os elementos da interface (linha 22).

O Quadro 18 apresenta um trecho do método `gerarElemento`. Um *loop* percorre todas as propriedades que um determinado elemento possui. Para cada uma delas, obtém o nome e seu respectivo valor. Ambos os valores são armazenados na variável `cod_elemento` (linha 24), que possui todo o código JavaScript do elemento em questão. Certas propriedades possuem tratamentos específicos, como é o caso das propriedades de posicionamento e tamanho (`top, left, height e width` – linhas 9 a 16) pois, caso o dispositivo para geração de código seja o iPad, os valores destas propriedades devem ser multiplicadas por 1,25. Isto se deve ao fato que o desenho do iPad na ferramenta representa somente o 80% do tamanho real da tela do dispositivo e para diminuir o máximo possível esta diferença, optou-se por aumentar o posicionamento e o tamanho dos elementos em 25%. Outro tratamento específico é feito para as propriedades `type, top, left, width e height` (linhas 18 a 22),

já que as mesmas exigem aspas simples antes e depois do valor da propriedade para um correto funcionamento no Titanium Studio.

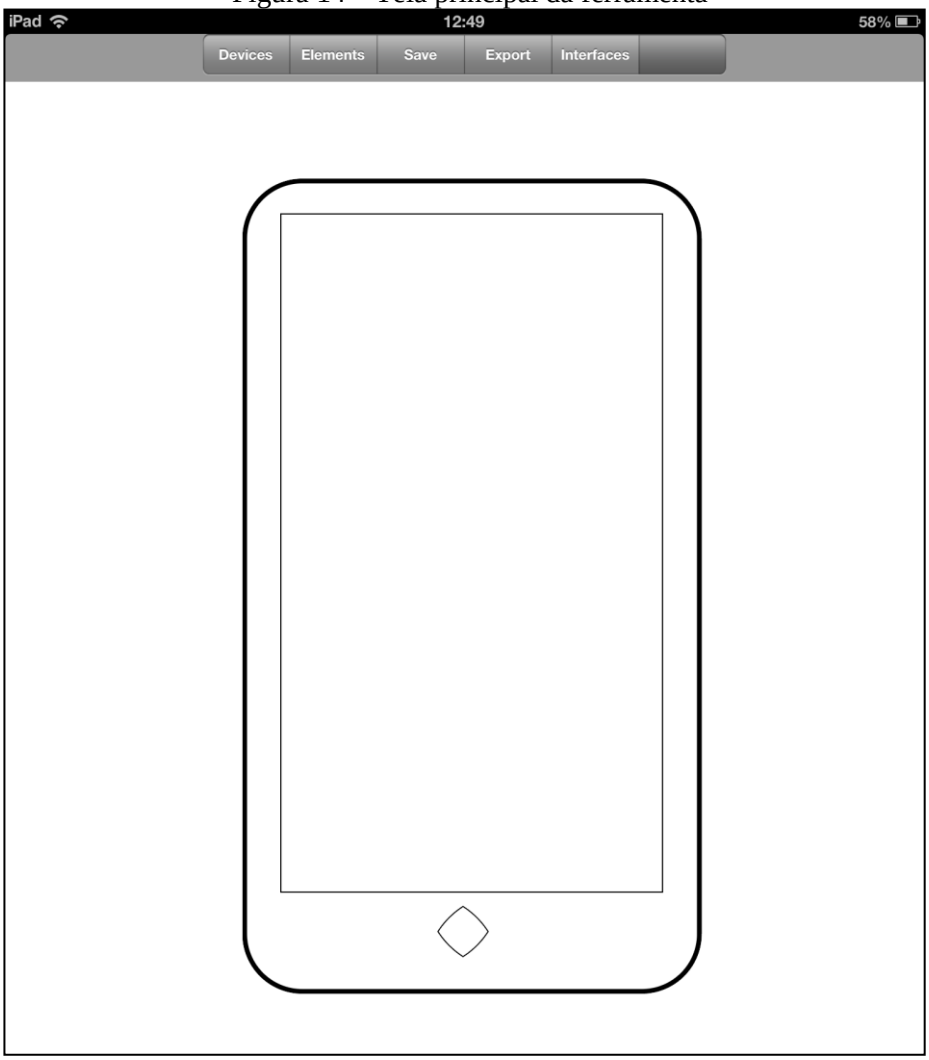
Quadro 18 – Trecho do método `gerarElemento`

```
1   // percorrer todos os atributos
2   for (var i = 0; i < arr_atributos.length; i++) {
3     var nm_atributo = arr_atributos.item(i).getNodeName();
4     var vl_atributo = arr_atributos.item(i).getNodeValue();
5     if (nm_atributo == "id") {
6       continue;
7     }
8
9     // gerar left, top, height, width com diferença no iPad por causa do 80% da
10    // tela
11    if (nm_atributo=='left' || nm_atributo=='top' || nm_atributo=="height" ||
12        nm_atributo=="width") {
13      if (Aplicacao.dispositivo == "iPad" && par_flgExporta) {
14        vl_atributo = Math.round(parseFloat(vl_atributo) * 1.25);
15      }
16    }
17
18    //se nao for type, top, left, width, height então adiciona ASPAS SIMPLES
19    if (nm_atributo!='type' && nm_atributo!='top' && nm_atributo!='left' &&
20        nm_atributo!='width' && nm_atributo!='height') {
21      vl_atributo = "'" + vl_atributo + "'";
22    }
23
24    cod_elemento += nm_atributo + " : " + vl_atributo + ",";
25  }
```

3.3.5 Operacionalidade da implementação

Esta seção apresenta as principais telas e funcionalidades da ferramenta, tais como: a seleção do dispositivo desejado, a exibição e a inclusão de elementos na interface, a edição das propriedades dos elementos gráficos, a exportação do XML e do código JavaScript e a visualização e a edição de interfaces já criadas. A Figura 14 apresenta a tela principal da ferramenta, contendo o dispositivo padrão (iPhone 5).

Figura 14 – Tela principal da ferramenta



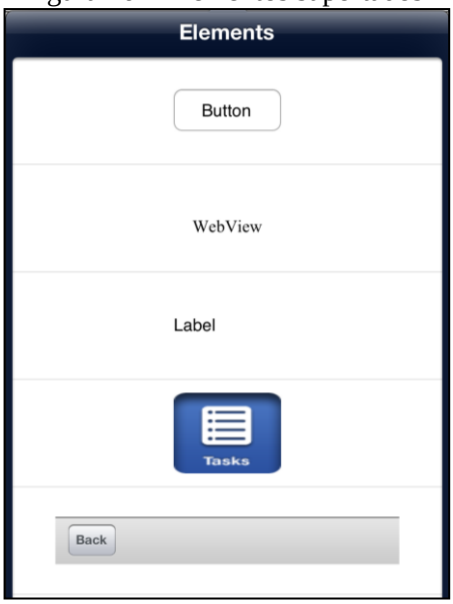
A tela principal da ferramenta disponibiliza cinco botões de ação na parte superior. O primeiro deles (`Devices`) permite selecionar para qual dispositivo a interface será criada. O segundo (`Elements`) disponibiliza os elementos gráficos para inserção na interface. O botão `Save` tem como funcionalidade salvar a interface editada no banco de dados local. O quarto botão (`Export`) permite visualizar e exportar o XML e o código JavaScript da interface. Por fim, o último botão (`Interfaces`) exibe as interfaces que estão salvas no banco de dados local, permitindo carregá-las para posterior edição das mesmas.

Ao pressionar o botão `Devices` é apresentanda a tela da Figura 15. É possível selecionar qual dispositivo será utilizado para criar a interface, sendo que as opções disponíveis são: iPhone 4, iPhone 5 e iPad. Após a seleção do dispositivo, o mesmo é desenhado na tela principal da ferramenta.

Figura 15 – Dispositivos

O botão `Elements` disponibiliza os elementos que a ferramenta suporta (Figura 16), sendo possível inseri-los na interface com um simples toque. Os elementos suportados são: `Button`, `WebView`, `Label`, `Image View`, `Navigation Group`, `Progress Bar`, `Search Bar`, `Data Picker`, `Time Picker`, `Slider`, `Switch`, `TextField` e `TextArea`.

Figura 16 – Elementos suportados



Ao selecionar algum elemento adicionado na interface, é exibida a tela de propriedades (Figura 17). As principais propriedades suportadas são: `Top`, `Left`, `Height`, `Width`, `Color`, `BackgroundColor`, `Title`, `Text` e `Value`. As demais propriedades que a ferramenta apresenta são específicas de determinados elementos, como é o caso do `Min Date` e `Max Date` que pertencem ao elemento `Data Picker`.

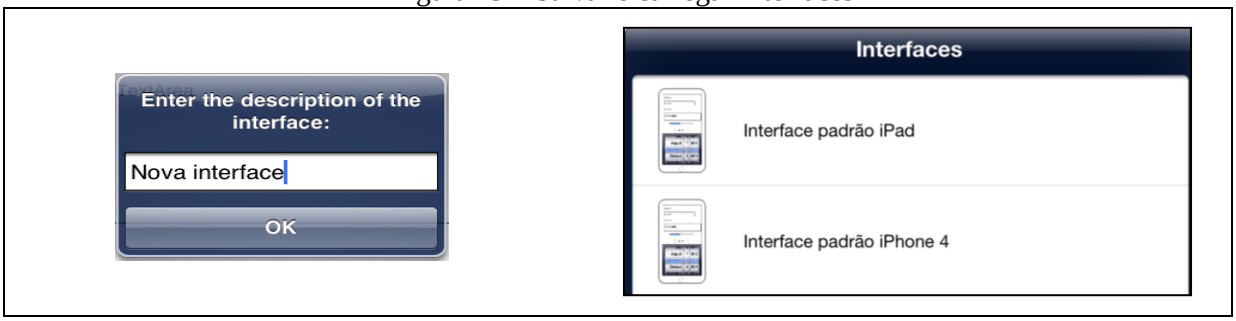
Figura 17 – Propriedades dos elementos

O botão `Export` disponibiliza opções para visualizar e exportar o XML, assim como visualizar e exportar o código JavaScript da interface criada. Na Figura 18, pode-se visualizar as opções disponíveis.

Figura 18 – Visualizar e exportar interfaces

É possível também salvar as interfaces criadas (botão `Save`) ou carregar (botão `Interfaces`) as interfaces que foram salvas no banco de dados local. A Figura 19 apresenta a tela para salvar interfaces do lado esquerdo e a tela para selecionar interfaces salvas localmente no lado direto.

Figura 19 – Salvar e carregar interfaces



Após selecionada uma interface, a mesma é carrega na ferramenta e pode ser editada. Na Figura 20 é possível visualizar a “Interface padrão iPad” contendo todos os elementos que a ferramenta suporta.

Figura 20 – Interface padrão iPad

3.4 RESULTADOS E DISCUSSÃO

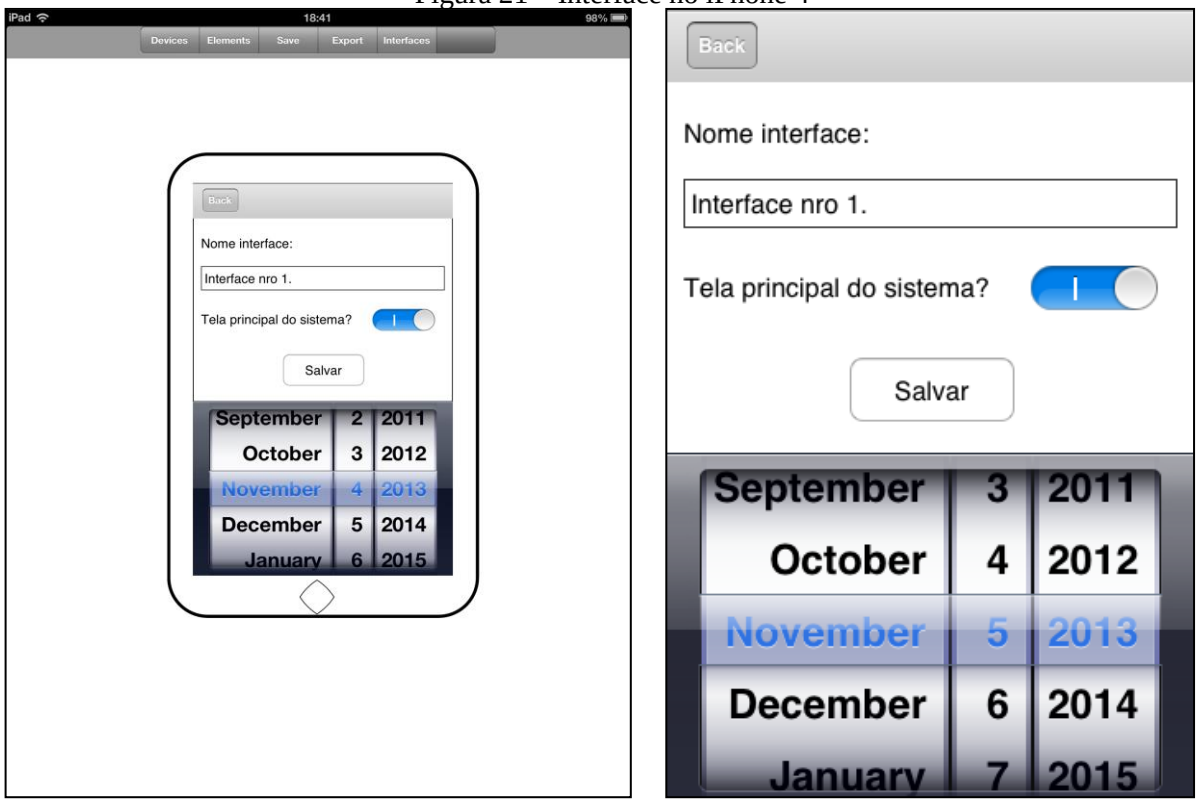
A ferramenta desenvolvida neste trabalho atende aos requisitos funcionais propostos, apresentando uma interface de fácil usabilidade e compreensão. Houve somente uma mudança com relação à proposta em um requisito não funcional. Inicialmente pensou-se em utilizar o formato *JavaScript Object Notation* (JSON) para armazenamento dos componentes gráficos utilizados. Porém, no decorrer do levantamento bibliográfico, verificou-se que já existia um padrão para criação de interfaces gráficas no formato XML, fornecido pelo *framework* Alloy da AppCelerator e este acabou sendo utilizado para o desenvolvimento da ferramenta.

Sobre as recomendações do HIG, para o desenvolvimento da ferramenta observou-se: o tamanho mínimo de 44x44 pontos dos elementos gráficos para não dificultar a manipulação

dos mesmos através de um arquivo XML de configuração da ferramenta, o desenvolvimento da maneira mais intuitiva possível e a consistência da ferramenta para que seja previsível o comportamento da mesma através de diversos testes com desenvolvedores e analistas de sistemas. No entanto, não foi possível atender outras recomendações, tais como cuidado com a rotação do dispositivo, manipulação dos objetos diretamente ao invés de usar componentes separados e customização da ferramenta no início do processo.

Foram realizados três diferentes testes sobre a ferramenta com o intuito de detectar possíveis falhas, bem como verificar o comportamento em cada um dos três dispositivos suportados. A Figura 21 apresenta o primeiro caso de teste em um iPhone 4. Do lado esquerdo da figura é possível visualizar a interface gráfica criada na ferramenta e do lado direito a interface resultante da execução do código gerado no emulador do iPhone 4.

Figura 21 – Interface no iPhone 4

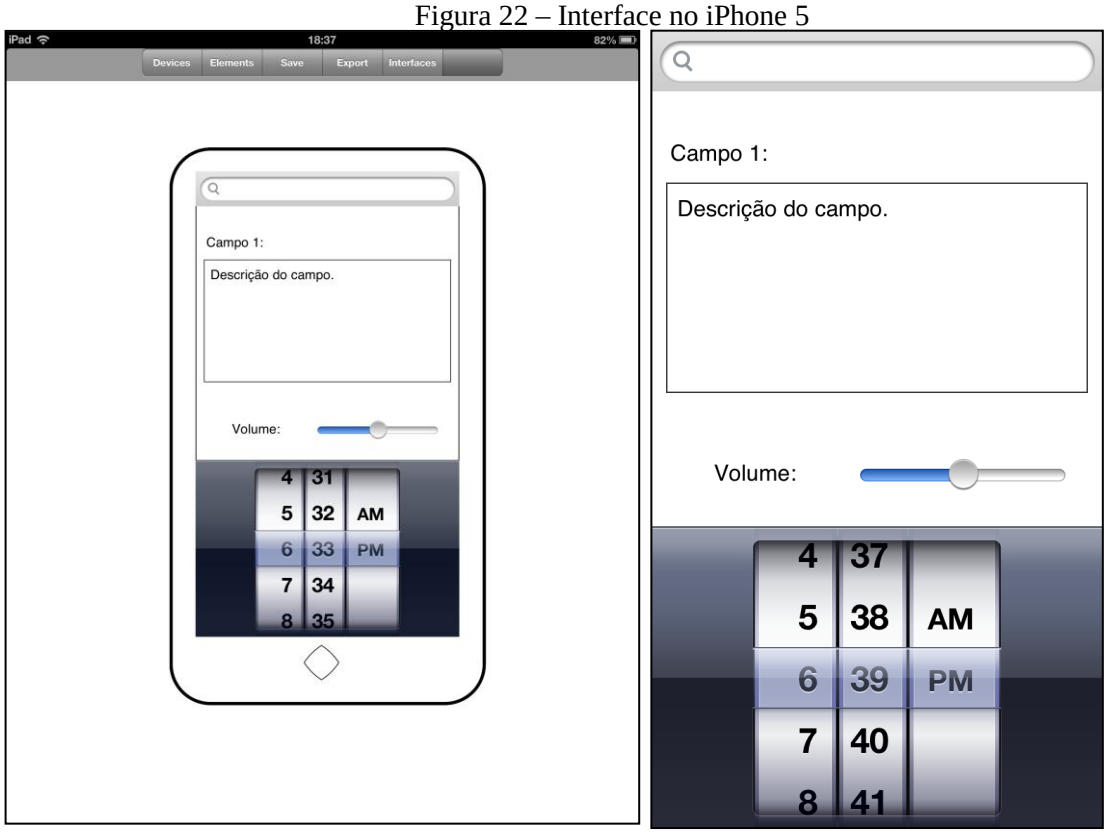


Observa-se que a interface gerada conservou todos os aspectos originais daquela criada a partir da ferramenta. A única diferença que pode ser destacada é o dia que aparece no elemento `DataPicker`. Na imagem do lado esquerdo é exibida a data de 04 de novembro de 2013 e na imagem do lado direito, a data é 05 de novembro de 2013. Isto se deve ao fato que não foi implementada a propriedade que armazena um valor para a data do elemento e, por consequência, a data padrão a ser exibida é a do dia na qual a aplicação for executada. Os elementos utilizados nesta interface foram os seguintes: `Navigation Group`, `Label`,

`TextField`, `Switch`, `Button` e `Data Picker`. O código Javascript e o XML correspondentes à interface exibida anteriormente podem ser visualizados nos apêndices A e B, respectivamente.

A Figura 22 apresenta o segundo caso de teste em um iPhone 5. Do lado esquerdo da figura é possível ver a interface criada na ferramenta e do lado direito a interface resultante da execução do código no emulador do iPhone 5.

Figura 22 – Interface no iPhone 5

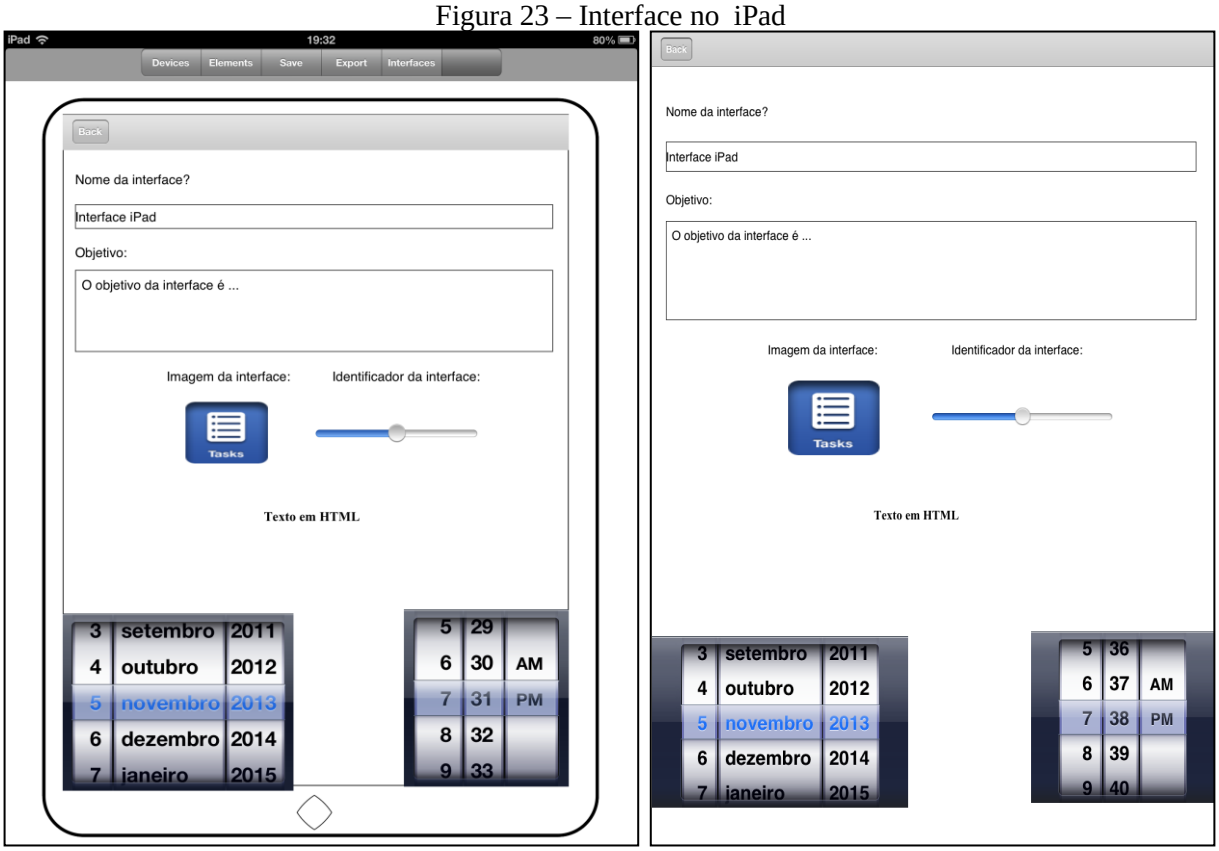


Novamente a interface gerada se manteve fiel à criada na ferramenta. O único ponto a ser mencionado é o horário que é exibido no componente `Time Picker`. Observa-se que o horário exibido difere de uma interface para a outra. Isto se deve ao fato de não ter sido implementada uma propriedade que armazene o valor do horário. O horário exibido é aquele do momento da execução da aplicação. Os elementos utilizados para esta interface são os seguintes: `Search Bar`, `Label`, `TextArea`, `Slider` e `Time Picker`. O código JavaScript correspondente à interface exibida anteriormente pode ser visualizado no Apêndice C.

A Figura 23 exibe o terceiro e último teste realizado no dispositivo iPad. Nas interfaces geradas para o iPad houve uma ligeira perda de precisão no posicionamento dos elementos. Pode ser observado que tanto o `Data Picker` como o `Time Picker` não foram posicionados no final da tela conforme foram criados inicialmente na ferramenta. Os demais elementos também não estão exatamente no lugar onde foram criados. Estas variações no

posicionamento dos elementos se devem ao fato que na ferramenta o tamanho disponível para a criação de interfaces para o iPad não representa o tamanho real da tela deste dispositivo. Não foi possível representar o tamanho real da tela do iPad por conta do menu superior da ferramenta e pelo desenho do dispositivo. Foi utilizado 80% do tamanho da tela do iPad, o que ocasiona que no código gerado para esta plataforma algumas propriedades dos elementos não tenham os valores mais acertados.

Figura 23 – Interface no iPad



No Apêndice D pode ser visualizado o código JavaScript da interface exibida anteriormente.

3.4.1 Comparativo dos trabalhos correlatos

Para uma melhor compreensão das funcionalidades implementadas neste trabalho em relação às funcionalidades existentes nos trabalhos correlatos, foi elaborado o Quadro 19. O presente trabalho é apresentado como Ramirez, os demais conforme descrito na seção 2.4.

Quadro 19 – Comparativo de funcionalidades entre os trabalhos correlatos

funcionalidades / ferramenta	Ramirez	Interface 2	jQuery Mobile	ForguedUI
geração de código para Android	NÃO	NÃO	SIM	SIM
geração de código para iOS	SIM	SIM	SIM	SIM
geração de código para Titanium Studio	SIM	NÃO	NÃO	SIM
geração de código para Xcode	NÃO	SIM	NÃO	NÃO
ferramenta disponível em ambiente *desktop*	NÃO	NÃO	SIM	SIM
ferramenta disponível para Android	NÃO	NÃO	SIM	NÃO
ferramenta disponível para iOS	SIM	SIM	SIM	NÃO
persistência das interfaces	SIM	SIM	SIM	SIM

O foco principal de todas as ferramentas é a geração de código para interfaces gráficas. Percebe-se que todos possuem suporte para geração de código para a plataforma iOS. Porém, somente duas suportam a geração de código para Android, o jQuery Mobile e o ForgedUI, e somente a ferramenta desenvolvida no presente trabalho e o ForgedUI são capazes de gerar o código em JavaScript para o ambiente do Titanium Studio. Isto porque o Interface 2 tem como objetivo gerar código das interfaces na linguagem Objective C para o ambiente do Xcode e o jQuery Mobile em HTML 5 com o intuito de alcançar a maior quantidade de dispositivos possíveis.

Sobre a disponibilidade das ferramentas nas diversas plataformas, observa-se que: o único que não é está disponível no iOS é o ForgedUI; somente o jQuery Mobile pode ser utilizado no Android; o ForgedUI e o jQuery mobile estão disponíveis no ambiente *desktop*.

A persistência das interfaces é implementada em todos os trabalhos pois é uma funcionalidade indespensável na criação das mesmas. Sem a persistência, cada interface teria que ser recriada cada vez que fosse necessária sua edição.

4 CONCLUSÕES

Diante da ausência de uma ferramenta que permitisse a prototipação de interfaces gráficas e a geração de código para o *framework* Titanium nos próprios dispositivos móveis, implementou-se a ferramenta descrita neste trabalho com a finalidade de suprir essa necessidade dos desenvolvedores da comunidade Titanium Studio. A ferramenta apresenta-se como uma opção para agilizar o desenvolvimento de aplicações uma vez que permite criar interfaces gráficas de uma maneira mais simples comparada à tradicional a qual envolve centenas de linhas de programação por parte do desenvolvedor no ambiente do Titanium Studio.

A ferramenta desenvolvida neste trabalho atingiu todos os objetivos propostos, porém com algumas limitações nas interfaces geradas para o iPad, já que estas apresentaram falta de precisão no posicionamento dos elementos gerados. Certos elementos gráficos importantes na plataforma do iOS acabaram não sendo contemplados, como por exemplo: `Table view`, `Toolbar`, `Popover`, `Split view`, entre outros. A maior dificuldade encontrada no desenvolvimento da ferramenta foi o carregamento das interfaces salvas no banco de dados, pois, além da geração do código, foi necessário executar o código gerado para efetuar a inclusão dos elementos no editor da aplicação.

Por fim, o *framework* Titanium mostrou-se bastante útil. A metodologia e a padronização utilizadas no desenvolvimento nesse ambiente justificam sua utilização. Sua documentação é ampla e a comunidade de desenvolvedores que o utilizam está em crescimento, comprovando que sua base é sólida. Esse ambiente possui um SDK fortemente integrado ao Eclipse IDE, utilizando toda a sua estrutura de componentes. Essa integração facilitou consideravelmente o desenvolvimento da ferramenta, assim como também os testes efetuados nos dispositivos e emuladores das plataformas utilizadas.

4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho, tem-se:

a) aumentar a quantidade de elementos gráficos suportados;
b) aumentar a quantidade de propriedades gráficas suportadas;
c) ter suporte a nova versão do iOS 7.0;
d) gerar código para o iPad mini;
e) melhorar a precisão do posicionamento dos elementos na geração do código para o iPad;

f) disponibilizar a ferramenta no iPhone e/ou iPad mini;
g) gerar o código das interfaces na linguagem nativa do iOS, o Objective C;
h) permitir a ligação entre determinadas interfaces, criando a ideia de um projeto;
i) disponibilizar a criação de eventos para os elementos gráficos;
j) permitir aumentar e diminuir o tamanho dos elementos através do toque e movimento dos dedos, ao invés de alterar em uma tela separada.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSON, John. **AppCelerator Titanium:** up and running. Sebastopol: O'Reilly Media, 2013.

APACHE SOFTWARE FOUNDATION. **Apache license**: version 2.0. [S.l.], 2004. Disponível em: <http://www.apache.org/licenses/LICENSE-2.0>. Acesso em: 6 out. 2013.

APPCELERATOR**. Studio architeture**. [S.l.], 2012a. Disponível em: <http://docs.appcelerator.com/titanium/latest/#!/guide/Studio_Architeture>. Acesso em: 06 out. 2013.

_____. **Titanium mobile overview**. [S.l.], 2012b. Disponível em: <http://docs.appcelerator.com/titanium/latest/#!/guide/Titanium_Mobile_Overview>. Acesso em: 06 out. 2013.

_____. **Alloy**. [S.1.], 2013a. Disponível em: <http://www.appcelerator.com/platform/alloy>. Acesso em: 06 out. 2013.

_____. **Titanium Studio:** a modern IDE that enables you to rapidly build mobile applications. [S.l.], 2013b. Disponível em: <http://www.appcelerator.com/platform/titanium-studio>. Acesso em: 30 mar. 2013.

APPLE. **iOS developer library:** learn about iOS 6. [S.l.], 2012a. Disponível em: <http://developer.apple.com/technologies/ios6/>. Acesso em: 30 mar. 2013.

_____. **iOS Human interface guidelines**. [S.l.], 2012b. Disponível em: <http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/Mobileg/Introduction/Introduction.html>. Acesso em: 08 abr. 2013.

CHARLAND, Andre; LEROUX, Brian. **Mobile application development**: web vs. native. [S.l.], 2011. Disponível em: <http://queue.acm.org/detail.cfm?id=1968203/>. Acesso em: 07 abr. 2013.

GOLDSTEIN, Neal; BOVE, Tony. **iPad application development for dummies.** 2nd ed. Hoboken: Wiley, 2011.

INTERNATIONAL DATA CORPORATION BRASIL. **Brasil cresce 127% e entra na lista dos 10 maiores mercados de tablets do mundo, revela estudo da IDC**. [São Paulo], 2012. Disponível em: <http://br.idclatin.com/releases/news.aspx?id=1439>. Acesso em: 23 mar. 2013.

KHANAPP. **KhanApp:** mobile app for Khan Academy. [S.l.], 2010. Disponível em: <http://khanapp.com/>. Acesso em: 03 abr. 2013.

LESS CODE LIMITED. **Interface 2:** advanced mockup for iOS. Nova Zelândia, 2013. Disponível em: <http://interface2.lesscode.co.nz//>. Acesso em: 07 abr. 2013.

MYERS, Brad A. et al. Taking handheld devices to the next level. **IEEE Computer Society**, [S.l.], v. 37, n. 12, p. 36-43, 2004. Disponível em: <http://www.cs.cmu.edu/~pebbles/papers/pebblesControlIEEE.pdf>. Acesso em: 23 mar. 2013.

PAES, Cristina; MOREIRA, Fernando. Dispositivos móveis: estratégia de gestão dos dispositivos na sala de aula e o toolkit do professor. **Revista da Faculdade de Ciência e Tecnologia**, Porto, n. 4, p. 48-57, 2007. Disponível em: <http://bdigital.ufp.pt/bitstream/10284/398/1/48-57.pdf>. Acesso em: 23 mar. 2013.

RIBEIRO, Francisco**. Uma abordagem comparativa do desenvolvimento de aplicações para dispositivos móveis**. [S.l.], 2011. Disponível em: <http://www.cin.ufpe.br/~tg/2011-1/finr.doc>. Acesso em: 21 set. 2013.

STOWELL, Sharry. **ForgedUI:** drag n drop for Titanium. [S.l.], 2011. Disponível em: <http://www.learningtitanium.com/software/forged-ui/drag-n-drop-for-titanium.html>. Acesso em: 07 abr. 2013.

TAFT, Darryl K. **jQuery Mobile set to democratize mobile app design**. [S.l.], 2010. Disponível em: <http://www.eweek.com/c/a/Application-Development/jQuery-Mobile-Set-to-Democratize-Mobile-App-Design-615229/>. Acesso em: 03 abr. 2013.

_____. **AppCelerator releases Titanium Studio IDE for mobile, desktop and web development**. [S.l.], 2011. Disponível em: <http://www.eweek.com/c/a/Application-Development/Appcelerator-Releases-Titanium-Studio-IDE-for-Mobile-Desktop-and-Web-Development-869474/>. Acesso em: 30 mar. 2013.

_____. **Mobile app development**: web or native, that is the question. [S.l.], 2012. Disponível em: <http://www.eweek.com/c/a/Application-Development/Mobile-App-Development-Web-or-Native-That-Is-the-Question-880358/>. Acesso em: 03 abr. 2013.

APÊNDICE A – Código JavaScript resultante da interface no iPhone 4

O código JavaScript gerado correspondente à interface exibida na Figura 21 pode ser visualizado no Quadro 20.

Quadro 20 – Código JavaScript resultante da interface no iPhone 4

```
var window = Ti.UI.createWindow({ backgroundColor : '#fff' ,
                                  fullscreen : true
                                });

var ele_buttonBar1 = Ti.UI.createButtonBar({ labels :["Back"],
                                             backgroundColor: "#ccc"
                                           });

var navgroup1 = Ti.UI.iOS.createToolbar({height : 50,
                                         width : 320,
                                         top : 0,
                                         left : 0,
                                         barColor : '#ccc',
                                         items : [ele_buttonBar1] ,
                                         color:  '#fff'
                                         zIndex : 3
                                        });
window.add(navgroup1);

var label1 = Ti.UI.createLabel({height : 20,
                                width : 200,
                                top : 65,
                                left : 10,
                                text : 'Nome interface:',
                                font: { fontFamily: 'Helvetica' ,
                                        fontSize: 16
                                      } ,
                                zIndex : 3
                               });
window.add(label1);

var textfield1 = Ti.UI.createTextField({height : 30,
                                        width : 300,
                                        top : 103,
                                        left : 10,
                                        value : ' Interface nro 1.',
                                        borderColor : '#333',
                                        font: { fontFamily: 'Helvetica' ,
                                                fontSize: 16
                                              },
                                        zIndex : 3
                                       });
window.add(textfield1);

var pickerdate1 = Ti.UI.createPicker ({height : 210,
                                       width : 320,
                                       top : 270,
                                       left : 0,
                                       type : Ti.UI.PICKER_TYPE_DATE,
                                       locale : 'en',
                                       zIndex : 3
                                      });
window.add(pickerdate1);

var button1 = Ti.UI.createButton({height : 40,
                                  width : 100,
```

Quadro 20 – Código JavaScript resultante da interface no iPhone 4 (continuação)

```
57                                      top : 212,
58                                      left : 111,
59                                      title : 'Salvar',
60                                      color : '#000',
61                                      font: { fontFamily: 'Helvetica' ,
62                                              fontSize: 16
63                                            },
64                                      zIndex : 3
65                                    });
66  window.add(button1);
67
68  var label3 = Ti.UI.createLabel({height : 40,
69                                    width : 200,
70                                    top : 149,
71                                    left : 10,
72                                    text : 'Tela principal do sistema?',
73                                    font: { fontFamily: 'Helvetica' ,
74                                            fontSize: 16
75                                          },
76                                    zIndex : 3
77                                  });
78  window.add(label3);
79
80  var switch1 = Ti.UI.createSwitch({height : 25,
81                                      width : 75,
82                                      top : 156,
83                                      left : 221,
84                                      value : 'true',
85                                      zIndex : 3
86                                    });
87  window.add(switch1);
88
89  window.open();
```

APÊNDICE B – Código XML resultante da interface no iPhone 4

O código XML gerado correspondente à interface exibida na Figura 21 pode ser visualizado no Quadro 21.

Quadro 21 – Código XML resultante da interface no iPhone 4

```
<Alloy>
  <Window class="container">
    <NavGroup height="50" width="320" id="1" top="0" left="0" barColor="#ccc"/>
    <Label height="20" width="200" id="1" top="65" left="10"
     text="Nome   interface:"/>
    <TextField height="30" width="300" id="1" top="103" left="10"
     value=" Interface nro 1." borderColor="#333"/>
    <PickerDate height="210" width="320" id="1" top="270" left="0"
     type="Ti.UI.PICKER_TYPE_DATE" locale="en"/>
    <Button height="40" width="100" id="1" top="212" left="111" title="Salvar"
     color="#000"/>
    <Label height="40" width="200" id="3" top="149" left="10" 12
     text="Tela principal do sistema?"/>
    <Switch height="25" width="75" id="1" top="156" left="221"
     value="true"/>
  </Window>
</Alloy>
```

APÊNDICE C – Código JavaScript resultante da interface no iPhone 5

O código JavaScript correspondente à interface exibida na Figura 22 pode ser visualizado no Quadro 22.

Quadro 22 – Código JavaScript resultante da interface no iPhone 5

```
var window = Ti.UI.createWindow({ backgroundColor : '#fff' ,
                                    fullscreen : true
                                  });

var pickertime1 = Ti.UI.createPicker({height : 215,
                                      width : 320,
                                      top : 353,
                                      left : 0,
                                      type : Ti.UI.PICKER_TYPE_TIME,
                                      locale : 'en',
                                      zIndex : 3
                                     });
window.add(pickertime1);

var searchbar1 = Ti.UI.createSearchBar({height : 43,
                                        width : 320,
                                        top : 0,
                                        left : 0,
                                        barColor : '#ccc',
                                        zIndex : 3
                                       });
window.add(searchbar1);

var slider1 = Ti.UI.createSlider({height : 25,
                                  width : 150,
                                  top : 305,
                                  left : 146,
                                  min : '0',
                                  max : '100',
                                  value : '50',
                                  zIndex : 3
                                 });
window.add(slider1);

var label1 = Ti.UI.createLabel({height : 40,
                                width : 60,
                                top : 295,
                                left : 44,
                                text : 'Volume:',
                                font: { fontFamily: 'Helvetica' ,
                                        fontSize: 16
                                      },
                                zIndex : 3
                               });
window.add(label1);

var label2 = Ti.UI.createLabel({height : 40,
                                width : 100,
                                top : 67,
                                left : 13,
                                text : 'Campo 1:',
                                font: { fontFamily: 'Helvetica' ,
                                        fontSize: 16
                                      },
                                zIndex : 3
```

Quadro 22 – Código JavaScript resultante da interface no iPhone 5 (continuação)

```
56                                  });
57  window.add(label2);
58
59  var textarea1 = Ti.UI.createTextArea({height : 150,
60                                        width : 300,
61                                        top : 108,
62                                        left : 10,
63                                        value : 'Descrição do campo.',
64                                        borderColor : '#333',
65                                        font: { fontFamily: 'Helvetica' ,
66                                                fontSize: 16
67                                              },
68                                        zIndex:3
69                                       });
70  window.add(textarea1);
71
72  window.open();
```

APÊNDICE D – Código JavaScript resultante da interface no iPad

O código JavaScript correspondente à interface exibida na Figura 23 pode ser visualizado no Quadro 23.

Quadro 23 – Código JavaScript resultante da interface no iPad

```
var window = Ti.UI.createWindow({ backgroundColor : '#fff' ,
                                  fullscreen : true
                                });

var pickertime1 = Ti.UI.createPicker({height : 269,
                                      width : 250,
                                      top : 755,
                                      left : 518,
                                      type : Ti.UI.PICKER_TYPE_TIME,
                                      locale : 'en',
                                      zIndex : 3
                                     });
window.add(pickertime1);

var ele_buttonBar1 = Ti.UI.createButtonBar({ labels :["Back"],
                                             backgroundColor: "#ccc"
                                           });

var navgroup1 = Ti.UI.iOS.createToolbar({height : 63,
                                         width : 768,
                                         top : 0,
                                         left : 0,
                                         barColor : '#ccc',
                                         items : [ele_buttonBar1] ,
                                         zIndex : 3
                                        });
window.add(navgroup1);
var imageview1 = Ti.UI.createImageView({height : 94,
                                        width : 125,
                                        top : 439,
                                        left : 186,
                                        backgroundImage : 'images/tasks.png',
                                        zIndex : 3
                                       });
window.add(imageview1);

var label1 = Ti.UI.createLabel({height : 50,
                                width : 213,
                                top : 75,
                                left : 19,
                                text : 'Nome da interface?',
                                font: { fontFamily: 'Helvetica' ,
                                        fontSize: 16
                                      },
                                zIndex : 3
                               });
window.add(label1);

var textfield1 = Ti.UI.createTextField({height : 38,
                                        width : 725,
                                        top : 138,
                                        left : 19,
                                        value : 'Interface iPad',
                                        borderColor : '#333',
                                        font: { fontFamily: 'Helvetica' ,
                                                fontSize: 16
```

Quadro 23 – Código JavaScript resultante da interface no iPad (continuação)

```
57                                                  },
58                                                zIndex : 3
59                                              });
60  window.add(textfield1);
61
62  var label2 = Ti.UI.createLabel({height : 50,
63                                  width : 125,
64                                  top : 186,
65                                  left : 19,
66                                  text : 'Objetivo:',
67                                  font: { fontFamily: 'Helvetica' ,
68                                          fontSize: 16
69                                        },
70                                  zIndex : 3
71                                });
72  window.add(label2);
73
74  var textarea1 = Ti.UI.createTextArea({height : 125,
75                                        width : 725,
76                                        top : 238,
77                                        left : 19,
78                                        value : 'O objetivo da interface é ...',
79                                        borderColor : '#333',
80                                        font: { fontFamily: 'Helvetica' ,
81                                                fontSize: 16
82                                              },
83                                        zIndex : 3
84                                      });
85  window.add(textarea1);
86
87  var label3 = Ti.UI.createLabel({height : 50,
88                                  width : 250,
89                                  top : 375,
90                                  left : 158,
91                                  text : 'Imagem da interface:',
92                                  font: { fontFamily: 'Helvetica' ,
93                                          fontSize: 16
94                                        },
95                                  zIndex : 3
96                                });
97  window.add(label3);
98
99  var slider1 = Ti.UI.createSlider({height : 31,
100                                    width : 250,
101                                    top : 473,
102                                    left : 381,
103                                    min : '0',
104                                    max : '100',
105                                    value : '50',
106                                    zIndex : 3
107                                  });
108 window.add(slider1);
109
110 var label4 = Ti.UI.createLabel({height : 50,
111                                 width : 250,
112                                 top : 375,
113                                 left : 409,
114                                 text : 'Identificador da interface:',
115                                 font: { fontFamily: 'Helvetica' ,
116                                         fontSize: 16
117                                       },
118                                 zIndex : 3
119                               });
120 window.add(label4);
```

Quadro 23 – Código JavaScript resultante da interface no iPad (continuação)

```
121
122 var webview1 = Ti.UI.createWebView({height : 125,
123                                    width : 250,
124                                    top : 591,
125                                    left : 295,
126                                    html : '<b> Texto em HTML <b>',
127                                    zIndex : 3
128                                   });
129 window.add(webview1);
130
131 var pickerdate1 = Ti.UI.createPicker({height : 263,
132                                        width : 350,
133                                        top : 761,
134                                        left : 0,
135                                        type : Ti.UI.PICKER_TYPE_DATE,
136                                        locale : 'pt',
137                                        zIndex : 3
138                                       });
139 window.add(pickerdate1);
140
141 window.open();
```