PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Framework para desenvolvimento flexível de clientes móveis para a arquitetura ContextNet

Sheriton Rodrigues Valim

PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, Janeiro de 2014

Sheriton Rodrigues Valim

Framework para desenvolvimento flexível de clientes móveis para a arquitetura ContextNet

Relatório Final de Projeto de Conclusão de Curso, apresentado ao curso de **Engenharia de Computação** da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador: Markus Endler

Rio de Janeiro
Janeiro de 2014.

Agradecimentos

Aos meus pais, Ana Maria R. Valim e Georgino B. Valim pelo apoio, carinho e ensinamentos, sem os quais não seria possível chegar até aqui.

Ao professor Markus Endler, pela paciência e revisão atenciosa.

À equipe do LAC PUC-Rio, por terem sido solícitos nas questões relacionadas a ClientLib e à VM ContextNet.

À todos os professores da PUC-Rio com os quais tive o prazer de aprender os conceitos mais importantes para minha formação profissional.

Aos meus amigos da PUC-Rio Rafael Basílio, Vanessa Moura, Thiago Motta, Tallita Souza, André Calfa, Luiz Felipe Marques e todos os outros com os quais tive o prazer de conviver durante esses anos.

À Isabelle Cunha, minha noiva, pelo apoio e incentivo nos momentos mais difíceis durante todo o curso.

Resumo

Valim, Sheriton Rodrigues. Endler, Markus. Framework para desenvolvimento flexível de clientes móveis para a arquitetura ContextNet. Rio de Janeiro, 2014. 44p. Relatório Final de Projeto de Conclusão de Curso – Centro Técnico Científico – CTC, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho tem como objetivo desenvolver um arcabouço que facilite o desenvolvimento de clientes para dispositivos móveis que utilizam os recursos disponíveis na arquitetura ContextNet [1], tais como comunicação textual assíncrona instantânea entre nós fixos e móveis usando os protocolos SDDL [2] e compartilhamento de dados de contexto, fornecidos pelos sensores dos dispositivos móveis, em tempo real.

Palavras-chave

Framework; dispositivos móveis ; comunicação; context.

Abstract

Valim, Sheriton Rodrigues. Endler, Markus. Framework for flexible development of mobile clients for architecture ContextNet. Rio de Janeiro, 2014. 44p. Relatório Final de Projeto de Conclusão de Curso – Centro Técnico Científico – CTC, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This work aims to develop a framework that facilitates the development of clients for mobile devices who use the resources available on the architecture ContextNet [1], such as asynchronous instant text communication between fixed and mobile nodes using SDDL [2] protocols and sharing context data, provided by device mobile's sensors, in real time.

Keywords

Framework; mobile devices; communication; context.

Sumário

1 Introdução

A motivação deste projeto veio de requisitos de um projeto em desenvolvimento no LAC (Laboratory for Advanced Collaboration) da PUC-Rio, o ContextNet [1].

O projeto ContextNet tem como objetivo o desenvolvimento de uma arquitetura de middleware que forneça serviços de contexto para aplicações colaborativas de larga escala. Tais aplicações podem ser serviços de monitoramento e/ou coordenação on-line de atividades de entidades móveis, ou seja, usuários com *smartphones* ou *tablets*, veículos ou até mesmo robôs móveis autônomos.

O ContextNet é indicado para qualquer aplicação onde haja necessidade de compartilhamento de dados entre um conjunto possivelmente grande de nós móveis, alguns exemplos de aplicações desse tipo são:

- monitoramento e/ou coordenação de frotas de veículos;
- coordenação de equipes de emergência;
- logística.

A arquitetura ContextNet foi projetada em camadas com serviços bem definidos, sendo a mais básica dessas camadas o middleware **Scalable Data Distribution Layer (SDDL)** [2]. Essa camada é responsável pela comunicação entre nós móveis e/ou fixos da aplicação e para isso utiliza os protocolos Data Distribution Service for Real-Time Systems (DDS) [4] para a comunicação entre os nós fixos e o Mobile Reliable-UDP (MR-UDP) [5] para os nós móveis. Além de prover comunicação ponto a ponto entre quaisquer dois nós (fixos ou móveis), o SDDL também possui serviços de broadcast e ou groupcast, isto é, envio simultâneo para todos os nós ou para um grupo de nós. Grupo esse, que inclusive pode ser definido dinamicamente, a partir da igualdade, ou proximidade, de algum dado de contexto. Assim, consegue-se, por exemplo, entregar uma mesma mensagem a todos os nós móveis que estejam co-localizados em uma região geográfica.

Uma aplicação distribuída ContextNet essencialmente consiste em software cliente, que executa no nó móvel, e software servidor, que executa em um ou mais nós do SDDL core. Devido à alta vazão de dados e baixa latência da comunicação no SDDL core, qualquer serviço de processamento de dados da aplicação pode ser naturalmente replicado em vários servidores no SDDL core, tornando assim as aplicações naturalmente escaláveis.

Sendo assim, podemos dividir uma aplicação ContextNet, de maneira

simplificada, em três componentes independentes, como descrito abaixo:

- **SDDL:** O middleware de comunicação e compartilhamento de contexto do ContextNet;
- **Nós fixos:** Sistemas interligados ao middleware SDDL e executados em servidores, desktops e/ou notebooks;
- **Nós móveis:** Sistemas interligados ao middleware SDDL e executados em Smartphones e/ou Tablets[1].

Para um cliente móvel se conectar ao middleware SDDL é necessária a utilização de uma biblioteca chamada **ClientLib** que disponibiliza uma API para a conexão e troca de dados, de forma assíncrona, com outros elementos da Aplicação ContextNet. A ClientLib abstrai o protocolo de comunicação entre os clientes móveis e os servidores que compõem o SDDL core (gateways), faz a gerência de handovers de forma transparente ao cliente móvel, fornece comunicação assíncrona e sua utilização é simples.

Com base no ContextNet, a equipe do LAC desenvolveu o Controlador ARFF [3], um Web-Applet com função de exemplificar funcionalidades de uma central de controle e monitoramento de fiscais e frotas de veículos. Esse applet tem a capacidade de compartilhar dados e de se comunicar textualmente com nós móveis através do SDDL. O ARFF tem como objetivo principal o acompanhamento do deslocamento de nós móveis, executando o aplicativo ContextNet-Mobile [6] em um mapa.

O ContextNet-Mobile foi desenvolvido para a plataforma Android, de forma a utilizar os recursos da ClientLib para estabelecer comunicação com o controlador ARFF. Esse aplicativo tem funcionalidades para comunicação instantânea com outros nós móveis e com o Controlador para fins de compartilhamento de sua localização e outras informações de contexto. Essas informações são obtidas através da API geolocalização disponibilizada pelo Google e dos dados obtidos de sensores do dispositivo onde a aplicação está instalada.

1.1 Objetivo

A partir da aplicação ContextNet-Mobile, notou-se a necessidade de um framework[2] que facilite o desenvolvimento de aplicativos que utilizem a

1 Veículos e robôs autônomos também podem ser considerados nós móveis, porém, o framework aqui descrito não tem como foco o desenvolvimento de sistemas para esses dispositivos.

2 Um framework é um conjunto de estruturas reutilizáveis, organizadas para auxiliar o desenvolvimento de outros projetos de forma rápida, fácil e eficiente.

arquitetura ContextNet para dispositivos móveis de maneira rápida, flexível[3] e para múltiplas plataformas mobile. Sendo assim, o principal objetivo do presente trabalho foi o desenvolvimento desse framework. Mais detalhes na sessão 3.

Para a execução desse projeto, foi necessária a aplicação de muitos dos conceitos aprendidos nas diversas disciplinas do curso, como encapsulamento, modularização, orientação a objetos, concorrência e sistemas distribuídos.

2 Estado da Arte

Como descrito na de introdução, o ContextNet é um middleware desenvolvido pelo Laboratory for Advanced Collaboration (LAC), na PUC-Rio, e portanto, até a data de desenvolvimento deste trabalho não existe um framework para desenvolvimento de clientes móveis especificamente para essa arquitetura.

O ponto de partida para o desenvolvimento deste framework foi dado com base no aplicativo ContextNet-Mobile e em estudos preliminares de tecnologias que permitem o desenvolvimento de aplicativos para múltiplas plataformas.

2.1 ContextNet-Mobile

O ContextNet-Mobile, como descrito na introdução, foi desenvolvido para a plataforma Android com o objetivo de compor a aplicação ARFF de forma a exemplificar as funcionalidades do ContextNet.

Esse aplicativo utiliza o protocolo SDDL para comunicação assíncrona instantânea para compartilhamento de dados de contexto obtidos pelos sensores do dispositivo móvel, além de opções para compartilhamento de **Pontos de Interesse (POIs)**, informações obtidas a partir dos sensores do dispositivo, comunicação textual com nós fixos e móveis e interpretação de arquivos descritores de formulários para acompanhamento de inspeção com um controlador em tempo real.

3 Entende-se por flexível a capacidade de desenvolver aplicações para diversos fins, utilizando o mesmo framework.

Figura 1 - Tela principal da aplicação ContextNet – Mobile

Na tela principal, o usuário pode ver o posicionamento de veículos e pontos de interesse e tem acesso a funções para *Compartilhar* informações sobre acidentes, obras ou obstruções na via, além de *Chat* para comunicação instantânea com a central de monitoramento (controlador) e/ou com condutores de outros veículos na rede ContextNet. A interface gráfica do ContextNet-Mobile foi pensada de forma que o usuário precise dispensar a menor atenção possível para o manuseio do aplicativo.

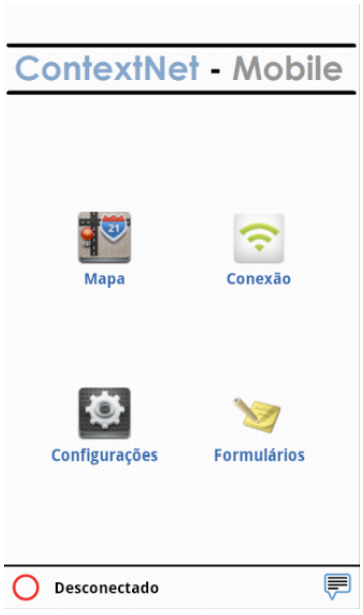


Figura 2 - Tela de menu do aplicativo ContextNet – Mobile

A partir da tela principal o usuário pode ativar o menu da aplicação, como mostra a Figura 2. Nesta tela, o usuário pode ver o status da conexão e ter acesso a configurações gerais ou de conexão, voltar para a tela principal

(Mapa), descrita anteriormente, e tem acesso a formulários de inspeção. Esses formulários são utilizados pelos fiscais, usuários do sistema ContextNet – Mobile, para enviar e receber informações sobre inspeções de veículos.

Os aplicativos a serem gerados com o Framework proposto poderão ter as mesmas funcionalidades do *ContextNet-Mobile*. Portanto, esse aplicativo será utilizado como benchmark para fins de comparação com aplicativos criados a partir do framework proposto.

2.2 ClientLib

A ClientLib é a biblioteca que permite que uma aplicação móvel tenha acesso a SDDL. Portanto, é de grande importância para o desenvolvimento do framework aqui proposto.

Essa biblioteca contém 3 interfaces principais necessárias para interagir com a SDDL. São elas:

2.2.1 NodeConnection

A interface NodeConnection contém todos os métodos necessários para se conectar, desconectar, enviar mensagens e incluir listeners que escutam eventos da SDDL que tenham relevância para o objeto que implementa NodeConnection.

A biblioteca ClientLib possui uma fábrica de NodeConnections que fornece implementações de NodeConnection com os principais protocolos suportados, sendo o mais usado deles o MrUdpNodeConnection. Porém, caso o usuário assim desejar, é possível fornecer uma implementação própria da NodeConnection, desde que o protocolo utilizado seja suportado pela SDDL.

2.2.2 NodeConnectionListener

Essa interface contém todos os métodos de callback utilizados pela SDDL para informar ao NodeConnection sobre eventos ocorridos, conexão estabelecida / perdida, recebimento de novas mensagens, etc.

2.2.3 Message

Por fim, essa interface deve ser implementada por qualquer objeto serializável que o usuário queira enviar ou receber nos NodeConnections através de mensagens. É nessa interface que devem ser encapsuladas as informações que serão trafegadas pelo middleware SDDL.

Até o momento em que esse projeto foi iniciado existia apenas, uma versão da ClientLib em Java, o que inviabilizava o desenvolvimento de aplicativos móveis para o ContextNet que pudessem ser executados em plataformas que não tenham suporte a Java, como o iOS por exemplo.

No entanto, no decorrer do projeto foi desenvolvida uma versão da ClientLib em Lua. Porém, não houve tempo suficiente para que se pudesse desenvolver um módulo dessa biblioteca de forma a integra-la ao framework.

3 Objetivos do trabalho

O objetivo principal deste trabalho foi projetar e desenvolver um framework que possibilite o desenvolvimento rápido, flexível e para múltiplas plataformas móveis, de aplicativos que utilizem os recursos da arquitetura ContextNet. Assim, parte do objetivo foi a implementação de funcionalidades disponíveis no aplicativo ContextNet-Mobile de forma a torna-las genéricas, para que o usuário final deste framework possa utiliza-lo para desenvolver aplicativos com funcionalidades semelhantes às encontradas no ContextNet-Mobile e, ao mesmo tempo, respeitando todas as regras estabelecidas para o próprio sistema em desenvolvimento.

Esse framework foi desenvolvido com a preocupação em manter o máximo de flexibilidade possível de forma que o usuário pudesse estendê-lo para melhor atender as necessidades do aplicativo que deseja implementar, permitindo assim incluir novas funcionalidades não pensadas num primeiro momento, ou melhorar as funcionalidades disponíveis. Além disso, foi decidido junto ao orientador que o framework fosse disponibilizado em forma de código aberto. Assim, o usuário que vier a utiliza-lo deverá incluir os fontes em seu projeto e alterar os trechos que achar necessário para o desenvolvimento do seu aplicativo.

Alguns trechos de código deverão ser alterados para a inclusão de objetos pertencentes à regra do negócio do aplicativo em desenvolvimento. Mais detalhes sobre isso na sessão 5.3 e Apêndice B.

4 Atividades Realizadas

Desde o início da realização deste projeto, foram estudados conteúdos conceituais e técnicos, de modo a formar uma base sólida para o início do desenvolvimento do framework. A maior parte desses estudos foram iniciados ainda durante o Projeto Final I (primeira parte desta disciplina) e continuaram até os últimos dias de desenvolvimento do projeto.

4.1 Estudos sobre a arquitetura ContextNet

No início do projeto, o Prof. Markus Endler sugeriu a leitura de alguns

artigos e publicações de autoria sua e de sua equipe do LAC. Também foi sugerido o estudo do código da aplicação ContextNet-Mobile[6], uma vez que o framework proposto deveria ser focado no desenvolvimento de aplicações semelhantes a essa.

Estes documentos foram essenciais para o melhor entendimento da arquitetura ContextNet e de sua utilização. A aplicação ContextNet-Mobile foi muito importante para exemplificar como uma aplicação mobile deve se comunicar com a rede SDDL. Além de estabelecer alguns parâmetros que, mais tarde, puderam ser utilizados para comparar e validar os resultados obtidos por uma aplicação desenvolvida com base no framework proposto.

4.2 Estudo e Definição de Tecnologias a Serem Utilizadas

Para a realização do projeto, foi necessário estudar as atuais tecnologias de desenvolvimento capazes de gerar aplicações móveis multi-plaforma, uma vez que esse era um dos objetivos. Mesmo que a ClientLib disponível no início do projeto não pudesse ser executada em ambientes que não suportam Java, uma das preocupações foi que o ambiente escolhido permitisse que o framework desenvolvido fosse expandido para incluir uma segunda versão da ClientLib escrita em uma outra linguagem, que não Java, que pudesse ser integrada sem grande dificuldade aos demais componentes do framework proposto.

Os principais Frameworks estudados foram Corona SDK [7], PhoneGap [8] e Titanium SDK [9]. No entanto, excluímos o Corona SDK por ser uma tecnologia paga.

Os frameworks PhoneGap e Titanium SDK têm algumas similaridades em relação ao objetivo final: disponibilizar ferramentas para o desenvolvimento de aplicações móveis *cross-platform*. Ambas plataformas são distribuídas sob a licença Apache, versão 2.0, que garante permissão perpétua para utilizar, reproduzir, desenvolver trabalhos derivados e distribuir o trabalho e/ou os trabalhos derivados que venham a ser desenvolvidos, em fonte ou em forma de objetos.

Entretanto, a proposta do PhoneGap é possibilitar o desenvolvimento de aplicações, baseadas em HTML5, CSS e JavaScript, para dispositivos móveis, gerando um aplicativo web que pode ser instalado nas plataformas Android ou iOS como um aplicativo nativo. A diferença entre as aplicações desenvolvidas com o PhoneGap e uma aplicação puramente web, é o uso de uma API que permite a utilização, por parte dessas aplicações, de recursos exclusivos dos

dispositivos móveis, tais como câmera, lista de contatos e sensores.

O Titanium SDK, por sua vez, faz uso, apenas, de JavaScript para o desenvolvimento de suas aplicações que são compiladas para uma plataforma específica gerando, então, uma aplicação nativa. Dessa forma, os aplicativos desenvolvidos pelo Titanium SDK têm desempenho muito próximo dos aplicativos desenvolvidos pelas ferramentas nativas dos sistemas operacionais dos dispositivos móveis. Esse foi o principal motivo para a escolha desse SDK para o desenvolvimento do presente projeto.

Além disso, o Titanium SDK fornece recursos que permitem ao usuário estendê-lo criando seus próprios módulos para plataformas nativas e/ou para múltiplas plataformas (CommonJS). Os módulos CommonJS podem utilizar os próprios módulos de interface gráfica fornecidos pela API do Titanium SDK. Com isso, é possível desenvolver certas funcionalidades em módulos que podem ser reaproveitados em outras aplicações. Esse recurso foi muito útil para o desenvolvimento do framework para o ContextNet.

4.3 Estudo do Aplicativo ContextNet-Mobile

Como já foi brevemente descrito, o ContextNet-Mobile é o principal aplicativo utilizado como exemplo para o desenvolvimento deste projeto. Foi a partir desse aplicativo que foram decididos quais seriam as principais funcionalidades que deveriam ser implementadas no framework.

O ContextNet-Mobile foi um aplicativo desenvolvido para Android, utilizando as ferramentas fornecidas pelo Google. Na versão utilizada como base para o presente projeto, o ContextNet-Mobile possuía sete activities[4] principais:

- ChatActivity;
- ConfigScreen;
- ConnectionScreen;
- LogActivity;
- MapScreen;
- MenuScreen;
- SplashScreen.

A SplashScreen é a MainActivity (activity que inicializa o aplicativo). Ela é a responsável por inicializar parâmetros e estruturas necessárias para o aplicativo. Logo após inicializar esses dados, a SplashScreen inicia a MapScreen que é, de fato, a activity principal da aplicação.

Uma Activity pode ser destruída a qualquer momento pelo SO Android

4 De forma simplificada podemos entender uma activity como um objeto que representa uma única tela em Android.
Mais detalhes em http://developer.android.com/reference/android/app/Activity.html.

desde que esteja no estado *stop*. Isso é necessário para o melhor gerenciamento de memória do dispositivo, já que o sistema Android foi idealizado para dispositivos com menos recursos que um PC comum. Portanto, o tratamento de eventos recebidos através do middleware SDDL não poderia ser feito em nenhuma Activity. Do contrário, só poderíamos garantir que o usuário receberá uma mensagem, caso o usuário estiver sempre utilizando a Activity responsável pelo tratamento dessa mensagem. Para evitar isso, existe a classe App que estende a classe Application, que é a classe base para quem precisa manter objetos globais da aplicação. Na classe App são inicializados todos os objetos que precisam existir durante todo o contexto da aplicação, como o objeto de conexão, handler de mensagens, controlador de mapa, entre outros. Além disso, a classe App possui métodos que permitem que as Activities alterem determinados estados da aplicação que devem ser refletidos em todas as outras Activities como, por exemplo, alterar o estado de conexão do aplicativo.

O controlador de mapa ou MapController é o objeto responsável por executar todas as alterações no mapa exibido na MapScreen. É através dela que podemos incluir novas notificações e/ou alterar a posição de outros objetos representados no mapa.

Além disso, a aplicação ContextNet-Mobile possui um conjunto de classes que implementam AsyncTask[5]. Essas classes são utilizadas para executar métodos que envolvem a transmissão de dados através do middleware SDDL, ou seja, a maioria dos métodos de NodeConnection chamados durante a aplicação têm a sua chamada encapsulada em uma AsyncTask. As 3 AsyncTasks mais utilizadas na aplicação são: ConnectionTask, FinalizeConnectionTask e ChatMessageTask.

O estudo do código desse aplicativo foi de extrema importância para evitar erros comuns e também para entender melhor o fluxo de operação correto em uma aplicação desse tipo.

4.4 Estudos sobre a Titanium API

A API disponível no Titanium SDK é composta de uma coleção de módulos que fornecem funcionalidades distintas como: utilização da câmera do dispositivo, utilização de mapa nativo da plataforma, entre outros. Existem vários tipos de módulos na Titanium API, sendo os *Packaged Titanium Modules* os mais importantes para o desenvolvimento do projeto.

5 Uma AsyncTask é uma classe utilizada para realizar uma tarefa assíncrona em Android.

Um *Packaged Module* pode conter tanto código em Javascript quanto na linguagem nativa de uma plataforma específica, o que torna possível a utilização dos recursos específicos de uma determinada plataforma como, por exemplo, uma AsyncTask da plataforma Android.

Além disso, um *Packaged Module* é capaz de exportar, também, métodos de bibliotecas externas. Esse mecanismo se dá pelo uso de *Proxie*s, um objeto escrito em linguagem nativa que exporta uma API em Javascript. Geralmente, uma *Proxy* é utilizada como um empacotador.

A Figura 3 abaixo, contém um exemplo de código de utilização de uma proxy.

```
// 1. Create a proxy object
var win = Ti.UI.createWindow();
// 2. Set a property
win.title = "Hello World";
// 3. Call a method on the proxy.
win.open();
```

Figura 3 - Código exemplificando a utilização de uma proxy (figura extraída da documentação da Titanium Module Concepts [10])

O diagrama abaixo ilustra o que acontece internamente ao executar o código da figura 3.

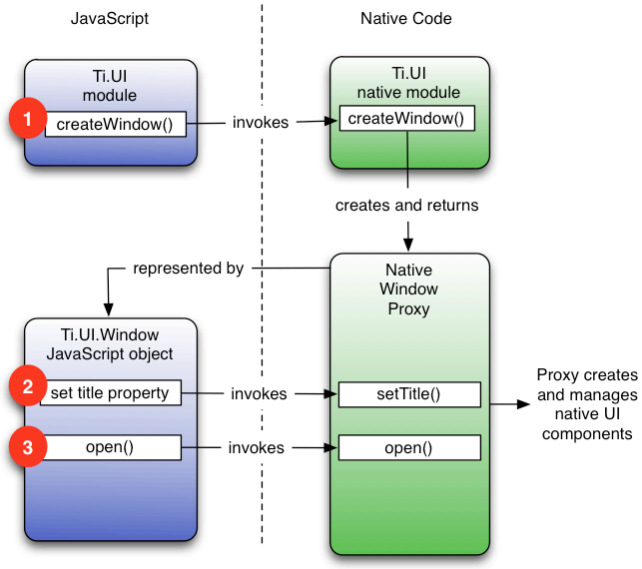


Figura 4 – Diagrama exemplificando funcionamento de uma proxy (figura extraída da documentação da Titanium Module Concepts[10])

Com base nessa característica da API do Titanium SDK, foi desenvolvida uma arquitetura em módulos que estendem o Titanium SDK e, ao mesmo tempo, fornecem alguns dos recursos necessários para um aplicativo do ContextNet. Esses módulos foram divididos em 2 grupos: Módulo clientlib e Módulos Componentes, que serão mais detalhados na sessão 5.1.

4.5 Estudos sobre o Titanium Studio

Para desenvolver uma aplicação que utilize o Titanium SDK, a empresa Appcelerator fornece uma IDE baseada em eclipse chamada Titanium Studio.

Essa IDE contém painéis de configuração que permitem ao usuário integrar Frameworks de diversas plataformas.

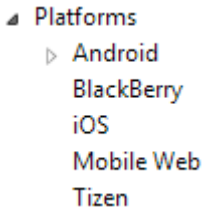


Figura 5 - Plataformas suportadas pelo Titanium Studio

Caso o usuário configure os caminhos de diretórios e variáveis de ambiente para essas plataformas[6], ao escrever o código da aplicação, utilizando somente módulos suportados por todas as plataformas (ou que possuam versões para todas as plataformas), o usuário pode escolher no momento do build, para qual plataforma deseja compilar a aplicação.

4.5.1 Estrutura de uma aplicação no Titanium Studio

A Figura 6 ilustra a estrutura geral de uma aplicação desenvolvida no Titanium Studio. Essa estrutura é composta, principalmente, pelo arquivo tiapp.xml, pelos diretórios Resources, i18n e, eventualmente, modules.

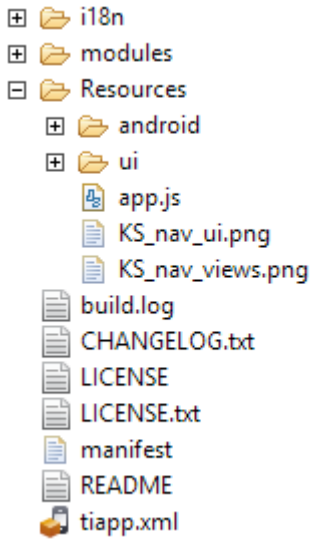


Figura 6 - Estrutura de uma aplicação no Titanium Studio

6 No caso do iOS, só é possível instalar o ambiente de desenvolvimento no sistema operacional Mac OS.

O arquivo **tiapp.xml** é semelhante ao arquivo de manifesto em um projeto Android e serve, inclusive, para incluir *tags* específicas no arquivo de manifesto do Android. Essa inclusão deve ser feita como na figura abaixo.

```
<android xmlns:android="http://schemas.android.com/apk/res/android">
    <manifest android:versionName="1.0.0">
        <activity android:configChanges="keyboardHidden"
            android:name="org.appcelerator.titanium.TiActivity" android:screenOrientation="portrait"/>
    </manifest>
</android>
```

Figura 7 - Inclusão de instruções no arquivo de manifesto do Android

Além disso, é no arquivo **tiapp.xml** que são incluídos os *Packaged Modules* utilizados no desenvolvimento da aplicação. Esse arquivo também possui um editor gráfico, como ilustrado na figura abaixo.

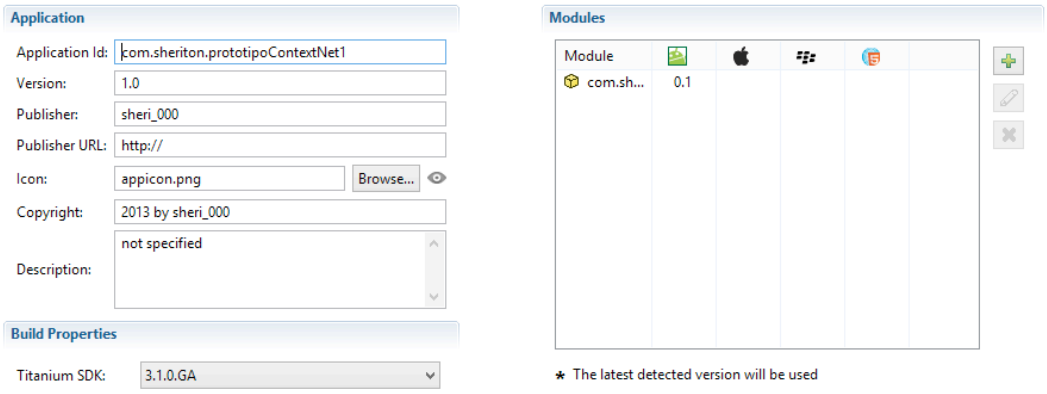


Figura 8 - Editor gráfico do arquivo tiapp.xml

Através desse editor, o usuário pode incluir módulos disponíveis no SDK instalado, escolher a versão do Titanium SDK que deseja utilizar e configurar Id, versão e outras informações sobre a aplicação.

O diretório **Resources** é o diretório principal da aplicação. Nele estão contidos todos os arquivos de código escritos em Javascript. Ao iniciar uma nova aplicação, esse diretório normalmente vem preenchido com um diretório **ui** que contém os códigos das telas e **Views** da aplicação, no formato da aplicação escolhida, e diretórios contendo arquivos específicos para cada plataforma selecionada ao criar a aplicação (Só é possível criar aplicações para plataformas previamente configuradas).

O diretório **i18n** é parte de um sistema de internacionalização dos aplicativos desenvolvidos com o Titanium. Nele existe um subdiretório para cada idioma suportado pela aplicação e, dentro desse subdiretório, existem

arquivos .xml compostos basicamente por strings identificadoras, as respectivas strings que serão exibidas pela aplicação no idioma desse subdiretório. Para uma aplicação desenvolvida para o idioma inglês, por exemplo, teríamos dentro do diretório **i18n** um subdiretório **en** e dentro do subdiretório **en**, um arquivo **strings.xml** (i18n\en\strings.xml). A estrutura desse arquivo é ilustrada na figura abaixo.

```
<resources>
    <string name="welcome">Welcome to %s!</string>
</resources>
```

Figura 9 - Arquivo strings.xml

No código da aplicação a string welcome pode ser utilizada da seguinte forma: *String.format(L('welcome'), 'usuário')*, onde o parâmetro de **L** é a string que idenfica o texto que desejamos exibir e o segundo parâmetro do método **format** é a string que desejamos incluir no lugar de **%s** no arquivo **strings.xml**. Com esse mecanismo o texto exibido na execução da aplicação dependerá do idioma para qual o dispositivo móvel estiver configurado. Caso o idioma do dispositivo não estiver disponível na aplicação, será utilizado o idioma padrão, normalmente, inglês.

4.5.2 Estrutura de um módulo no Titanium Studio

Além de aplicações, o Titanium Studio também é utilizado para o desenvolvimento de *Packaged Modules* para as diversas plataformas suportadas, sendo mais comum encontrar módulos disponíveis para as duas plataformas mais difundidas no mercado, Android e iOS.

A estrutura geral de um *Packaged Modules* é mais semelhante à estrutura de uma aplicação Java no Eclipse. No entanto, é possível desenvolver módulos nas linguagens nativas das plataformas escolhidas. Portanto, caso o usuário esteja desenvolvendo um Módulo para a plataforma iOS, existirá código em Objective-C nessa estrutura. A Figura 10 ilustra a estrutura de um módulo para Android.

O projeto de um módulo para o Titanium SDK também possui um arquivo semelhante ao tiapp.xml, chamado timodule.xml, que tem a mesma função que o tiapp. No entanto, não contém instruções para configuração de módulos de plataformas específicas, uma vez que, diferente de uma aplicação, um *Packaged Module* só pode ser compilado para uma única plataforma.

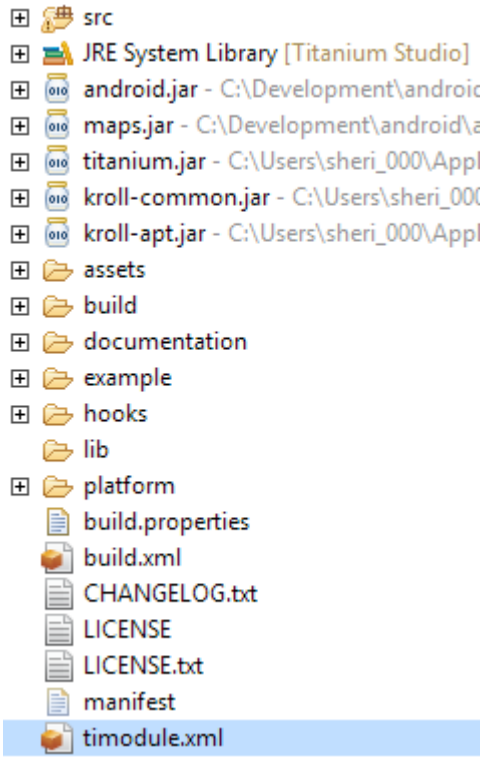


Figura 10 - Estrutura de um módulo no Titanium Studio

Todo o *Packaged Module* tem uma classe principal, que leva o nome do módulo desenvolvido com o sufixo **Module** (p.ex.: FirstmoduleModule), que estende uma classe abstrata específica para módulos da plataforma escolhida (KrollModule para Android e TiModule para iOS). Essa classe é responsável por fornecer os métodos *create* chamados no código da aplicação em Javascript (vide Figura 3). Porém, não é necessário que o usuário crie manualmente esses módulos pois, ao criar uma proxy, deve-se incluir uma anotação @Kroll.proxy(creatableInModule=Modulo.class) indicando qual o módulo responsável por instanciar essa proxy, como pode ser observado na figura abaixo.

```
// This proxy can be created by calling Firstmodule.createExample({message: "hello world"})
@Kroll.proxy(creatableInModule=FirstmoduleModule.class)
public class ExampleProxy extends TiViewProxy
```

Figura 11 - Código exemplificando o protótipo de uma Proxy

Além disso, tanto a classe principal do módulo quanto as proxies podem exportar métodos e propriedades que devem ser acessíveis em Javascript. Em Android, isso deve ser feito usando a anotação @Kroll.method e @Kroll.getProperty e/ou @Kroll.setProperty, como pode ser visualizado no

exemplo abaixo.

```
// Methods
@SuppressWarnings("deprecation")
@Kroll.method
public void printMessage(String message)
{
    Log.d(TAG, "printing message: " + message);
}


@Kroll.getProperty @Kroll.method
public String getMessage()
{
    return "Hello World from my module";
}

@SuppressWarnings("deprecation")
@Kroll.setProperty @Kroll.method
public void setMessage(String message)
{
    Log.d(TAG, "Tried setting module message to: " + message);
}
```

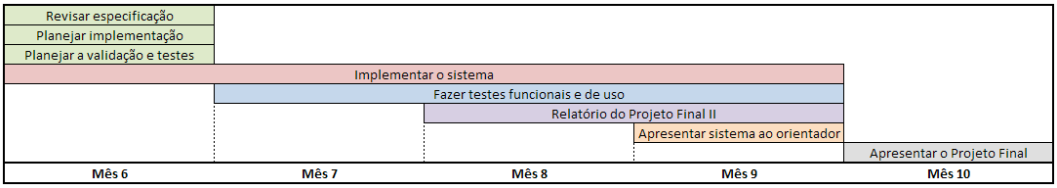
Figura 12 - Exemplo de como exportar métodos e propriedades para Javascript

Dessa forma, ao fazer o comando *Package Module*, o compilador do Titanium SDK irá criar todos os mecanismos necessários para tornar esses métodos acessíveis em Javascript.

4.6 Cronograma

No relatório de Projeto Final I, foi elaborado o seguinte cronograma para a execução do projeto final II.

Tabela 1 - Cronograma de projeto final elaborado durante Projeto Final I

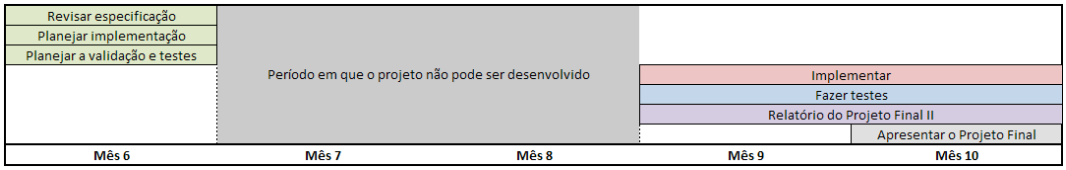


Esse cronograma foi descrito de maneira bastante genérica, já que, na época em que foi escrito, ainda não era possível visualizar com clareza todas as tarefas que deveriam ser executadas até a conclusão do projeto.

O projeto precisou ser interrompido por cerca de dois meses, sendo retomado, somente, no final de novembro. Essa pausa dificultou a implementação e encurtou muito os prazos. Além disso, logo após a retomada do desenvolvimento do projeto, foram observados alguns pontos importantes que levaram a mudanças substanciais no que havia sido planejado como, por exemplo, o uso de bibliotecas externas para encapsular estruturas de dados que seriam trafegadas pelo middleware, o que de certa forma inviabilizaria a ideia inicial de desenvolver as funcionalidades encontradas no ContextNet-Mobile,

uma vez que inevitavelmente será necessária alteração no código para a inclusão dessas bibliotecas externas. Também foram encontradas dificuldades para encapsular a biblioteca ClientLib em um *Packaged Module, o* que levou ao cronograma real apresentado abaixo:

Tabela 2 - Cronograma real de atividades realizadas durante o Projeto Final II



5 Projeto e especificação

5.1 Arquitetura

Como dito anteriormente, aproveitando características da Titanium API, foi idealizada para o framework uma arquitetura em Módulos que, por sua vez, se subdivide em dois grupos, o módulo clientlib e os módulos componentes.

- **Módulo clientlib**

Esse módulo encapsula as classes da biblioteca ClientLib, de forma a exportar seus métodos para Javascript podendo, então, serem utilizados numa aplicação qualquer desenvolvida com o Titanium SDK.

Será necessário implementar uma segunda versão desse módulo, utilizando uma ClientLib suportada pela plataforma iOS.

O Titanium SDK consegue identificar a plataforma para a qual se deseja efetuar o *build*, sendo possível identificar qual desses módulos deverá ser carregado. Portanto, a única restrição para que o aplicativo seja multi-plataforma é que essas versões do módulo de conexão exportem métodos com o mesmo nome, ou seja, a interface deve ser comum.

- **Módulos de componentes**

Esses são os módulos que fornecem esqueletos de código que devem ser adaptados de acordo com a regra de negócio estabelecida para o aplicativo que o usuário deseja desenvolver utilizando este framework.

Nesta categoria se encontram os componentes que normalmente compõem uma aplicação ContextNet como mapas, ferramentas para envio e recebimento de mensagens, utilização de alguns sensores dos dispositivos, etc.

A diferença entre estes componentes e os componentes já fornecidos pelo

Titanium SDK é que os componentes desenvolvidos para compor o framework foram pensados para interagir com os demais, incluindo o módulo clientlib. Essa interação deverá ser feita com base em métodos de callback e eventos disponibilizados nos módulos, assim todos os eventos podem ser tratados por um único objeto responsável por isso. Essa forma de interação entre os objetos foi escolhida por possibilitar que as aplicações pudessem tratar todos os eventos recebidos, independente de qual *view* estiver sendo exibida.

Dessa forma, os módulos de componentes poderão ser reaproveitados em todas as aplicações, exigindo do usuário apenas a customização do layout e a definição do fluxo de ações da aplicação e a correta configuração de um módulo de conexão, além da adaptação do componente para uma alguma estrutura externa que venha a ser utilizada.

Os benefícios de uma arquitetura baseada em módulos são muitos, sendo o mais importante o de possibilitar maior flexibilidade no desenvolvimento de aplicações, que é um dos principais requisitos desse projeto.

Nesta arquitetura o único módulo obrigatório é o módulo de conexão. A utilização dos demais módulos depende apenas das necessidades impostas pelos requisitos do aplicativo que se queira desenvolver utilizando o framework proposto. Ou seja, se o usuário desejar utilizar o framework para desenvolver uma aplicação onde possa visualizar em tempo real o posicionamento de uma frota de veículos, mas não quer a funcionalidade de troca de mensagens instantâneas, basta que o usuário inclua o módulo de conexão devidamente configurado para o seu servidor e inclua, também, um módulo de mapas que utilize essa configuração, sem precisar incluir o módulo de chat.

No início do projeto, acreditava-se que seria possível fornecer implementações fechadas[7] para esses módulos. No entanto, é comum nas implementações de sistemas distribuídos encontrarmos bibliotecas contendo implementações de objetos necessários para a comunicação entre as diversas partes do sistema. Logo, como qualquer sistema que utiliza o ContextNet é naturalmente um sistema distribuído, é natural também que esses sistemas possuam suas próprias restrições de negócios e estruturas para enviar e receber dados.

Daí existem duas abordagens possíveis para o desenvolvimento de uma aplicação:

7 Entende-se por componente com implementação fechada, um componente que fornece funcionalidade completa com suas próprias estruturas de dados. (p.ex. o componente MapView disponível para dispositivos Android)

1. As interfaces são apenas um conjunto de *tags* (p.ex: formato xml) que devem ser incluídas nas mensagens (*strings*) transmitidas através do middleware SDDL e que são interpretadas pelos outros nós da rede ContextNet. (Arquitetura Fraca)

2. As interfaces estão implementadas em uma biblioteca externa conhecida por todos os componentes do sistema e todas as mensagens são encapsuladas em estruturas fornecidas por essa biblioteca. (Arquitetura Forte)

Utilizando ambas as arquiteturas é possível desenvolver um sistema semelhante ao Projeto ARFF e CNet Mobile. No entanto, é fácil perceber que um sistema desenvolvido com base na primeira abordagem seria muito mais complexo e de difícil manutenção do que um sistema baseado na segunda abordagem. Por esse motivo, descrevo a primeira abordagem como fraca e a segunda como forte.

Para facilitar o entendimento, chamaremos de UserLib a biblioteca que contém o conjunto de estruturas pertencentes a regra do negócio.

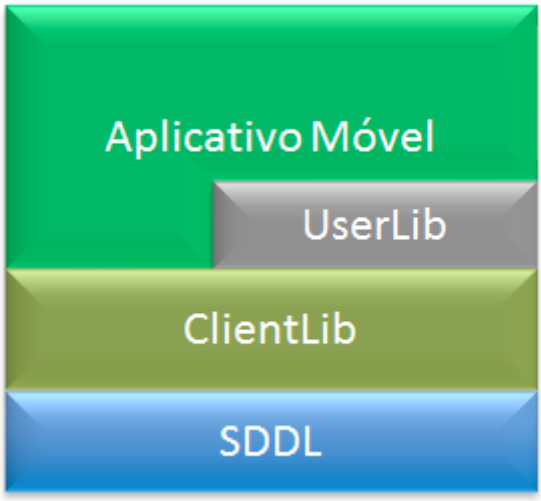


Figura 13 - Arquitetura de uma aplicação móvel usando uma UserLib

A Figura 13 ilustra a Arquitetura Forte descrita na segunda abordagem. Observe que o aplicativo móvel ainda tem acesso a recursos da ClientLib, sem que seja necessário utilizar a UserLib. Além disso, a UserLib pode ter acesso a recursos da ClientLib. Dessa forma, é possível que a UserLib possa tratar as mensagens recebidas diretamente do NodeConnection fornecendo métodos de callback para o NodeConnectionListener, como será melhor detalhado na sessão 5.2.3. Além disso, também poderá enviar mensagens de controle sem necessidade de intervenção humana através do aplicativo móvel, recurso que pode ser interessante para o desenvolvimento de aplicativos de rastreamento. No caso da UserLib usar o NodeConnection ou outros componentes da ClientLib

é necessário que a sua implementação conheça esses componentes. Porém, é importante ressaltar que o uso da ClientLib pela UserLib é opcional e não recomendado, uma vez que um mesmo componente de um sistema distribuído usando duas ou mais bibliotecas interdependentes aumentaria a complexidade do código, obrigando o desenvolvedor a refletir as alterações realizadas em uma biblioteca na outra e, qualquer erro nesse processo, poderia ocasionar problemas de incompatibilidade difíceis de serem corretamente identificados e corrigidos.

Portanto, foi acordado que parte desse projeto seria elaborado na forma de um guia de recomendações para auxiliar o usuário no desenvolvimento de seus aplicativos.

5.2 O módulo clientlib

O módulo clientlib é o módulo responsável por integrar a biblioteca clientlib no Titanium SDK, tornando possível sua utilização no desenvolvimento de aplicativos usando o Framework Titanium.

A estrutura do projeto é exibida na figura abaixo.

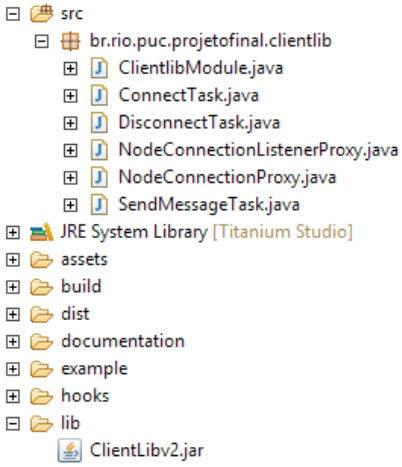


Figura 14 - Estrutura do projeto do módulo clientlib

Como pode ser observado, o módulo clientlib contém seis classes como segue:

5.2.1 ClientlibModule

Essa classe é a classe principal que estende *KrollModule*[8], o que permite ao módulo exportar métodos e propriedades para Javascript. Nesse projeto, a classe ClientlibModule nada mais é que um meio para acessar as proxies de NodeConnection e NodeConnectionListener que serão mais detalhadas à frente.

8 *KrollModule* é o tipo que representa um módulo nativo para Android. Na versão para iOS, o *KrollModule* deve ser substituído por *TiModule*.

5.2.2 NodeConnectionProxy

A NodeConnectionProxy é a classe que encapsula um objeto que implementa a interface NodeConnection e exporta os métodos desse objeto para Javascript. Abaixo segue uma lista das propriedades e métodos da NodeConnection que são acessíveis em Javascript.

Propriedades Get/Set:

- getProtocol/setProtocol;
- getIpAddress/setIpAddress;
- getPort/setPort.

Métodos:

- connect();
- disconnect();
- getNumberOfReconnectionsMad();
- getUUID();
- sendMessage(HashMap) throws IOException;
- addNodeConnectionListener(NodeConnectionListenerProxy).

Os métodos connect, disconnect e sendMessage são executados com o auxílio das classes ConnectTask, DisconnectTask e SendMessageTask, respectivamente.

O método sendMessage teve sua assinatura alterada para suportar um recurso muito utilizado no desenvolvimento de aplicativos com o Titanium, que são os HashMaps pois, ao exportar um método que recebe um HashMap para Javascript, é possível passar uma estrutura bem simples, porém robusta, quando chamamos esse método. Esse recurso é melhor ilustrado na figura abaixo.

```
nodeConnection.sendMessage({
    recipientID: '788b2b22-baa6-4c61-b1bb-01cff1f5f000',
    contentObject: 'Eu estou aqui'
});
```

Figura 15 - Exemplo de utilização do método sendMessage

Ao chamar esse método, a NodeConnectionProxy se encarrega de criar uma ApplicationMessage com o recipientID e contentObject passados e chama o método sendMessage de NodeConnection, passando a ApplicationMessage criada.

O método addNodeConnectionListener também precisou ter sua assinatura alterada em relação a assinatura de addNodeConnectionListener, da interface NodeConnection. Isso foi necessário porque não é possível acessar a

interface NodeConnectionListener diretamente em Javascript, motivo pelo qual existe a classe NodeConnectionListenerProxy.

5.2.3 NodeConnectionListenerProxy

Essa classe tem o mesmo objetivo da NodeConnectionProxy, que é exportar os métodos para Javascript. No entanto, a abordagem nesse caso precisa ser diferente, pois é o usuário que deve definir o que será executado no caso de algum dos eventos abaixo ocorrer:

- Connected;
- Reconnected;
- Disconnected;
- NewMessageReceived;
- UnsentMessages;
- InternalException.

Para tornar isso possível, utilizamos um dicionário que vincula uma string identificadora a uma *KrollFunction*[9]. Dessa forma, o usuário tem a liberdade de tratar todos os eventos que sua aplicação precisar com códigos em Javascript, ou ainda, caso deseje criar um tratamento mais robusto, pode criar seu próprio *PackagedModule,* exportando métodos para o tratamento desses eventos.

A figura abaixo ilustra como definir em Javascript os métodos de callback que serão chamados pelo NodeConnectionListenerProxy.

```
nodeConnectionListener.addEventListener('connected', function(e) {
    Ti.App.connected = true;
    nodeConnection.fireEvent('connectionStatusChanged');
});
nodeConnectionListener.addEventListener('disconnected', function(e) {
    Ti.App.connected = false;
    nodeConnection.fireEvent('connectionStatusChanged');
});
```

Figura 16 - Exemplo de uso do método addEventListener

5.2.4 ConnectTask, DisconnectTask e SendMessageTask

Um conceito simples e bastante importante ao desenvolver aplicações para dispositivos móveis, tem a ver com o cuidado de não executar tarefas longas na thread principal. Para isso foram criadas as classes ConnectTask, DisconnectTask e SendMessageTask. O objetivo principal dessas classes é evitar que os aplicativos fiquem congelados durante a execução dos respectivos métodos.

Para evitar esse problema, essas classes estendem AsyncTask, que é uma estrutura fornecida pela API da plataforma Android para executar tarefas em

9 *KrollFunction* é uma interface que expõe uma função Javascript para Java.

Background.

Portanto, toda vez que o aplicativo executar um comando connect, disconnect ou sendMessage, uma segunda thread será criada para tratar o respectivo método em NodeConnection, deixando a thread principal livre para continuar atualizando a tela e tratando os eventos de interação com o usuário.

5.3 Módulos componentes

Os módulos componentes, como anteriormente descritos, são módulos escritos diretamente em Javascript que utilizam tanto os componentes presentes fornecidos pelo Titanium SDK, quanto os recursos fornecidos pelo Módulo clientlib. Como foi descrito na sessão 5.1, é recomendado que exista uma biblioteca externa que encapsule as estruturas de dados que o usuário deseja utilizar na aplicação ContextNet. Portanto, os módulos aqui descritos não possuem sua funcionalidade completa, sendo necessário que o usuário faça a adaptação necessária para incluir sua própria biblioteca[10] no projeto.

Todos os módulos componentes devem acessar um único NodeConnection. Logo, é importante instanciar a NodeConnectionProxy e NodeConnectionListenerProxy no arquivo **App.js**, que é a classe onde devem ser instanciados todos os objetos globais da aplicação. Também é nesse arquivo que devem ser definidos os principais métodos de callback pelos quais o aplicativo será notificado sobre eventos relacionados ao NodeConnection.

Para que o objeto seja visível por todos os módulos da aplicação, é necessário que seja registrado junto à classe Ti.App. Abaixo segue um exemplo de como isso é feito.

```
// define as propriedades da aplicação
Ti.App.nodeConnection = nodeConnection;
Ti.App.connected = false;
```

Figura 17 - Exemplo de registro de objeto global

Para que as demais Views tenham acesso a esses eventos, é recomendado o uso das instruções fireEvent e addEventListener ao NodeConnection. Ambas as instruções estão presentes em qualquer objeto fornecido pelo framework Titanium, pois fazem parte da classe Titanium.Proxy que é a base de qualquer objeto Titanium.

O método fireEvent dispara um evento para um listener registrado e o método addEventListener registra uma callback sob um dado nome. Essa

10 Será necessário criar um módulo semelhante ao módulo clientlib para tornar os métodos e estruturas disponíveis na userlib acessíveis em Javascript. (vide Apêndice A)

callback será executada em resposta a um evento, disparado com o respectivo nome sob o qual foi registrada.

A figura abaixo ilustra como é feita a chamada ao método fireEvent para notificar sobre a ocorrência do evento *connectionStatusChanged*. Esse exemplo foi extraído do código contido em App.js da aplicação demo.

```
nodeConnectionListener.addEventListener('connected', function(e) {
    Ti.App.connected = true;
    nodeConnection.fireEvent('connectionStatusChanged');
});
```

Figura 18 - Exemplo de uma chamada a fireEvent

Em seguida, temos um trecho do código do módulo ConnectionView, exemplificando o registro de uma callback para o tratamento do evento *connectionStatusChanged.*

```
nodeConnection.addEventListener('connectionStatusChanged', function() {
    lStatusConexao.text = Ti.App.connected ? 'Conectado' : 'Desconectado';
});
```

Figura 19 - Listener para o evento connectionStatusChanged

Como foi dito anteriormente, uma aplicação distribuída do ContextNet, normalmente, possui uma biblioteca externa que encapsula as estruturas de dados necessárias ao negócio. A utilização de uma terceira biblioteca inviabilizaria a utilização deste framework por parte do usuário desenvolvedor, uma vez que o usuário desenvolvedor teria que reescrever todo o código da aplicação para incluir suas próprias estruturas. Portanto, os únicos módulos desenvolvidos nessa categoria foram o ConnectionView, MapView e ChatView. Também foi desenvolvido um MenuView que serve como exemplo para uma tela de Menu para uma aplicação desse tipo. Os demais módulos devem seguir o modelo do que foi desenvolvido na aplicação ContextNet-Mobile, no que se refere ao layout da aplicação.

5.3.1 ConnectionView

O módulo ConnectionView trata-se de uma tela onde o usuário pode configurar os dados necessários para a conexão com o middleware SDDL.

Figura 20 - Tela ConnectionView

Essa tela foi construída pensando na estrutura mínima necessária para a conexão de um objeto NodeConnection ao middleware SDDL. Se o usuário necessitar do envio de algum tipo de mensagem para autenticação de usuário, será necessário incluir manualmente esse tratamento.

Para facilitar o uso dessa tela, é recomendado que o usuário configure a propriedade keyboardType dos campos TextField para Ti.UI.KEYBOARD_NUMBER_PAD. Dessa forma, o teclado exibido quando o usuário deseja editar o IP ou Porta de acesso será como o exibido na figura abaixo.

Figura 21 - Teclado NUMBER PAD para edição dos campos IP e Porta da ConnectionView

5.3.2 MapView

Essa é a tela mais importante de uma aplicação ContextNet. É nela que o usuário permanecerá na maior parte do tempo em que estiver utilizando a aplicação, além de conter botões para acesso às funções mais comuns em um aplicativo ContextNet.

Para o desenvolvimento dessa tela, é necessário fazer o download do módulo Maps[12], disponível no Titanium Market. Esse módulo é gratuito e fornece as funcionalidades do Google Maps v2.

A figura abaixo exibe uma visão geral dessa tela.

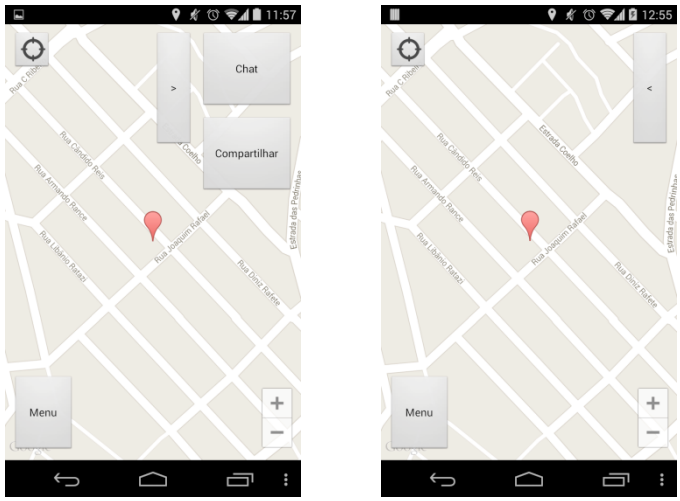


Figura 22 - Tela MapView

No canto superior direito, existe um pequeno menu que pode ser escondido durante a execução do aplicativo para permitir maior visualização do mapa. Nesse menu, o usuário tem a opção para acessar dois outros módulos, Chat e Compartilhar.

No canto superior esquerdo, existe um botão que centraliza o mapa na posição atual, função comum a qualquer aplicação que contenha um mapa.

No canto inferior esquerdo, existe o botão Menu que dá acesso a um módulo exemplo que contém apenas uma série de botões que dão acesso a outros módulos, como o módulo de Conexão e a uma possível tela de configuração.

Além disso, internamente, esse componente usa um módulo disponível na API do Titanium, chamado Geolocation. Esse módulo faz parte do SDK e só precisa ser chamado e configurado para entrar em operação. Para garantir que o módulo ativará os recursos de geolocalização do dispositivo, basta que seja definido pelo menos um listener para o evento location. Enquanto houver pelo menos um listener ouvindo esse evento, o Titanium SDK garante que o sistema de geolocalização, com as características configuradas, estará ativo.

O evento location do módulo Geolocation retorna uma estrutura coords com as coordenadas (latitude e longitude) do dispositivo. Internamente, o módulo MapView já conta com um método de callback chamado *locationHandler,* responsável por atualizar o marcador que indica a posição atual do dispositivo no mapa.

Além de tratar os eventos de geolocalização, o MapView fornece os seguintes métodos:

- setLocation – útil para definir a posição central do mapa. Essa função pode ser usada para um caso simples, como centralizar o mapa na posição atual (como faz o botão centralizar no canto superior esquerdo), ou ainda para uma função de alerta, onde uma central poderia chamar a atenção de um usuário do aplicativo para um determinado ponto do mapa, centralizando o ponto de interesse no mapa, sem que o usuário precise procurar por esse ponto.
- updateCurrentLocation – essa função é útil somente para atualizar a posição do marcador da posição atual do dispositivo no mapa. Fica disponível como método público do módulo MapView para caso o usuário desenvolvedor deseje alterar o comportamento do listener, ou criar seu próprio módulo de geolocalização.
- addAnnotation – uma annotation nesse contexto é um marcador que indica alguma coisa no mapa. Pode ser um acidente, a localização de outro veículo, ou um ponto de interesse qualquer. Caso a aplicação que o usuário desenvolvedor estiver criando necessitar compartilhamento de pontos de interesse, esse método deverá ser utilizado para incluir esses pontos no MapView.
- removeAnnotation – esse método deve ser usado para remover uma anotação, ou seja, um marcador qualquer que deixou de existir ou não é mais de interesse do usuário.

Além disso, o MapView ainda dispara os seguintes eventos que deverão ser tratados pelo usuário desenvolvedor.

- openChatView – Indica que o usuário da aplicação clicou sobre o botão Chat.
- openMenuView – Indica que o usuário da aplicação clicou sobre o botão Menu.
- openShareView – Indica que o usuário da aplicação clicou sobre o botão Compartilhar.

Todas as funções podem ser completamente reimplementadas pelo usuário que desejar utilizar o framework aqui descrito.

5.3.3 ChatView

O módulo de chat disponível no framework serve como base para o desenvolvimento da funcionalidade de Chat no sistema, pois é necessário respeitar as regras que possam vir a existir para o tráfego de mensagens pelo middleware SDDL (p.ex. Necessidade de log para acompanhamento das

mensagens, estrutura para armazenamento das mensagens, entre outras). Portanto, foi desenvolvido um esqueleto que dispõe dos métodos mais básicos para o envio e recebimento de mensagens de forma a exemplificar a funcionalidade de Chat no aplicativo ContextNet.

Figura 23 – ChatView

O Apêndice B item 6 contém um passo a passo para a construção desse módulo, que serve de base para que o usuário desenvolvedor possa estender este framework de forma a integrar todas as funcionalidades que deseja.

6 Implementação e avaliação

6.1 Projeto de implementação

A implementação do presente framework foi planejada para ser como uma decomposição da aplicação ContextNet-Mobile em módulos independentes que pudessem ser reutilizados para o desenvolvimento de uma aplicação qualquer, que utilize a arquitetura ContextNet. No entanto, como optamos por tentar criar um framework que além de possibilitar o desenvolvimento flexível de aplicações para o ContextNet também fosse multiplataforma, foram necessárias adaptações e nem todas as funcionalidades presentes na aplicação ContextNet-Mobile puderam ser devidamente implementadas.

Além disso, como não é possível prever os tipos de estruturas definidos pelo negócio, e não era objetivo do trabalho forçar o usuário a respeitar um protocolo específico, já que middleware SDDL suporta a transferência de dados em qualquer estrutura, desde que esta seja serializável, parte do framework foi desenvolvido apenas de forma teórica, como um guia para que os usuários possam tirar o máximo proveito das estruturas fornecidas pelo Titanium SDK e

suportadas pelo módulo clientlib.

Portanto, o foco principal durante o desenvolvimento do projeto foi possibilitar maior flexibilidade ao módulo clientlib, de forma que este pudesse ser reutilizado em qualquer aplicação.

6.2 Testes e avaliações

Para testar e avaliar o framework foi desenvolvida uma aplicação demo, bastante simples e sem a utilização de uma UserLib, de forma a avaliar o funcionamento dos recursos da clientlib e da estrutura de interação entre os módulos escolhida para o projeto.

Essa aplicação, por se tratar de uma demo, não tinha um foco específico. Porém, contém exemplos dos módulos que podem ser incluídos em uma aplicação ContextNet.

6.3 Problemas encontrados

Ocorreram alguns problemas durante o desenvolvimento do módulo clientlib, sendo o maior deles, uma incompatibilidade da maquina virtual utilizada pelo Android em trabalhar o arquivo .classpathentry dentro do .JAR. Essa incompatibilidade causava um erro relacionado ao ClientLibProtocol, que ocorria em virtude da falta do pacote protobuf fornecido pelo Google. Esse erro impossibilitava a aplicação de enviar uma pacote qualquer através da ClientLib e foi contornado com a ajuda da equipe do LAC, que construiu uma versão da biblioteca ClientLib, chamada ClientLibv2, que inclui o pacote protobuf.

Também existe um problema relacionado à integração de uma UserLib ao projeto do framework. Como a maior parte do framework diz respeito à implementação de funcionalidades disponíveis através da arquitetura ContextNet, sem o conhecimento prévio das estruturas que deverão ser utilizadas não é possível implementar uma funcionalidade completa. Com isso, foi necessário criar um manual para o desenvolvimento de aplicações para o ContextNet com o Titanium SDK, utilizando o módulo clientlib. Tal fato foge ao escopo inicial do trabalho onde se pretendia criar um conjunto de funcionalidades a serem utilizadas pelo usuário para o desenvolvimento da aplicação. Com essa nova abordagem, o framework passa a ser algo mais teórico e menos prático.

7 Considerações finais

Concluindo este trabalho, acredito que os objetivos foram alcançados em parte. É possível desenvolver uma aplicação semelhante ao ContextNet-Mobile usando o framework desenvolvido mas, no entanto, a parte que poderá melhor ser aproveitada é o módulo clientlib, sendo o usuário ainda obrigado a aprender uma nova tecnologia e criar seu próprio módulo, expor os métodos disponíveis na UserLib (similar a biblioteca ARFF) para Javascript e desenvolver toda a integração desse novo módulo aos demais módulos o que, de certa forma, permite que o usuário cometa diversos erros durante a implementação do aplicativo.

Por outro lado, o desenvolvimento deste projeto foi bastante interessante do ponto de vista do aprendizado de novas tecnologias e de uma linguagem que desconhecia e não tive contato durante o curso, que foram o Titanium SDK e Javascript. O Titanium SDK, desenvolvido pela Appcelerator, por ser um framework bastante rico e versátil para o desenvolvimento de aplicações simples, fornecendo ferramentas para desenvolvimento rápido e multiplataforma e Javascript por ser uma linguagem incrivelmente simples e poderosa.

Para o futuro, acredito que o framework possa ser melhorado e expandido, de forma que seja definida uma biblioteca com as estruturas que irão trafegar os dados de contexto (p.ex. uma estrutura comum para armazenar e transferir os dados de gps). Hoje, algo semelhante a essa biblioteca seria a ARFF_Library, que ainda tem um foco específico.

8 Referências Bibliográficas

[1] ENDLER, M.; BAPTISTA, G., L.; SILVA, L.D.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V., **ContextNet: Context Reasoning and Sharing Middleware for Large-scale Pervasive Collaboration and Social Networking.** *Poster Session, ACM/USENIX Middleware Conference*, Lisbon, December 2011.

[2] DAVID, L.; VASCONCELOS, R.; ALVES, L.; ANDRÉ, R.; BAPTISTA, G., ENDLER, M., **A Communication Middleware for Scalable Real-time Mobile Collaboration.** *IEEE 21st International WETICE, Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architetures (AROSA), pp. 54-59*, Toulouse, June 2012.

[3] VASCONCELOS, I.; VASCONCELOS, R.; BAPTISTA, G.; SEGUIN, C.; ENDLER, M, **Desenvolvendo Aplicações de Rastreamento e Comunicação**

Móvel usando o Middleware SDDL. SBRC, 2013.

[4] OMG, **Data Distribution Service for Real-time Systems.** 2007.

[5] SILVA, L.D.N.; ENDLER, M.; RORIZ, M., **MR-UDP: Yet another Reliable User Datagram Protocol, now for Mobile Nodes.** MCC 06/2013, Dept. de Informática, PUC-Rio, ISSN 0103-9741, May 2013.

[6] MAC DOWELL, André V. G. de A., **Cliente Android para comunicação instantânea e compartilhamento de contexto usando mapas.** Projeto Final de Graduação, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro, 2012.

[7] Corona Labs Inc., **Corona SDK** [http://www.coronalabs.com/products/corona-sdk/]

[8] Adobe Systems Inc., **PhoneGap** [http://phonegap.com]

[9] Appcelerator Inc., **Titanium SDK** [http://www.appcelerator.com/platform/titanium-sdk]

[10] Appcelerator Inc., **Titanium Module Concepts**. Disponível em: http://docs.appcelerator.com/titanium/latest/#!/guide/Titanium_Module_Concepts (último acesso em Junho 2013)

[11] Google Inc., **Android APIs Develop Reference**. Disponível em: http://developer.android.com/reference/packages.html (último acesso em Janeiro 2014)

[12] Appcelerator Inc., **Modules.Map**. Disponível em: http://docs.appcelerator.com/titanium/3.0/#!/api/Modules.Map (último acesso em Janeiro 2014)

[13] PULSEN, T.; WHINNERY, K.; LUKASAVAGE, T.; DOWSETT, P., **Building Mobile Apps with Titanium.** Appcelerator, 2012.

Apêndice A Passo a passo para o desenvolvimento de módulos para o Titanium SDK.

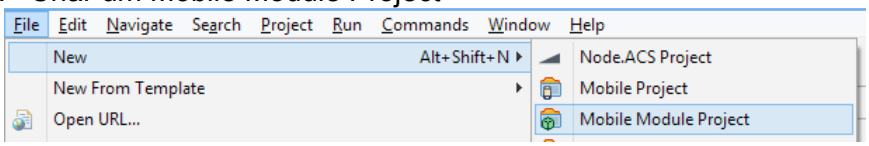
9 O desenvolvimento de módulos no Titanium SDK é bastante simples. Porém, é necessário esclarecer alguns conceitos importantes. Para isso é recomendada a leitura do capítulo Titanium Module Concepts da documentação da API do Titanium, disponível no seguinte endereço: http://docs.appcelerator.com/titanium/3.0/#!/guide/Titanium_Module_Concepts.

10 Um módulo do Titanium pode ser útil para encapsular uma biblioteca externa e expor seus métodos e objetos em Javascript (os objetos seriam expostos como proxies). Módulos de extensão podem ser facilmente desenvolvidos com o auxílio do Titanium Studio. (OBS. Para aqueles que não desejam utilizar a IDE fornecida, existe na documentação do Titanium um guia pra o desenvolvimento de módulos com Eclipse e Ant).

11 Para o desenvolvimento de um módulo com o Titanium Studio basta seguir os seguintes passos:

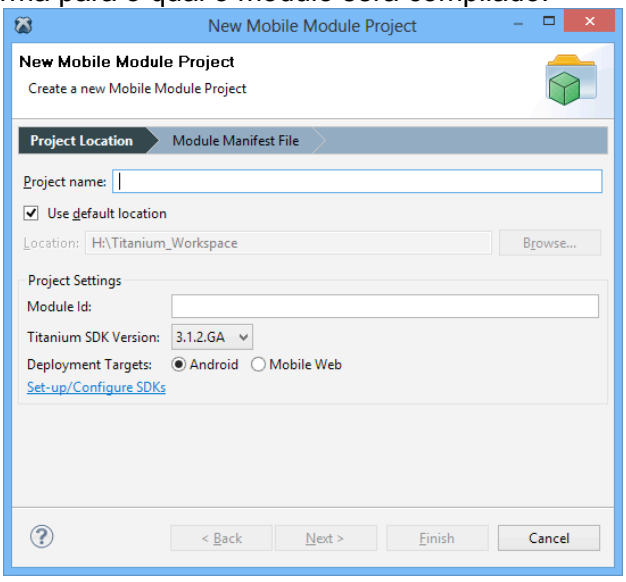
1. Criar um Mobile Module Project

12



13 **Imagem 1 – Criação de Mobile Module Project**

2. Depois o usuário deverá preencher o nome do projeto, o id do module, a versão do Titanium SDK que deseja utilizar e a plataforma para o qual o módulo sera compilado.



14

15 **Imagem 2 - Configuração do novo módulo**

3. Em seguida será exibida a seguinte janela, para a inclusão de formações sobre versão, autoria, licença e descrição do módulo que será desenvolvido.

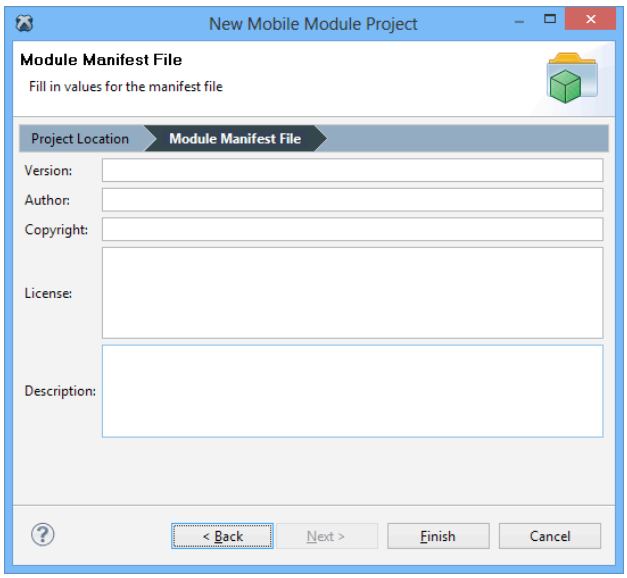


Imagem 3 - Informações sobre o módulo

4. Depois de clicar em Finish, o projeto do novo módulo será aberto na view App Explorer, e a seguinte estrutura poderá ser visualizada na pasta src.

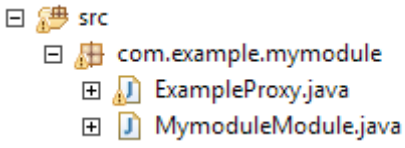


Imagem 4 - Arquivos criados automaticamente

Estes arquivos foram criados automaticamente pelo Titanium e contém exemplos de métodos e propriedades que podem ser criadas em um módulo.

Para o desenvolvimento de um módulo semelhante ao clientlib, será necessário incluir uma biblioteca externa. Isso pode ser feito incluindo essa biblioteca no diretório lib e no buildpath do projeto. Depois, basta criar as estruturas necessárias para encapsular os objetos que deseja e incluir as anotações para expor os métodos e propriedades em Javascript.

23

Apêndice B Recomendações para o desenvolvimento de aplicações móveis para o ContextNet usando Titanium e o módulo ClientLib.

24 Para o desenvolvimento de aplicações utilizando o Titanium e o módulo clientlib é necessário respeitar algumas instruções quanto ao uso e tratamento de eventos relacionados ao NodeConnection assim como, considerar o fluxo entre as diversas telas da aplicação.

25 Este guia não tem a pretensão de impor ao usuário um fluxo necessário entre as telas, porém, indica um fluxo comum para as aplicações às quais o presente framework foi desenvolvido.

26

1. Inicialização do módulo ClientLib.

27 Ao iniciar a aplicação será necessária a criação de um NodeConnection e do respectivo listener. Isso deverá ser realizado no arquivo App.js, conforme ilustrado na imagem abaixo.

28

```
28    /*****************************************************************
29     * Configuração da aplicação ContextNet
30     */
31    // inicia o módulo ClientLib
32    var ClientLib = require('br.rio.puc.projetofinal.clientlib');
33
34    // Cria node connection
35    var nodeConnection = ClientLib.createNodeConnection();
36
37    // define as propriedades da aplicação
38    Ti.App.nodeConnection = nodeConnection;
39    Ti.App.connected = false;
40
41    // Cria node connection listener principal
42    var nodeConnectionListener = ClientLib.createNodeConnectionListener();
43    nodeConnection.addNodeConnectionListener(nodeConnectionListener);
```

2. Definição de variáveis globais

29 Na figura acima, nas linhas 38 e 39, são definidas duas variáveis globais à aplicação, uma para o nodeConnection e outra para indicar o estado da conexão.

30 A definição dessas variáveis é importante para que todos os componentes da aplicação possam ter conhecimento sobre o estado atual da conexão e

também conhecer o NodeConnection responsável por conectar essa aplicação ao middleware SDDL.

3. **Interação entre Views.**

31 É recomendado que a interação entre as diversas views da aplicação seja realizada por meio de eventos, com a utilização dos métodos fireEvent e addEventListener. Isso ajuda a centralizar os métodos de callback para os eventos do NodeConnectionListener, permitindo que todas as views da aplicação permaneçam atualizadas, mesmo as que não estiverem sendo exibidas no momento.

32 Como no Titanium SDK as aplicações, normalmente, possuem uma implementação diferente de Window para cada plataforma, é recomendado que o tratamento aos eventos disparados pelas Views seja feito diretamente na Window que contém a View que disparou o evento. Dessa forma, o evento pode ser tratado usando as funções específicas de cada plataforma.

4. **Centralizar listeners e eventos**

33 É recomendado, também, que seja criado um arquivo Javascript para centralizar todos os listeners e eventos globais da aplicação.

34 Numa aplicação com um número muito grande de eventos distintos, é importante que o arquivo que contém o código que trata todos estes eventos seja separado do código de inicialização da aplicação. Dessa forma, se tornará mais fácil a manutenção e evolução do aplicativo.

5. **Criar MapController**

35 Assim como foi sugerido para os listeners e eventos gerais, é importante que exista um mecanismo único para atualização e controle do MapView. Apesar de o próprio MapView possuir métodos que permitam sua customização no que se refere a inclusão/remoção de pontos de interesse e centralização da região atual do mapa, não há mecanismos para monitoramento de eventos. Portanto, é recomendado que exista um mecanismo que traduza os eventos relacionados ao mapa e execute os métodos necessários para refletir o resultado desses eventos no MapView.

6. **Criação de novos módulos**

36 Será necessário a criação de novos módulos para compor a aplicação. Esses módulos deverão ser compatíveis com o modelo sugerido pelo framework. Logo, terão de interagir com os demais módulos através de eventos.

37 Para criar um novo módulo View, basta criar um arquivo .js, dentro desse arquivo. Deve-se criar uma função que leva o mesmo nome do módulo.

Para exemplificar vamos criar o módulo ChatView.

Inicialmente o arquivo criado estará vazio, portanto, devemos escrever todo o código do zero. Primeiro, criamos a assinatura da função ChatView(). Dentro do escopo da função, criamos uma variável para guardar uma referência para o nodeConnection da aplicação e uma variável self que é semelhante ao “this” em Java. A função criada deve retornar a variável self.

```
function ChatView(){
    // obtém o nodeConnection da aplicação
    var nodeConnection = Ti.App.nodeConnection;

    // cria uma instância de View
    var self = Ti.UI.createView();



    return self;
}

module.exports = ChatView;
```

Depois disso, precisamos preencher a tela com os componentes visuais. Para esse exemplo, vou criar a tela ChatView com uma TableView para armazenar a conversa, onde cada linha da tabela será um item da conversa.

TableView

```
// cria uma tabela de linhas para armazenar o histórico das mensagens
var tableMensagens = Ti.UI.createTableView({
        top: '10dip',
        bottom: '60dip'
});
self.add(tableMensagens);
```

Quando trabalhamos com o componente TableView, é interessante utilizarmos uma função auxiliar que cria cada linha da tabela. Dessa forma, podemos definir o layout dos elementos da tabela de forma independente do restante do código, o que facilita a atualização do aplicativo. Abaixo segue o código dessa função:

```
var createChatRow = function(item) {
    var tablerow = Ti.UI.createTableViewRow({
        height: 'auto',
        className: 'itemRow',
        hasChild: false
    });
    var senderview = Ti.UI.createLabel({
        text: item.sender,
        textAlign: item.alignName,
        font: { fontSize: '10dip' },
        top: '3dip',
        width: '360dip',
        left: '10dip',
        height: 'auto'
    });
    var messageview = Ti.UI.createLabel({
        text: item.message,
        textAlign: 'left',
        font: { fontSize: '14dip' },
        top: '20dip',
        width: '360dip',
        left: '10dip',
        height: 'auto'
    });

    tablerow.add(senderview);
    tablerow.add(messageview);

    return tablerow;
};
```

Além da TableView, criamos também um TextField, onde o usuário do aplicativo irá digitar a mensagem que deseja enviar e um Button, para que o usuário possa enviar a mensagem.

TextField:

```
// TextField para escrever a msg q deseja enviar.
var txtMsg = Ti.UI.createTextField({
    height : 'auto',
    bottom : '10dip',
    left : '10dip',
    width : '240dip',
    hintText : 'Mensagem',
    softKeyboardOnFocus : Ti.UI.Android.SOFT_KEYBOARD_DEFAULT_ON_FOCUS, // Android only
    keyboardType : Ti.UI.KEYBOARD_DEFAULT,
    returnKeyType : Ti.UI.RETURNKEY_DEFAULT,
    borderStyle : Ti.UI.INPUT_BORDERSTYLE_ROUNDED
});
self.add(txtMsg);
```

Button:

```
// Create a Button.
var btnEnviar = Ti.UI.createButton({
    title : 'Enviar',
    height : 'auto',
    width : '120dip',
    bottom : '10dip',
    right : '10dip'
});
```

50 Para o botão também teremos que registrar a função de callback que será executada no evento '*click*', como segue:

51

```
btnEnviar.addEventListener('click', function() {
    /*********************************************
     * Aqui entra o código para encapsular a msg na estrutura de dados utilizada
     * e envia-la através do nodeConnection
     */
    // exemplo
    try
    {
        var item = {
            sender: 'me',
            alignName: 'right',
            message: txtMsg.value
        };
        tableMensagens.appendRow(createChatRow(item));
        var message = {
            recipientID: '788b2b22-baa6-4c61-b1bb-01cff1f5f000',
            contentObject: txtMsg.text
        };
        nodeConnection.sendMessage(message);
    }
    catch (Err)
    {
        var toast = Ti.UI.createNotification({
            message: 'Erro ao enviar msg',
            duration: Ti.UI.NOTIFICATION_DURATION_LONG
        });
        toast.show();
    }
});
```

52 Com isso temos toda a estrutura da tela de Chat montada e já é possível enviar mensagens. No entanto, as mensagens recebidas através do nodeConnection deverão ser tratadas por um objeto global. Do contrário, só seria possível receber as mensagens se o usuário permanecesse na tela de Chat. Portanto, é necessário que o módulo ChatView permita que um agente externo possa incluir mensagens através de algum método. Da mesma maneira, não é função da ChatView armazenar e organizar as conversas. Logo, também é necessário que o agente externo responsável por isso tenha acesso a todas as mensagens, para que possa organiza-las e armazena-las da forma que o usuário desenvolvedor escolher e carrega-las

na ChatView sempre que for necessário.

53 É recomendado então criar pelo menos dois métodos de acesso às mensagens: um para incluir uma nova mensagem no chat e outro para receber todas as mensagens carregadas, como ilustrado abaixo.

54

```
/*****************************************************************
 * É necessário criar um método, semelhante a esse que permita
 * que um agente externo inclua uma nova mensagem
 */
self.receiveMessage = function(message) {
    tableMensagens.appendRow(createChatRow(message));
};

/*****************************************************************
 * Também é necessário criar um método pelo qual um agente externo
 * possa copiar todos os métodos.
 * Esse agente externo é que deve ser responsável por guardar o
 * histórico da conversa
 */
self.getAllMessages = function() {
    return tableMensagens.data;
};
```

55

56

57