UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SISTEMA MÓVEL NA PLATAFORMA ANDROID PARA COMPARTILHAMENTO DE GEOLOCALIZAÇÃO USANDO MAPAS E NOTIFICAÇÕES DA GOOGLE

BRUNO ANDRÉ KESTRING

BLUMENAU
2014

2014/2-03

BRUNO ANDRÉ KESTRING

SISTEMA MÓVEL NA PLATAFORMA ANDROID PARA COMPARTILHAMENTO DE GEOLOCALIZAÇÃO USANDO MAPAS E NOTIFICAÇÕES DA GOOGLE

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

Prof. Dalton Solano dos Reis, M.Sc. - Orientador

BLUMENAU
2014

2014/2-03

SISTEMA MÓVEL NA PLATAFORMA ANDROID PARA COMPARTILHAMENTO DE GEOLOCALIZAÇÃO USANDO MAPAS E NOTIFICAÇÕES DA GOOGLE

Por

BRUNO ANDRÉ KESTRING

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

Presidente:	Prof. Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, M.Sc. – FURB
Membro:	Prof. Francisco Adell Péricas, M.Sc. – FURB

Blumenau, 08 de dezembro de 2014

Dedico este trabalho a todos que de alguma forma me ajudaram no decorrer do curso, aos meus colegas, amigos, professores e em especial à minha família.

AGRADECIMENTOS

A Deus por permitir que eu chegasse até este momento, mesmo tendo muitas adversidades.

À minha família que sempre me apoiou na conclusão da minha graduação.

À minha namorada, Vivian de Lima Panzenhagen, que mesmo com adversidades me apoiou sempre para a conclusão do curso e na realização deste trabalho.

Aos meus amigos que indiretamente me ajudaram a passar por este momento difícil.

Ao meu orientador Dalton Solano dos Reis, por acreditar em mim e por ser fonte de inspiração para o meu futuro.

À Google, por permitir que grandes tecnologias estejam disponíveis para uso acadêmico, permitindo uma grande difusão de conhecimento no meio acadêmico.

RESUMO

Este trabalho apresenta o desenvolvimento de um aplicativo cliente para a plataforma Android, que apresenta em um mapa interativo, através de recursos de geolocalização, a posição do usuário e pontos cadastrados pelo usuário ou pela universidade, e um servidor *webservice* responsável por suprir as informações do primeiro. É composto por um aplicativo cliente desenvolvido na plataforma Android capaz de apresentar no mapa a posição atual do usuário, a posição de outros usuários logados, pontos de interesse do usuário e da universidade. Utilizou-se da nova versão do Google Maps no Android, o Google Maps Android API v2 e para notificações implementou-se um servidor GCM Cloud Connection Server para que os dispositivos móveis também possam enviar notificações ao Google Cloud Message da Google. Durante o trabalho apresentou-se os conceitos e implementações das novas funcionalidades apresentadas pela Google. Como resultado foi implementado uma aplicação cliente, que permite ao usuário visualizar pontos de interesse e outros usuários no mapa, e um aplicativo servidor implementado como *webservice* responsável por responder as requisições do aplicativo cliente.

Palavra-chave: Computação móvel. Android. Notificação. Geolocalização.

ABSTRACT

This paper presents the development of a client application for the Android platform, which features an interactive map through geolocation features, the user's position and points registered by the user or by the university, and a webservice server responsible for supplying the information first. It consists of a client application developed on the Android platform able to display on the map the current position of the user, the position of other logged in users, user interest points and the university. It was used the new version of Google Maps on Android, the Google Maps API v2 and Android notifications was implemented one GCM Cloud Connection Server so that mobile devices can also send notifications to Google Cloud Message from Google. During the work presented the concepts and implementations of new features introduced by Google. As a result was implemented a client application that allows the user to view points of interest and other users on the map, and a server application implemented as webservice responsible for answering the client application requests.

Keyword: Mobile computing. Android. Notification. Geolocation.

LISTA DE FIGURAS

Figura 1– Interface de visualização de proximidade 20
Figura 2 – Tela de informações de ônibus 21
Figura 3 – Cenário do sistema 23
Figura 4 – Diagrama de casos de uso 27
Figura 5 – Diagrama de Classe do pacote Modelo do Servidor 33
Figura 6 – Diagrama de Classe do pacote DAO 34
Figura 7 – Diagrama de classe do pacote controlador 35
Figura 8 – Diagrama de classe do pacote recursos 35
Figura 9 – Diagrama geral de relação das camadas do servidor 36
Figura 10– Diagrama de classes do pacote `br.com.viseduandroid.notification` 37
Figura 11 – Diagrama de classes do pacote `br.com.viseduandroid.adapter` 38
Figura 12 – Diagrama de classe do pacote `br.com.viseduandroid.view` 39
Figura 13 – Diagrama de classes do pacote `br.com.viseduandroid.webserviceclient` 39
Figura 14 – Diagrama de classes do aplicativo cliente 40
Figura 15 – Diagrama de sequência `Visualizar pontos de interesse local` 42
Figura 16 – Diagrama de sequência `Visualizar amigos` 43
Figura 17 – MER do banco de dados do servidor 44
Figura 18 – Tela inicial de *login* (esquerda) e autenticação do usuário (direita) 62
Figura 19 – Tela Mapa com a posição do usuário (esquerda) e adicionar novo ponto local (esquerda) 62
Figura 20 – Na `tela Mapa` (esquerda), tela de cadastro de Ponto Local (centro) e finalizando o ponto é apresentado na tela Mapa (direita). 63
Figura 21 – Tela Lista de amigos (esquerda), o registro do amigo (centro) e o recebimento de notificação (direita) 64
Figura 22 – Tela de Lista de Pontos Globais (esquerda) e o registro do ponto global (direita). 65
Figura 23 – Imagem do Google Maps no Browser (esquerda) e Google Maps no aplicativo cliente 66

LISTA DE QUADROS

Quadro 1 – Caso de uso Efetuar *login* 28
Quadro 2 – Caso de uso Visualizar posição atual 28
Quadro 3 – Caso de uso Manter pontos de interesse local 29
Quadro 4 – Caso de uso Visualizar pontos de interesse globais 29
Quadro 5 – Caso de uso Manter amigos 30
Quadro 6 – Caso de uso Receber informações para registrar novo Ponto de Interesse Local 30
Quadro 7 – Caso de uso Receber informações para registrar novo Amigo 31
Quadro 8 – Caso de uso Informar Pontos de Interesse Globais 31
Quadro 9 – Caso de uso Validar login 32
Quadro 10 – Código da classe ConnectionManager 45
Quadro 11 – Método construtor da classe ConnectionManager 46
Quadro 12 – Classe UsuarioDAO 47
Quadro 13 – Termino do método inserir da classe UsuarioDAO 47
Quadro 14 – Implementação da classe UsuarioResource 48
Quadro 15 – Conexão XMPP e o método send 50
Quadro 16 – Configuração da conexão XMPP 50
Quadro 17 – Continuação do método conect() 51
Quadro 18 - Continuação do método conect 52
Quadro 19 – Código da classe GcmPacketExtension 53
Quadro 20 – Fim da classe GcmPacketExtension 53
Quadro 21 – Envio de mensagem Ack para quem enviou notificação 54
Quadro 22 – Configuração da entidade que trata notificações na aplicação cliente 54
Quadro 23 – Classe Globals 55
Quadro 24 – Classe GcmBroadcastReceiver 55
Quadro 25 – Método onHandleIntent da classe GcmIntentService 56
Quadro 26 – Método sendMessage da classe AmigoActivity 57
Quadro 27 – Configuração do XML activity_mapa da tela do mapa 58
Quadro 28 – Métodos responsáveis por manter os pontos de interesse atualizados 59

Quadro 29 – Método que trata a mudança de posição do usuário 61
Quadro 30 – Quadro comparativo entre os trabalhos correlatos 68

LISTA DE TABELAS

Tabela 1 – Testes de desempenho 67

LISTA DE SIGLAS

ADT – *Android Development Tools*

API – *Application Programming Interface*

CCS – *Cloud Connection Server*

DAO – *Data Access Object*

EA – Enterprise Architect

FURB - Fundação Universidade Regional de Blumenau

GCM – *Google Cloud Message*

GPS – *Global Position System*

HTTP – *HyperText Transfer Protocol*

IDE – *Integrated Development Enviornment*

JDBC – *Java Database Connectitivy*

JSON – *JavaScript Object Notation*

MER – Modelos de Entidades e Relacionamentos

RF – Requisito Funcional

RNF – Requisito Não Funcional

SDK – *Software Development Kit*

SQL – *Structured Query Language*

UC – *Use Case*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

UTM - Universal Transversa de Mercator

XMPP – *eXtensible Messaging and Presence Protocol*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 14
1.2 ESTRUTURA 14
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 INTERAÇÃO FURB 16
2.2 GEOLOCALIZAÇÃO (GPS, CARTOGRAFIA, COORDENADAS UTM) 17
2.3 ANDROID E OS FRAMEWORKS (MAPAS, GPS E NOTIFICAÇÃO) 17
2.4 TRABALHOS CORRELATOS 18
2.4.1 Protótipo de sistema móvel na plataforma Android para compartilhamento de arquivos e mensagens entre dispositivos baseado em proximidade geográfica 19
2.4.2 Harvard Mobile Web application 20
2.4.3 Protótipo de mundo virtual para relacionamento com participantes do Interação FURB 21
3 DESENVOLVIMENTO DO APLICATIVO 23
3.1 APLICATIVO PROPOSTO 23
3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 25
3.3 ESPECIFICAÇÃO 26
3.3.1 Casos de uso 26
3.3.2 Classes do aplicativo servidor 32
3.3.3 Classes do aplicativo cliente 36
3.3.4 Diagrama de sequência 40
3.3.5 Banco de Dados 43
3.4 IMPLEMENTAÇÃO 44
3.4.1 Técnicas e ferramentas utilizadas 45
3.4.2 Operacionalidade da implementação 61
3.5 RESULTADOS E DISCUSSÃO 65
3.5.1 Teste de Desempenho 66
3.6 COMPARATIVO DOS TRABALHOS CORRELATOS 67
4 CONCLUSÕES 69
4.1 EXTENSÕES 70

1 INTRODUÇÃO

A evolução dos meios de comunicações mudou a forma de viver. O início dos telefones móveis foi no ano de 1983 com o Motorola DynaTAC 8000x. A primeira geração da telefonia celular se iniciava com celulares não tão portáteis, tanto que a maioria era desenvolvida para instalação em carros (TECMUNDO, 2014).

Com o passar do tempo, a evolução dos celulares trouxe a internet para os aparelhos. Trouxe mais mobilidade ao usuário, pois agora ele pode acessar a internet de onde quer que ele esteja. Conquistar a atenção do usuário passou a ser uma tarefa difícil e para isto, os celulares passaram a ser alvo de técnicas de computação antes vistas apenas em computadores, devido à alta taxa de processamento demandado.

Uma das formas de conquistar a atenção destes usuários são os mundos virtuais. Mundos virtuais proporcionam a interação não só entre usuários, mas do usuário com os objetos do mundo virtual (ALBUQUERQUE; VELHO, 2001, p. 4). Não apenas novas técnicas de computação, mas também novas formas de interação dos usuários para que a imersão do mesmo seja maior. Cresce muito a quantidade de aplicativos que utilizam informações de geolocalização do usuário.

Atualmente a Universidade Regional de Blumenau realiza um evento que tem a intenção de aproximar os alunos do Ensino Médio à Universidade. Este evento se chama Interação FURB (ANGELI, 2010). No Interação FURB, o visitante tem a oportunidade de conhecer a estrutura da Universidade Regional de Blumenau e, também, de participar das oficinas organizadas. Através das atividades práticas desenvolvidas, é possível vivenciar o que a Universidade e os organizadores podem oferecer (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014b).

A principal forma de comunicação entre os organizadores e os alunos envolvidos é a plataforma virtual (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014b) e caso as escolas solicitem, é realizado também palestras. Não há outras formas de contato ou interação com os alunos.

Um cenário que poderia se propiciar com a evolução da mobilidade (avanços nos meios de comunicação e dos dispositivos móveis) associada com a interatividade usando informações de geolocalização seria o Interação FURB (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014b). O Interação FURB é um evento de ocorrência anual, que tem a intenção de aproximar os alunos (visitantes) do Ensino Médio à Universidade. Neste evento são apresentadas informações para os visitantes, além de ações de relacionamento que façam

o visitante ter motivação em interagir com a FURB. Também procura mostrar os diversos espaços físicos da universidade, muitas vezes desconhecido, fazendo com que os visitantes se locomovam entre salas, blocos e campus. Toda esta movimentação exige certa infraestrutura de sinalização, pessoas guias, entre outros, para que a logística de mudança de espaço físico seja possível de ocorrer no curto período de tempo (um dia). Desta forma, somando-se o fato de cada vez mais os visitantes possuírem um dispositivo móvel, com a necessidade locomoção entre espaços desconhecidos em curto período de tempo, acreditasse que uma aplicação que indique a posição atual do usuário e permita visualizar pontos de interesse no espaço físico da FURB, possam auxiliar a locomoção deste visitante.

Por fim, este trabalho criou um aplicativo para a plataforma Android que permite o usuário, frequentador do Interação FURB, movimentar-se através do mapa da FURB de acordo com sua locomoção no mundo real. Para isto, utilizou-se de informações de periféricos como o *Global Positioning System* (GPS). As informações de posicionamento dos usuários serão armazenadas em uma base de dados. Os usuários podem visualizar amigos que estão logados na aplicação através do mapa, visualizar pontos de interesse local cadastrados previamente e pontos de interesse globais cadastrados pela universidade. A aplicação permite também que usuários enviem notificações ao servidor de notificações para permitir a visualização de amigos no mapa.

1.1 OBJETIVOS

O objetivo deste trabalho é criar um aplicativo que possibilite identificar a posição do usuário utilizando mapas e pontos de referência.

Os objetivos específicos do trabalho são:

a) utilizar de pontos de interesse globais e locais para auxiliar a localização do usuário no mapa;
b) possibilitar a manutenção de uma lista de amigos com suas respectivas localizações no mapa;
c) sincronizar as informações de localização, pontos globais e locais, bem como a lista de amigos.

1.2 ESTRUTURA

Este trabalho está estruturado em quatro capítulos. O primeiro capítulo apresenta a introdução do trabalho juntamente com os objetivos gerais e específicos do mesmo. O capítulo dois contém a fundamentação teórica necessária para permitir um melhor

entendimento sobre as tecnologias empregadas neste trabalho. O capitulo três apresenta o desenvolvimento do aplicativo proposto, apresentando os principais requisitos do problema proposto e a especificação contendo diagrama de casos de uso, diagrama de classes e diagrama de sequência. Neste capítulo foi apresentado também às ferramentas utilizadas na especificação e implementação. Foi apresentado também os resultados e discussões. O capítulo quatro apresenta as conclusões do trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é apresentado um pouco de como a FURB foi criada e o Interação FURB. Na seção 2.2 são apresentados conceitos de localização, mapas, geolocalização, GPS, coordenadas Universal Transversa de Mercator (UTM) e cartografia. Na seção 2.3 são apresentados conceitos de geolocalização, mas para a plataforma Android. Por fim, a seção 2.4 apresenta três trabalhos correlatos.

2.1 INTERAÇÃO FURB

O início das atividades da FURB foi em 1964 com a Faculdade de Ciências Econômicas. Em 1968 foi fundada a Fundação Universidade Regional de Blumenau. Em 1986, o Ministério da Educação reconhece e credencia a Universidade. A partir de 21 de março de 1995, pela Lei Complementar Municipal nº 80, a Universidade Regional de Blumenau figura como uma Instituição de Ensino Superior criada e mantida pela Fundação Universidade Regional de Blumenau (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014a).

Desde 2007 a FURB tem realizado o evento Interação FURB. Este evento tem como principal objetivo aproximar o aluno do ensino médio por meio de informações específicas para o futuro profissional e educacional, bem como ações que tornem possível ampliar a interação entre a FURB e os alunos. O evento ocorre uma vez por ano e possui espaço próprio no site da instituição (ANGELI, 2010). A realização do evento no dia 10 de setembro de 2014 contou com a presença de 164 escolas de quase 50 cidades, resultando na inscrição de 17.154 alunos e para atender a esta demanda foi realizadas 432 oficinas. Para agregar no relacionamento entre universidade e futuros alunos foram criados também o Interação Júnior, realizado nos dias 6 e 7 de novembro de 2014 visando contato com alunos do sexto ao nono ano do ensino fundamental e o Interação Kids, que visa contato com alunos de 3 a 6 anos de idade.

A organização do Interação FURB realiza visitas em diversas escolas que possuem ensino médio. As visitas ocorrem entre março e agosto, onde ocorre uma conversa com os alunos. Neste momento é apresentada a estrutura da FURB para os alunos (ANGELI, 2010), que possibilita mostrar os diversos espaços físicos da universidade, muitas vezes desconhecido, fazendo com que os visitantes se locomovam entre salas, blocos e campus. Toda esta movimentação exige certa infraestrutura de sinalização, pessoas guias, entre outros, para que a logística de mudança de espaço físico seja possível de ocorrer no curto período de tempo (um dia).

2.2 GEOLOCALIZAÇÃO (GPS, CARTOGRAFIA, COORDENADAS UTM)

A cartografia é a ciência da representação gráfica da superfície terrestre, tendo como produto final o mapa. Na cartografia, as representações de área podem ser acompanhadas de diversas informações, como símbolos, cores, entre outros elementos (SÓ GEOGRAFIA, 2014).

Uma das artes mais antigas da cartografia é a Geolocalização. Antes do advento dos mapas, antigos viajantes descobririam a sua posição com base nas estrelas, já explorando conceitos relacionados com a Geolocalização. Isto provou ser uma ciência inexata, devido à relativamente baixa precisão, dependência de céu claro, e a mudança do tempo a noite com base no local e época (GRAZIADIO E-LEARNING, 2011).

A tecnologia em relação a Geolocalização entrou pela primeira vez na equação do *Global Positioning System* (GPS) na década de 1920, quando os sinais de rádio terrestres calcularam a direção e a distância de navios. O conceito de usar satélites para GPS veio por cortesia dos russos após o lançamento do Sputnik, em 1957. Dois físicos, William e George Guier Weiffenbach, perceberam que eles poderiam identificar a localização do satélite russo através da gravação de sinais de frequência de rádio e corrigindo para o efeito Dopplar, em que frequência de rádio é maior quando um objeto está se aproximando e menor que o objeto se afasta (GRAZIADIO E-LEARNING, 2011).

Outro conceito relacionado com Geolocalização refere-se ao Universal Transversa de Mercator (UTM), que é um sistema de coordenadas baseado no plano cartesiano (eixo x,y), e usa o metro (m) como unidade para medir distâncias e determinar a posição de um objeto. O sistema UTM não acompanha a curvatura da Terra e por isso seus pares de coordenadas também são chamados de coordenadas planas (GEOREFERENCE, 2014).

2.3 ANDROID E OS FRAMEWORKS (MAPAS, GPS E NOTIFICAÇÃO)

A utilização de GPS em aparelhos celulares não é algo recente, desde sua primeira utilização até os dias de hoje muita coisa evoluiu. Apenas aplicativos instalados em alguns aparelhos celulares com Location API e Java ME podiam utilizar tais recursos (DEVMEDIA, 2014). Hoje, na plataforma Android, para utilizar os recursos de mapas e GPS, é necessário utilizar a API de mapas fornecida pelo Google incluída no pacote `com.google.maps` (DEVMEDIA, 2014).

Este pacote, por sua vez, possui uma classe que engloba os sensores que a plataforma controla que é a classe Sensor. Esta classe possui as identificações dos sensores que serão

utilizados para associar o sensor a classe de gerenciamento dos sensores, a classe `SensorManager`. Através dela é possível associar uma classe `listener` ao sensor desejado. Está classe `listener` deve implementar a interface `SensorEventListener`. Quando o sensor gerar um evento, a classe manager disparará um evento e enviará a classe `listener` que será responsável por tratar os eventos gerados que são denominados `SensorEvent`. A classe `SensorEvent` possui o método `values` que é responsável por retornar os valores capturados pelo sensor configurado inicialmente na classe `listener`. Lembrando que os valores retornados sempre serão retornados em valores específicos de acordo com o tipo de sensor configurado (ANDROID DEVELOPERS, 2014a).

Já em relação ao tratamento de notificações para a distribuição de informações entre os dispositivos móveis a Google criou o *Google Cloud Message* (GCM). Este serviço permite que notificações sejam enviadas diretamente para os dispositivos que têm o seu aplicativo instalado. Na Google I/O 2013, conferencia onde a Google apresenta novos serviços e tecnologias, foi apresentada uma nova funcionalidade para o GCM, receber notificações dos dispositivos móveis. Esta nova função foi denominada de *Cloud Connection Server* (CCS) (ANDROID DEVELOPERS, 2014b).

Por sua vez, a API de notificações *Google Cloud Message* é usada nos dispositivos móveis através da biblioteca `google_play_service`. Isto permite que a aplicação cliente instancie um objeto `GoogleCloudMessage` e realizem o cadastro do dispositivo móvel no servidor *cloud* da Google. Este cadastro gera o identificador de registro utilizado para assinar as notificações. Para que o servidor da aplicação receba estas notificações é necessário implementar o conceito Cloud Connection Server, que permite ao servidor receber as notificações enviadas pela aplicação cliente. Conforme apresentado por Android Developers (2014c), esta implementação utiliza-se uma conexão *eXtensible Messaging and Presence Protocol* (XMPP) para comunicar-se com o servidor *cloud* da Google. Porém, é necessário uma troca de mensagens padronizadas entre os servidores e para isto, implementa-se uma classe que estenda o comportamento da classe `DefaultPacketExtension` para padronizar o envio de pacotes na conexão XMPP.

2.4 TRABALHOS CORRELATOS

Esta seção traz informações sobre trabalhos correlatos a esta proposta, onde serão apresentadas as principais características que se assemelham aos objetivos deste trabalho. Dentre estes, foi incluído o trabalho de KUEHL (2012); a aplicação feita em HARVARD

(2014); o trabalho de ANGELI (2010) foi incluído também, pois nele são tratados assuntos como Interação FURB e a representação gráfica de um Avatar em um mundo virtual.

2.4.1 Protótipo de sistema móvel na plataforma Android para compartilhamento de arquivos e mensagens entre dispositivos baseado em proximidade geográfica

O trabalho de Kuehl (2012) disponibilizou um protótipo de sistema para a plataforma Android capaz de listar vários usuários que estejam próximos geograficamente, permitindo a troca de mensagens e arquivos entre os mesmos. A sua principal contribuição foi permitir que o usuário visualize os dados dos usuários próximos a ele e os arquivos compartilhados por estes. O usuário seleciona os arquivos que deseja compartilhar e estes são copiados para o *webservice*. O *download* de arquivos entre os usuários é feito quando o usuário solicita o *download* de um arquivo e a aplicação solicita o *download* deste arquivo no *webservice*. A localização do usuário é obtida através do GPS e de localização via torres de telefonia para então ser atualizada na base de dados do servidor da aplicação através do *webservice*. Desta forma a posição de todos os usuários fica visível para o servidor e o mesmo poderá processar as informações quando solicitado.

A Figura 1 apresenta uma das interfaces deste trabalho, aonde se pode corrigir a distância limite de visualização dos usuários e com isto aumentar ou diminuir o seu campo de visão. Ao selecionar o usuário desejado, serão mostrados os arquivos compartilhados pelo usuário e caso seja solicitado, será realizado o *download* do arquivo. Na mesma Figura 1, identifica-se o ícone de troca de mensagens entre os usuários na opção conversa (`Conversas`). Nesta opção, o sistema disponibiliza uma interface simplificada para a troca de mensagens entre dois usuários.

Figura 1– Interface de visualização de proximidade

Fonte: Kuehl (2012, p. 100).

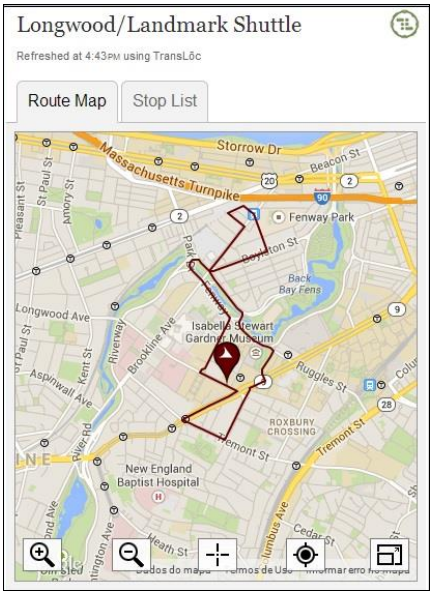
2.4.2 Harvard Mobile Web application

A aplicação Harvard Mobile Web possui a funcionalidade de auxiliar na verificação dos horários de ônibus. Onde é possível ver em um mapa, a posição atual do ônibus e ainda ver a que horas ele passou por cada ponto. Harvard Mobile é uma iniciativa móvel em toda a universidade para agregar e entregar conteúdo móvel útil, utilizável e apropriado para a comunidade da Universidade de Harvard, a nível local e mundial (HARVARD, 2014). A aplicação reúne várias funções de fácil utilização como: novidades que ocorrem na Universidade, informações de cursos, informações de esportes, eventos, cardápio das lanchonetes e informações de ônibus, entre outras funções. A aplicação está disponível para iPhone/iPod Touch que tenham o iOS na versão 4.0 ou superior e para Android com versão superior à 2.2.

Com esta aplicação é possível fazer um caminho em torno do campus através da visualização de mapas em tempo real de ônibus e rotas, levantando-se atualizado anúncios de mudanças no serviço, e visualização de horários de ônibus e calendários. Também estão disponíveis listas de telefone para serviço de transporte, estacionamento e escritórios suburbanos (HARVARD, 2014). A função de informações de ônibus deste sistema mostra no mapa da cidade, a localidade onde o ônibus selecionado transita. É grifada em vermelho a rota

que o veículo percorre como pode ser visto na Figura 2. Com isto, é atualizado de tempos em tempo o local onde o ônibus está. Desta forma, é possível saber onde o mesmo está e se há tempo de chegar até o ponto de parada mais próximo.

Figura 2 – Tela de informações de ônibus



Fonte: Harvard (2014).

Conforme a Figura 2, a aplicação apresenta um mapa com as informações do ônibus. É possível movimentar o mapa para poder visualizar pontos de interesse ao redor. É possível também, dar *zoon in* e *zoon out* no mapa.

2.4.3 Protótipo de mundo virtual para relacionamento com participantes do Interação FURB

O sistema desenvolvido por Angeli (2010) consiste em disponibilizar um ambiente virtual que permita o encontro dos participantes do Interação FURB, provendo também informações sobre a estrutura da Universidade, laboratórios, projetos e cursos através de vídeos, páginas da internet ou painéis que estarão disponíveis no ambiente virtual. Foi desenvolvido também, um módulo de gerenciamento do ambiente virtual, onde os administradores poderão adicionar ou remover objetos do mundo virtual. Através deste módulo de gerenciamento é possível definir a configuração das estruturas do ambiente virtual, a localização de salas, pátios ou prédios.

O controle de acesso e permissões será controlado pelo módulo de segurança. As informações de acesso são salvas em uma base de dados. Cadastros são pré-determinados de acordo com o cadastro da FURB. Novos usuários poderão se cadastrar com um usuário, mas este será excluído após um determinado tempo.

Para o desenvolvimento do módulo do servidor, foi utilizado a ferramenta OpenWonderland. O OpenWonderland é uma ferramenta *open source* e seu principal objetivo é possibilitar a criação de mundos virtuais 3D para ambientes empresariais, onde os usuários podem comunicar-se, compartilhar dados ou realizar reuniões virtuais (ANGELI, 2010, p. 23). Porém, foi necessário realizar alterações no código fonte do OpenWonderland e foi necessário trocar diversos *e-mails* com a equipe de desenvolvimento da ferramenta.

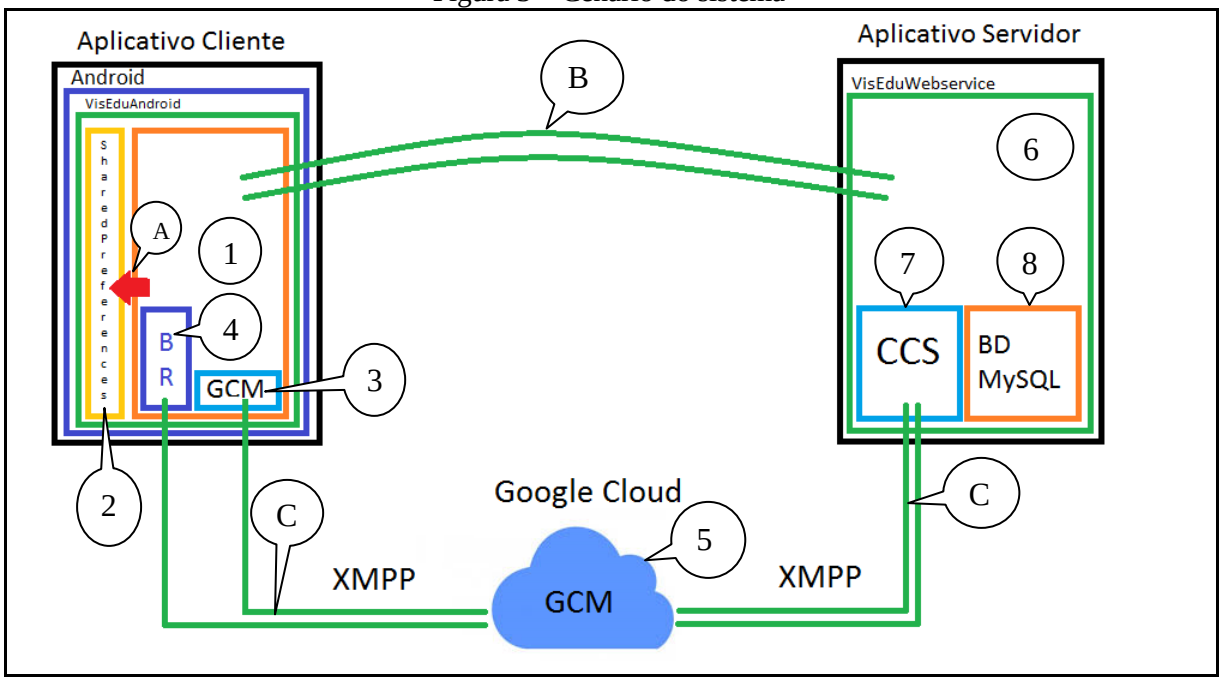
3 DESENVOLVIMENTO DO APLICATIVO

Neste capítulo são apresentadas as etapas de desenvolvimento do aplicativo móvel.A primeira seção apresenta o sistema proposto. A segunda seção apresenta os principais requisitos do problema trabalhado. A terceira seção descreve a especificação da solução através de diagramas da *Unified Modeling Language* (UML) e de Modelos de Entidades e Relacionamentos (MER). Na quarta seção é descrito a implementação do trabalho, juntamente com a operacionalidade do aplicativo. Por fim, a quinta seção descreve os resultados e discussões deste trabalho.

3.1 APLICATIVO PROPOSTO

O aplicativo proposto tem como objetivo interagir com o usuário na plataforma Android e manter os dados dos usuários em um lugar para que sejam acessíveis a qualquer momento pelo mesmo. Para isto foi necessário criar dois projetos distintos. No primeiro projeto, o aplicativo cliente foi desenvolvido de forma dependente ao segundo projeto. O segundo projeto consistem em prover suporte ao primeiro de forma que o acesso ao banco de dados é realizado através de um *webservice*. Para um melhor entendimento é apresentada a Figura 3.

Figura 3 – Cenário do sistema



A arquitetura do projeto apresentada na Figura 3 possui os seguintes pontos relevantes: a aplicação cliente denominada `VisEduAndroid` (1), a área de recursos compartilhados (2), o serviço Google Cloud Message (GCM) na aplicação cliente (3), o serviço

BroadcastReceiver configurado para receber notificações (4), o próprio servidor GCM da Google (5), o aplicativo servidor VisEduWebservice (6), o serviço Cloud Connection Server (CCS) no servidor (7) e o armazenamento das informações na base de dados MySQL no servidor (8). O aplicativo cliente salva o gcmId, que é o registro de identificação do aparelho no GCM, na SharedPreferences do Android, para que posteriormente ele utilize este mesmo identificador sem a necessidade de realizar um novo registro (A). O aplicativo VisEduAndroid envia e recebe informações do aplicativo VisEduWebservice através do *webservice* construído para realizar a comunicação entre o aplicativo cliente e o aplicativo servidor (B). O aplicativo VisEduAndroid identifica a necessidade de registrar o aparelho no GCM e inicia o processo requisitando o identificador. Ao receber o identificador, guarda o valor na SharedPreferences do Android e envia via notificação o valor para o servidor VisEduWebservice. O primeiro projeto desenvolvido foi o VisEduAndroid, conforme representado na legenda (1) da Figura 3, responsável por apresentar os dados ao usuário e trata os eventos ocorridos para enviar os dados do aplicativo para o servidor *webservice* (B). O servidor *webservice* por sua vez, representa o segundo projeto, que disponibiliza acesso ao banco de dados conforme representado na legenda (6) da Figura 3.

O aplicativo cliente foi desenvolvido em Java utilizando o Android SDK na versão 19, no ambiente de desenvolvimento Eclipse Luna. A biblioteca google_play_services.jar foi utilizada para usar o Google Maps Android API v2, necessário para acessar o Google Maps dentro do aplicativo, e também o Google Cloud Messaging for Android, necessário para enviar e receber notificações.

O servidor foi desenvolvido na versão 8 da linguagem Java, no ambiente de desenvolvimento Eclipse Luna. Como servidor de aplicação foi utilizado o Apache Tomcat v7.0. O acesso ao banco de dados foi efetuado através do Java Database Connectivity (JDBC) do banco de dados MySQL versão 5.6.

O aplicativo cliente possui a implementação do Google Cloud Message (GCM) para que seja possível enviar notificações ao servidor e o servidor possui a implementação do Cloud Connection Server (CCS) necessário para que o servidor possa enviar e receber notificações dos dispositivos móveis.

O aplicativo cliente permite que o usuário cadastre informações que sejam relevantes para o mesmo como, pontos de interesse local, que é um ponto que o usuário considera importante saber a localização. O usuário pode criar uma relação de amizade com outro

usuário e assim visualizar este usuário no mapa, caso o mesmo permita a visualização dele no mapa.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Serão listados a seguir, os requisitos funcionais e não funcionais da aplicação da parte cliente e da parte servidor.

Os Requisitos Funcionais (RFs) do aplicativo cliente são:

a) RF01: o aplicativo deverá permitir que os usuários solicitem a permissão de visualizar o amigo;
b) RF02: o aplicativo deverá permitir que os usuários permitam ser visualizados por seus amigos de interesse;
c) RF03: o aplicativo deverá permitir o acesso dos usuários sem vinculo estudantil com a universidade;
d) RF04: o aplicativo deverá permitir que o usuário cadastre e visualize pontos de interesse local;
e) RF05: o aplicativo deverá permitir que o usuário visualize pontos de interesse global;
f) RF06: o aplicativo deverá permitir que o usuário cadastre e visualize suas amizades;
g) RF07: o aplicativo deverá permitir que o usuário visualize pontos de interesse globais no mapa;
h) RF08: o aplicativo deverá permitir que o usuário visualize sua posição atual no mapa;
i) RF09: o aplicativo deverá permitir que o usuário visualize seus pontos de interesse cadastrados, no mapa;
j) RF10: o aplicativo deverá permitir que o usuário visualize seus amigos no mapa.

Os Requisitos Não Funcionais (RNFs) do aplicativo cliente são:

a) RNF01: o aplicativo deverá ser desenvolvido em Java, na plataforma Android;
b) RNF02: o aplicativo deve suportar Android’s das versões 2.3 a 4.4;
c) RNF03: o aplicativo deve usar o Google Maps Android API v2.

Os Requisitos Funcionais (RFs) do aplicativo servidor são:

a) RF11: o aplicativo servidor deverá manter os registros de pontos de interesse local;
b) RF12: o aplicativo servidor deverá manter os registros de amizade;

c) RF13: o aplicativo servidor deverá manter os registros de pontos de interesse globais.

Os Requisitos Não Funcionais (RNFs) do aplicativo servidor são:

a) RNF04: o aplicativo servidor deverá ser desenvolvido em Java na forma de *webservice*;
b) RNF05: o aplicativo servidor deverá acessar uma base MySQL;
c) RNF06: o aplicativo servidor devera implementar uma classe responsável por receber notificações denominada GCM CCS.

3.3 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando diagramas da UML aliados ao MER. Os casos de uso são apresentados através de um diagrama de casos de uso, seguido da descrição do cenário de cada caso de uso. Em seguida, as classes da aplicação cliente e do servidor são apresentadas através de diagramas de classes. Complementando a especificação, as entidades do banco de dados do servidor são demonstradas através de diagramas MER. A ferramenta utilizada para desenvolvimento dos diagramas da UML foi o Enterprise Architect (EA) versão 11.1.1112 para Windows 7.

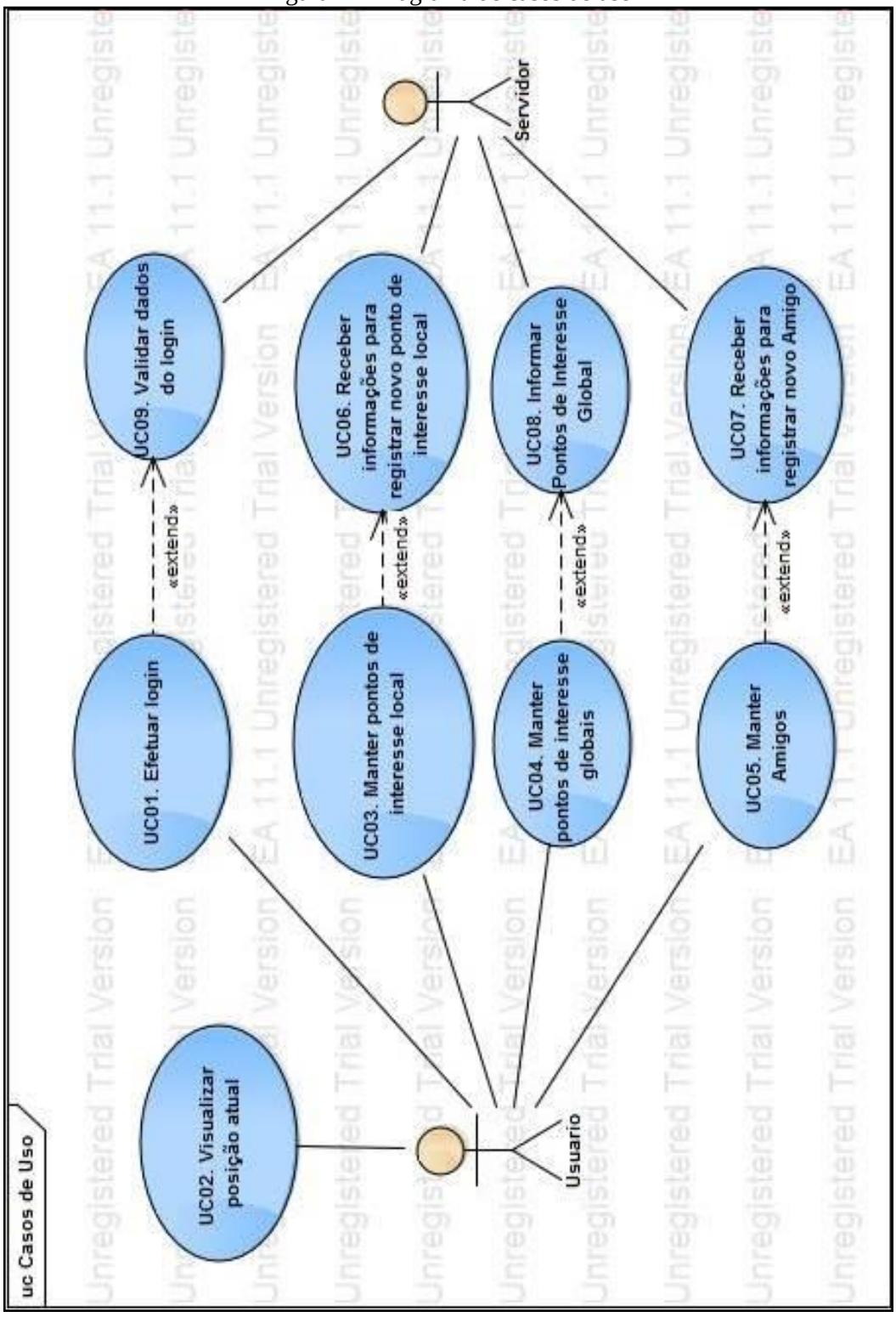
3.3.1 Casos de uso

A partir da elicitação dos requisitos do aplicativo, foram desenvolvidos onze casos de uso. Esses casos de uso objetivam organizar os requisitos em funcionalidades que possam ser executadas de forma simples pelo usuário.

Os casos de uso são desempenhados por três atores. O ator `Usuário` representa o utilizador do aplicativo, interagindo através dos casos de uso que apresentam interface gráfica. O ator `GCM` representa a parte dos dois aplicativos, cliente e servidor, que faz o envio e recebimento de notificações. O ator `Servidor` representa o aplicativo servidor e é este, o responsável por manter o banco de dados e responder as requisições do aplicativo cliente.

O diagrama de casos de uso foi desenvolvido observando os padrões da UML. Como pode ser visto na Figura 4, os casos de uso estão associados aos atores responsáveis pela ação no caso do uso.

Figura 4 – Diagrama de casos de uso



3.3.1.1 Efetuar login

Deve ser efetuado o *login* na aplicação para que seja possível acessar as funções disponíveis. O caso de uso `Efetuar login` tem como objetivo autenticar o usuário através do usuário e senha. O Quadro 1 contém os cenários do caso de uso `Efetuar login`.

Quadro 1 – Caso de uso `Efetuar login`

UC01. Efetuar *login*: possibilita ao usuário efetuar *login* na aplicação.	
Requisitos atendidos	RF03.
Pré-condições	
Cenário principal	1) o sistema exibe a tela inicial do sistema solicitando usuário e senha, 2) o usuário digita usuário e senha e clica em *login*, 3) o sistema conecta-se com a aplicação servidor solicitando autenticação, enquanto exibe mensagem de espera.
Fluxo alternativo 01	No passo 2, caso o usuário não possua cadastro: 2) usuário digita nome do usuário desejado e senha desejada e clica em registrar, 3) o sistema envia informações ao servidor, 4) servidor cadastra o usuário.
Fluxo alternativo 02	No passo 3, caso os dados sejam incorretos: sistema mostra mensagem de alerta informando dados inválidos.
Pós-condições	Dados da autenticação são retornados e o sistema abre o mapa principal.

3.3.1.2 Visualizar posição atual

O caso de uso `Visualizar posição atual` apresenta o mapa ao usuário. Nele será mostrado a posição atual do usuário através do GPS, os seus amigos, os pontos de interesse globais e locais. O Quadro 2 contém os cenários do caso de uso `Visualizar posição atual`.

Quadro 2 – Caso de uso `Visualizar posição atual`

UC02. Visualizar posição atual: possibilita ao usuário visualizar sua posição, seus amigos e os pontos de interesse globais e locais.	
Requisitos atendidos	RF07, RF08, RF09 e RF10.
Pré-condições	
Cenário principal	1) o sistema exibe no mapa a posição atual do usuário, 2) o sistema exibe no mapa a posição dos amigos que permitiram serem visualizados, 3) o sistema exibe no mapa os pontos de interesse globais, 4) o sistema exibe no mapa os pontos de interesse locais.
Pós-condições	O mapa carregado com as informações do usuário.

3.3.1.3 Manter pontos de interesse local

O caso de uso `Manter pontos de interesse local` apresenta uma tela com os registros de pontos de interesse local, que são os pontos cadastrados pelo próprio usuário. Através desta tela o usuário poderá alterar ou excluir os pontos locais. O Quadro 3 contém os cenários do caso de uso `Manter pontos de interesse local`.

Quadro 3 – Caso de uso `Manter pontos de interesse local`

UC03. Manter pontos de interesse local: possibilita ao usuário manter os pontos de interesse local, editá-los ou remove-los.	
Requisitos atendidos	RF04.
Pré-condições	
Cenário principal	1) o sistema mostra a tela com a lista de pontos de interesse locais, 2) o usuário seleciona um ponto, 3) o sistema carrega uma nova tela com os dados do ponto selecionado, 4) o usuário realiza a alteração e clica no botão para salvar as alterações, 5) o sistema fecha a tela e volta para a lista de pontos.
Fluxo alternativo 01	Após o passo 1, caso o usuário queira excluir o ponto: 2) o usuário clica no botão de exclusão de ponto no registro desejado, 3) o sistema atualiza a lista de pontos de interesse locais.
Pós-condição	O sistema atualiza a lista de pontos conforme ação do usuário.

3.3.1.4 Visualizar pontos de interesse globais

O caso de uso `Visualizar pontos de interesse globais` apresenta uma tela com os registros de pontos de interesse globais, que não são cadastrados pelo usuário. Estes pontos serão mantidos pela FURB. O Quadro 4 contém os cenários do caso de uso `Visualizar pontos de interesse globais`.

Quadro 4 – Caso de uso `Visualizar pontos de interesse globais`

UC04. Visualizar pontos de interesse globais: possibilita ao usuário visualizar os pontos de interesse globais.	
Requisitos atendidos	RF05.
Pré-condições	
Cenário principal	1) o sistema mostra a tela com a lista de pontos de interesse globais, 2) o usuário seleciona o ponto, 3) o sistema carrega uma nova tela com os dados do ponto selecionado.

3.3.1.5 Manter amigos

O caso de uso `Manter amigos` apresenta uma tela com a lista de amigos que o usuário possui. O usuário seleciona um amigo para visualizar as informações do mesmo. O Quadro 5 contém os cenários do caso de uso `Manter amigos`.

Quadro 5 – Caso de uso `Manter amigos`

UC05. Manter amigos: possibilita ao usuário manter as suas amizades.	
Requisitos atendidos	RF01, RF02, RF06.
Pré-condições	
Cenário principal	1) o sistema mostra a tela com a lista de amigos, 2) o usuário seleciona um amigo e o sistema carrega nova tela com os dados do amigo, 3) o usuário visualiza as informações e clica no botão voltar.
Fluxo alternativo 01	No passo 3, caso o usuário clique no *check box* "permitir": 3) o sistema enviará uma notificação para o servidor informando que o usuário permite que o amigo visualize sua posição atual no mapa.
Fluxo alternativo 02	No passo 3, caso o usuário clique no check box "solicitar": 3) o sistema enviará uma notificação para o servidor solicitar que o amigo permita ou não que o usuário visualize sua posição.
Pós-condição	O sistema atualiza as informações sobre a amizade dos usuário.

3.3.1.6 Receber informações para registrar novo Ponto de Interesse Local

O caso de uso `Receber informações para registrar novo Ponto de Interesse Local` define como o servidor deve cadastrar o registro recebido via *webservice*. O Quadro 6 contém os cenários do caso de uso `Receber informações para registrar novo Ponto de Interesse Local`.

Quadro 6 – Caso de uso `Receber informações para registrar novo Ponto de Interesse Local`

UC06. Receber informações para registrar novo Ponto de Interesse Local: define como o servidor deve efetuar o registro da informação recebida via *webservice*.	
Requisitos atendidos	RF11.
Pré-condições	
Cenário principal	1) o sistema recebe a requisição HTTP com as informações do ponto de interesse local, 2) o sistema valida o usuário, 3) o sistema insere o ponto de interesse local no banco de dados, 4) o sistema retorna a requisição HTTP com uma mensagem informando a inserção da informação na base de dados.
Fluxo alternativo 01	No passo 3, caso ocorra erro na inserção do ponto de interesse local na base de dados: 4) o sistema trata a exceção e retorna a requisição HTTP informando erro no processo.
Pós-condição	Base de dados atualizada com o Ponto de Interesse Local.

3.3.1.7 Receber informações para registrar novo Amigo

O caso de uso `Receber informações para registrar novo Amigo` define como o servidor deve cadastrar o registro Amizade via *webservice*. O Quadro 7 contém cenários do caso de uso `Receber informações para registrar novo Amigo`.

Quadro 7 – Caso de uso `Receber informações para registrar novo Amigo`

UC07. Receber informações para registrar novo Amigo: define como o servidor deve efetuar o registro da Amizade recebido via *webservice*.	
Requisitos atendidos	RF12.
Pré-condição	
Cenário principal	1) o sistema recebe a requisição HTTP com as informações da amizade, 2) o sistema valida o usuário e o amigo, 3) o sistema insere o registro da amizade no banco de dados, 4) o sistema retorna a requisição HTTP com uma mensagem informando a inserção da informação na base de dados.
Fluxo alternativo 01	No passo 3, caso ocorra erro na inserção da amizade na base de dados: 4) o sistema trata a exceção e retorna a requisição HTTP informando erro no processo.
Pós-condição	Base de dados atualizada com a Amizade.

3.3.1.8 Informar Pontos de Interesse Globais

O caso de uso `Informar Pontos de Interesse Globais` define como o servidor deve retornar todos os pontos de interesse globais via *webservice*. O Quadro 8 contém os cenários do caso de uso `Informar Pontos de Interesse Globais`.

Quadro 8 – Caso de uso `Informar Pontos de Interesse Globais`

UC08. Informar Pontos de Interesse Globais: define como o servidor deve efetuar o retorno dos Pontos de Interesse Globais via webservice.	
Requisitos atendidos	RF13.
Pré-condição	
Cenário principal	1) o sistema recebe a requisição HTTP solicitando todos os pontos de interesse globais, 2) o sistema realiza a consulta na base de dados, 3) o sistema retorna a requisição HTTP com uma mensagem informando a inserção da informação na base de dados.
Fluxo alternativo 01	No passo 2, caso não existam registros: 3) o sistema trata a exceção e retorna a requisição HTTP informando que não existem registros.
Pós-condição	

3.3.1.9 Validar *login*

O caso de uso `Validar login` define como o servidor deve validar o usuário e senha informados no *login* da aplicação. O Quadro 9 contém os cenários do caso de uso `Validar login`.

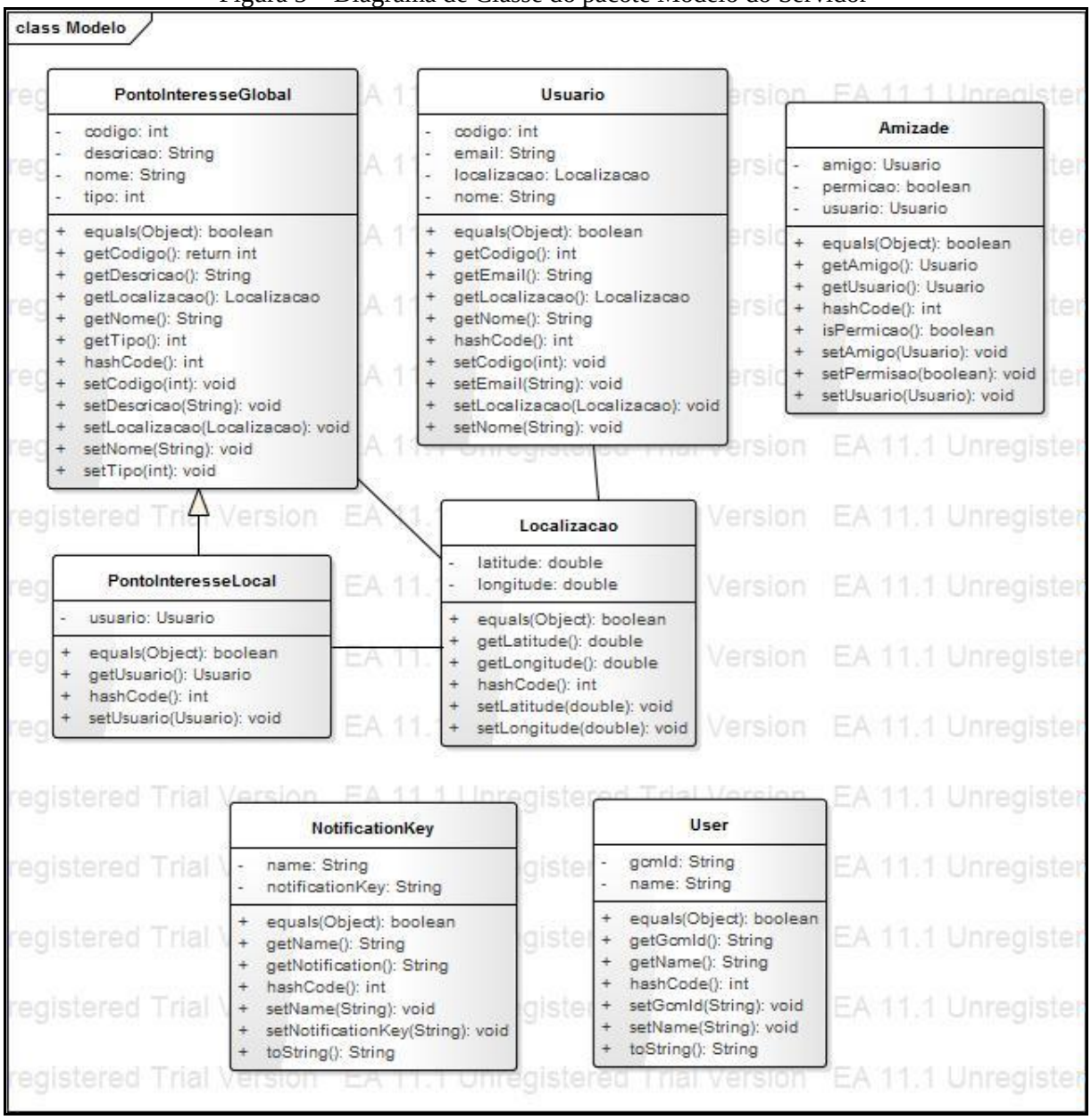
Quadro 9 – Caso de uso `Validar login`

UC09. Validar *login*: define como o servidor deve realizar a autenticação do usuário.	
Requisitos atendidos	RF03.
Pré-condição	
Cenário principal	1) o sistema recebe a requisição HTTP solicitando autenticação do usuário e senha, 2) o sistema valida se existe algum usuário com o nome e senha informados, 3) o sistema retorna a requisição HTTP com o usuário que possui as credenciais.
Fluxo alternativo 01	No passo 2, caso o usuário e a senha não estejam corretos e não seja encontrado nenhum usuário: 4) o sistema retorna a requisição HTTP com uma exceção informando a inexistência de usuário com as credenciais informadas.
Pós-condição	Usuário realiza acesso ao sistema com sucesso.

3.3.2 Classes do aplicativo servidor

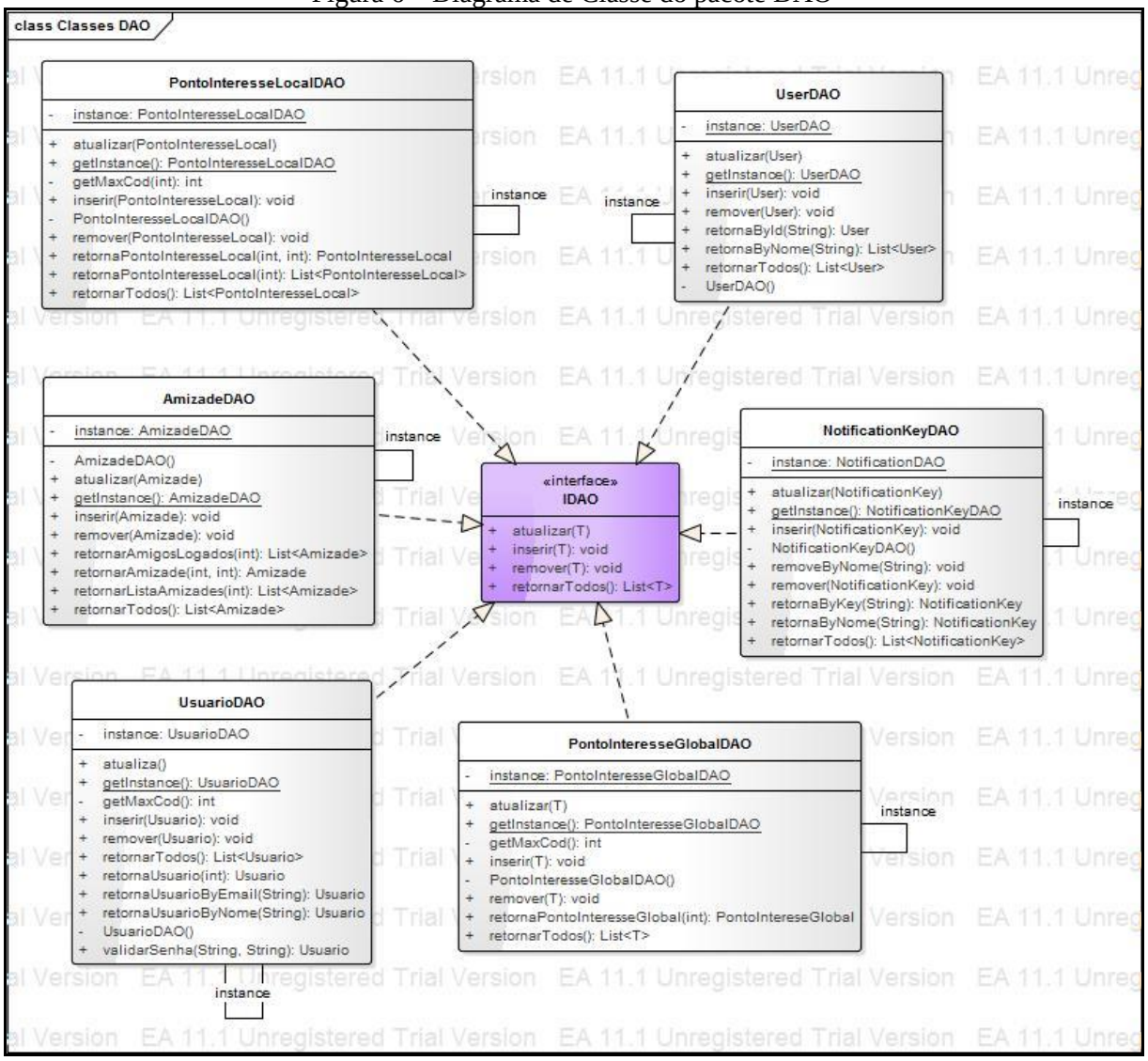
Partindo da definição dos objetivos deste trabalho, o servidor foi desenvolvido para suprir as necessidades do aplicativo cliente quanto a acesso ao banco de dados. As classes do pacote `br.com.visedu.model` que representam objetos que armazenam informações referente aos registros da aplicação são: `Usuario`, `Amizade`, `PontoInteresseGlobal`, `PontoInteresseLocal`, `Localizacao`, `User` e `NotificationKey`. Estes registros são armazenados na base de dados. A Figura 5 apresenta o diagrama de classes do pacote modelo do servidor.

Figura 5 – Diagrama de Classe do pacote Modelo do Servidor



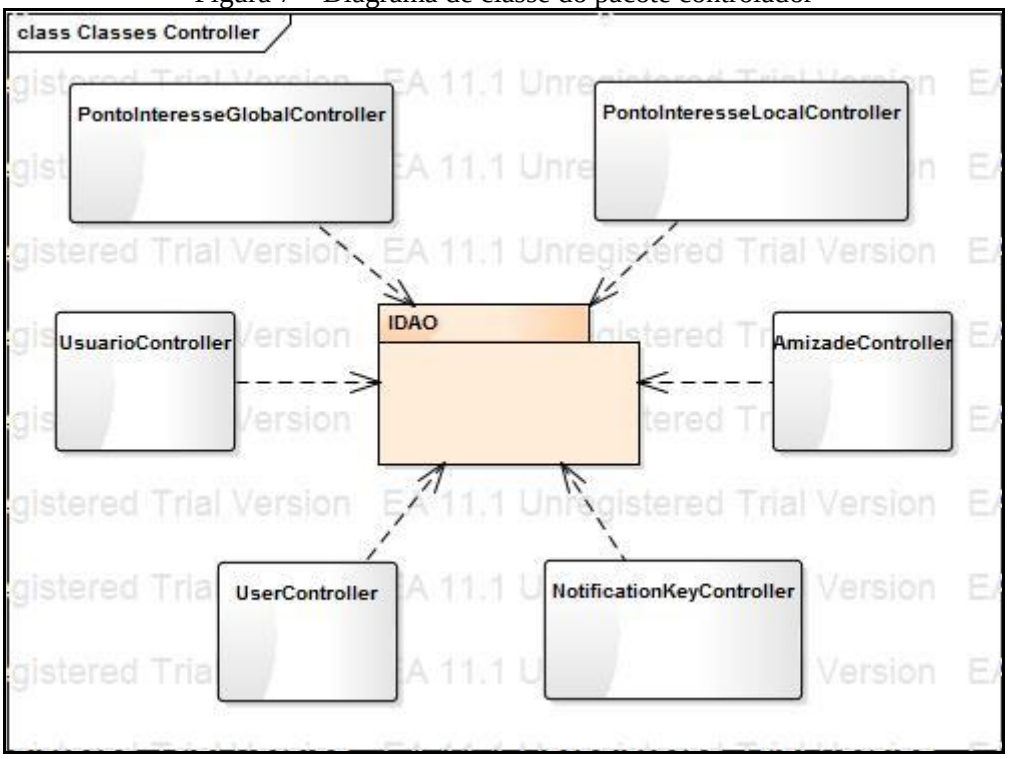
As classes `UsuarioDAO`, `AmizadeDAO`, `PontoInteresseGlobalDAO`, `PontoInteresseLocalDAO`, `UserDAO` e `NotificationKeyDAO` do pacote `br.com.visedu.dao são` responsáveis por acessar o banco de dados e realizar consultas. A Figura 6 apresenta o diagrama de classes do pacote DAO do servidor.

Figura 6 – Diagrama de Classe do pacote DAO



As classes do pacote `br.com.visedu.controller` que representam a camada controladora da aplicação são: `UsuarioController`, `AmizadeController`, `PontoInteresseGlobalController`, `PontoInteresseLocalController`, `UserController` e `NotificationKeyController`. A Figura 7 apresenta o diagrama de classes do pacote controlador do servidor. As classes controladoras usam as suas respectivas classes DAO para fazer acesso ao banco.

Figura 7 – Diagrama de classe do pacote controlador



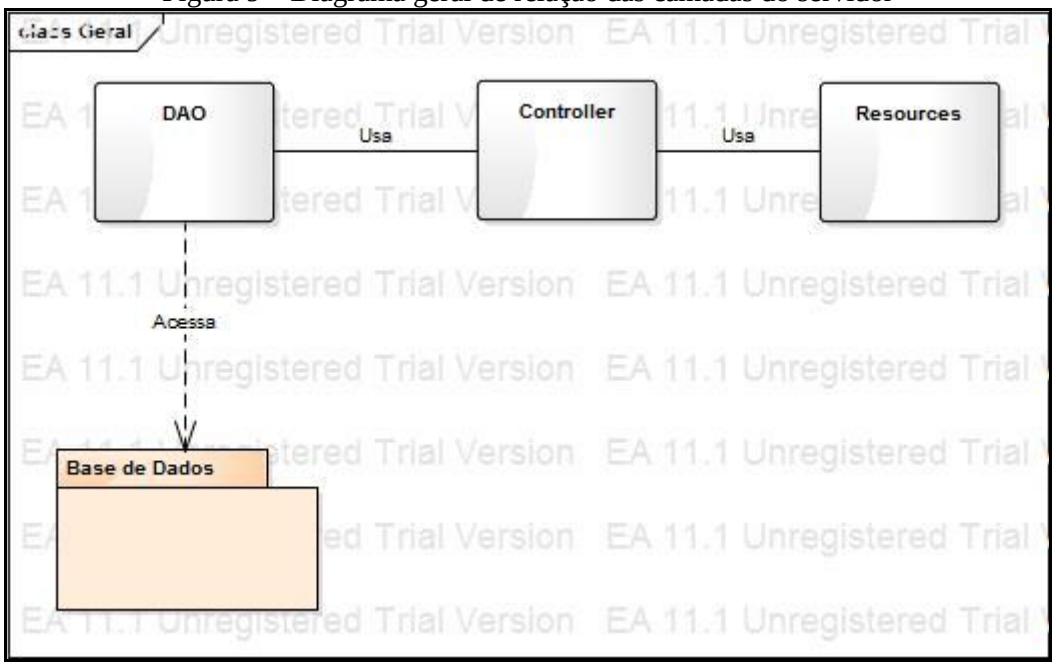
As classes do pacote `br.com.visedu.resources` representam a camada de recursos do aplicativo. Cada recurso necessário no servidor é implementado em forma de classe que operam como *webservices*, que processam as requisições recebidas via *HyperText Transfer Protocol* (HTTP) e retornam objetos em formato JSON. A Figura 8 apresenta o diagrama de classes do pacote de recursos.

Figura 8 – Diagrama de classe do pacote recursos

A Figura 8 representa as classes do pacote `br.com.visedu.resources` que são responsáveis por tratar as requisições HTTP que vem do aplicativo cliente para o servidor *webservice* (conforme legenda B da Figura 3). Esta camada instancia e usa os métodos da camada controladora para executar a lógica necessária. Por sua vez a camada controladora instancia e acessa as respectivas informações da base de dados através da camada DAO. A camada DAO é responsável por instanciar uma conexão com o banco de dados, realizar a consulta e retornar o resultado a camada controladora, que retorna o resultado para a camada recursos para então responder à requisição HTTP. Para que se tenha um entendimento da

relação das classes, foi criado um diagrama que demonstra a relação das camadas do aplicativo servidor (Figura 9).

Figura 9 – Diagrama geral de relação das camadas do servidor

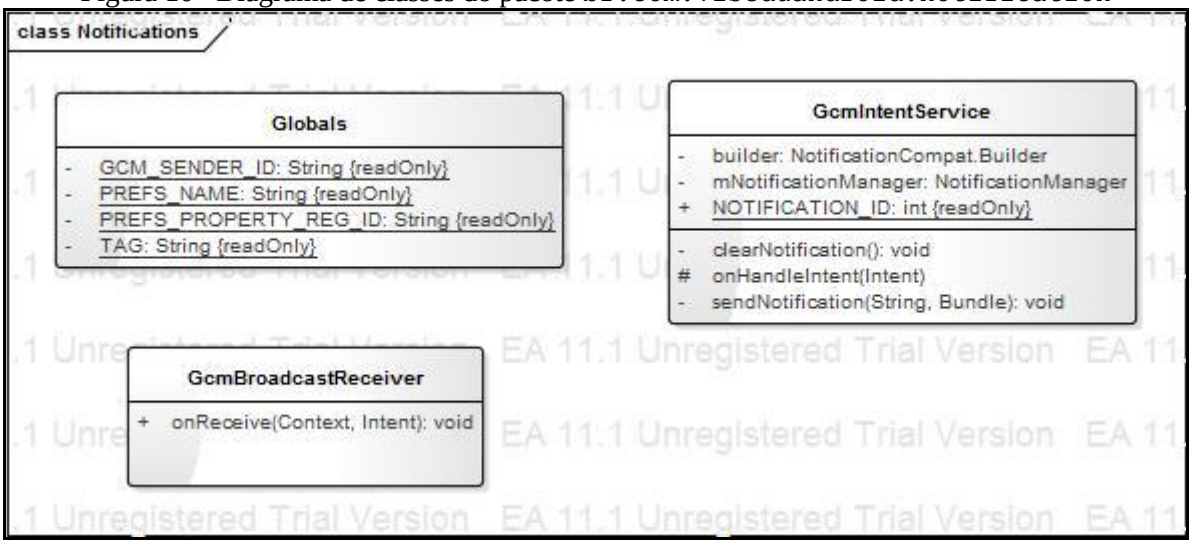


3.3.3 Classes do aplicativo cliente

Partindo da definição dos objetivos deste trabalho, o aplicativo cliente foi desenvolvido para interagir com o usuário, tratar eventos no dispositivo móvel, mostrar os dados ao usuário e buscar as informações na base de dados através do *webservice* desenvolvido para o aplicativo servidor. As classes do pacote `br.com.viseduandroid.model` que representam objetos que armazenam informações referente aos registros da aplicação são: `Usuario`, `Amizade`, `PontoInteresseGlobal`, `PontoInteresseLocal`, `User` e `NotificationKey`. Estes registros são armazenados na base de dados através do *webservice* e por isso são um espelho das classes modelo do aplicativo servidor, conforme Figura 5.

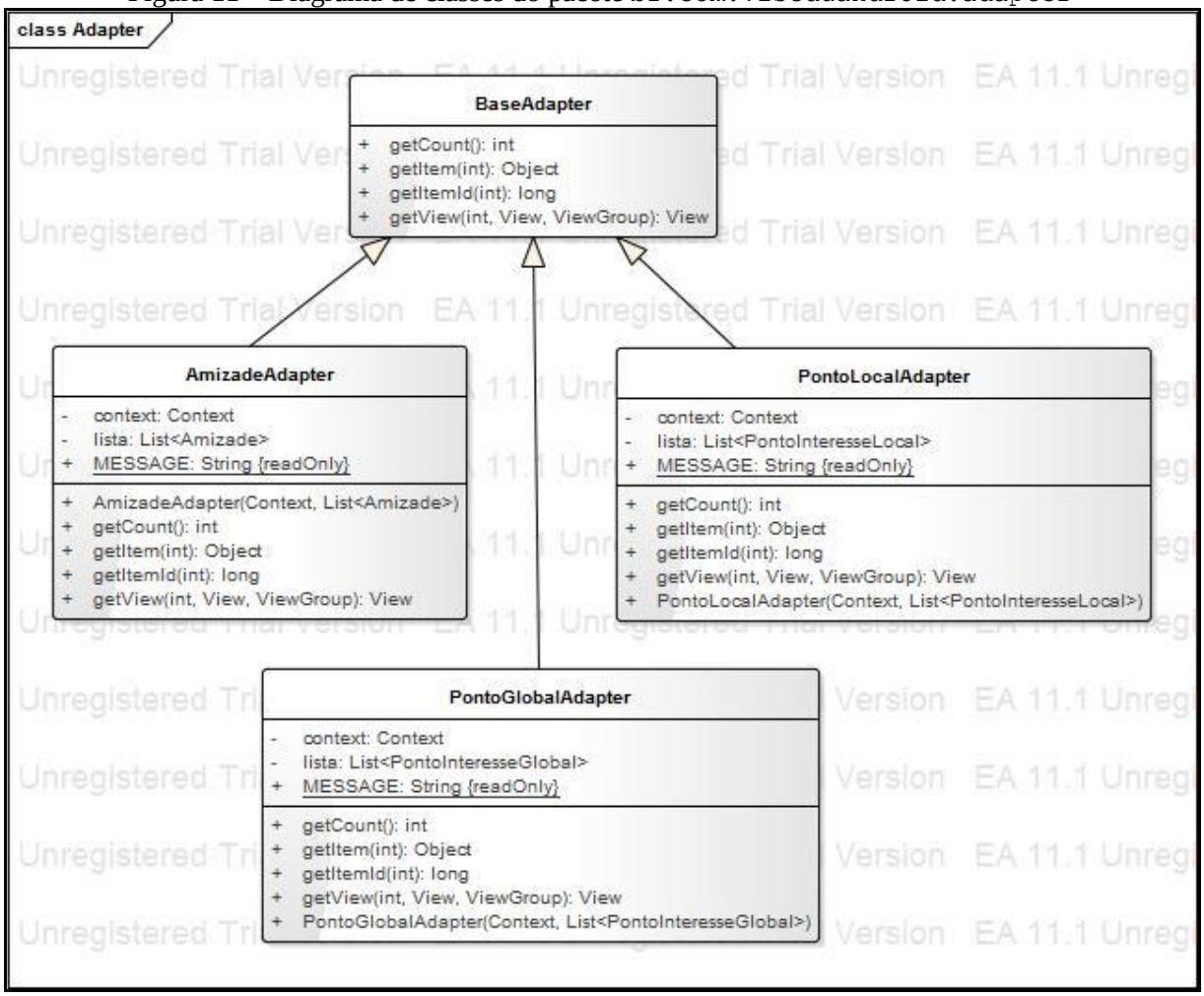
As classes do pacote `br.com.viseduandroid.notification` são configuradas na aplicação para que a mesma consiga tratar notificações enviadas pelo aplicativo servidor, conforme a legenda (`3`) da Figura 3. A classe `Globals` é responsável por conter as constantes utilizadas no processo de notificações. A Classe `GcmBroadcastReceiver` é responsável por receber a notificação, e informar a notificação ao serviço responsável por tratar a informação. A classe `GcmIntentService` é a classe responsável por processar os dados recebidos na notificação. A Figura 10 apresenta o diagrama de classes do pacote.

Figura 10– Diagrama de classes do pacote `br.com.viseduandroid.notification`



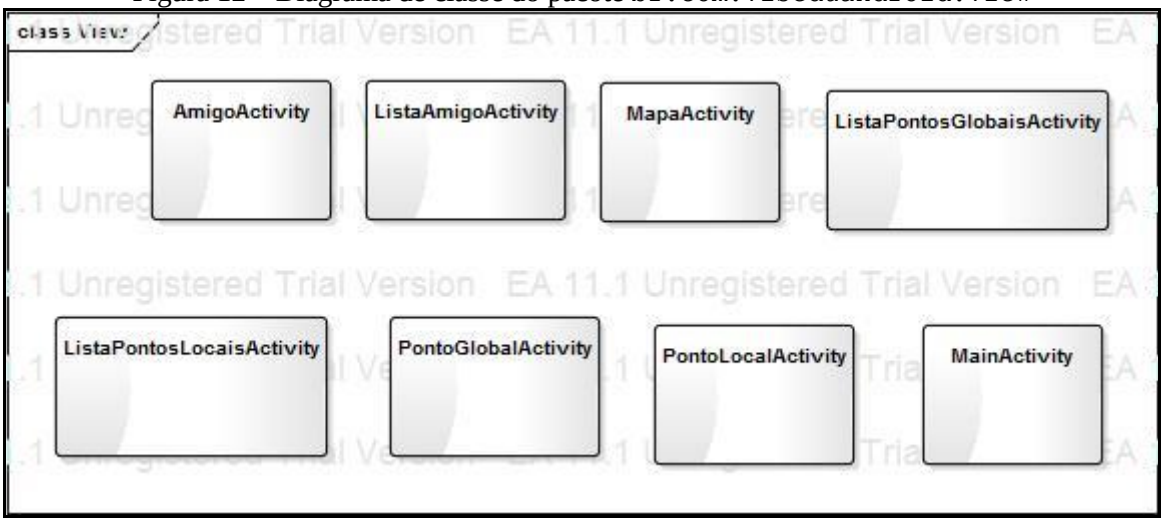
As classes do pacote `br.com.viseduandroid.adapter` são responsáveis por desenhar itens da classe `ListView`. Desta forma é possível estilizar (definir cor, tamanho, orientação dos itens, entre outras coisas) da maneira necessária cada tipo de lista com seus respectivos adaptadores. As classes `AmizadeAdapter`, `PontoGlobalAdapter` e `PontoLocalAdapter` estendem o comportamento da classe `BaseAdapter` para permitir a configuração dos itens da lista de componentes do Android. A Figura 11 apresenta o diagrama de classes do pacote `br.com.viseduandroid.adapter`.

Figura 11 – Diagrama de classes do pacote `br.com.viseduandroid.adapter`



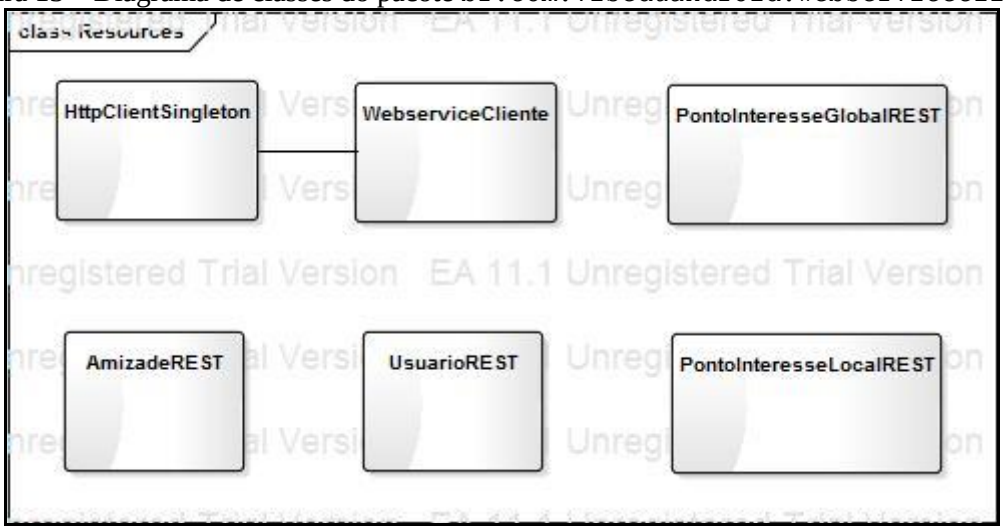
As classes do pacote `br.com.viseduandroid.view` são as classes responsáveis por criar as `Activity's` disponíveis para o usuário. As classes `AmigoActivity`, `ListaAmigoActivity`, `ListaPontosGlobaisActivity`, `ListaPontosLocaisActivity`, `MapaActivity`, `PontoGlobalActivity`, `PontoLocalActivity` e `MainActivity` são as classes de interface com o usuário que tratam os eventos gerados pelo usuário. A Figura 12 apresenta o diagrama de classes do pacote `br.com.viseduandroid.view`.

Figura 12 – Diagrama de classe do pacote `br.com.viseduandroid.view`



As classes do pacote `br.com.viseduandroid.webserviceclient` são responsáveis por realizar as requisições HTTP's no aplicativo servidor. Para isto, foi implementado as classes `HttpClientSingleton` e `WebserviceCliente`, responsáveis por criar uma conexão HTTP, configurá-la e implementar as requisições `GET` e `POST` respectivamente. Este pacote representa a troca de informação ilustrada na Figura 3 (`B`). As classes `AmizadeREST`, `PontoInteresseGlobalREST`, `PontoInteresseLocalREST` e `UsuarioREST` permitem a aplicação cliente comunicar-se com o aplicativo servidor. A Figura 13 apresenta o diagrama de classes do pacote `br.com.viseduandroid.webserviceclient`.

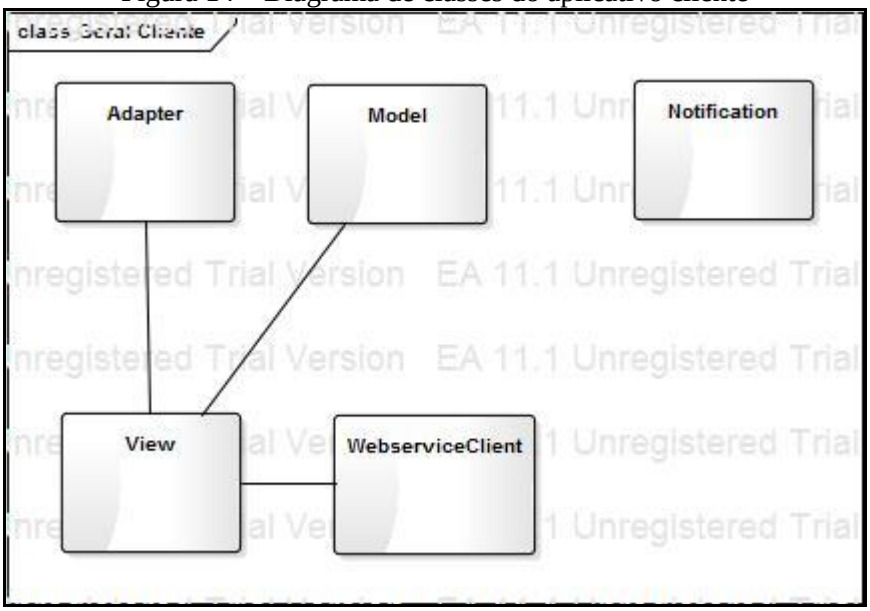
Figura 13 – Diagrama de classes do pacote `br.com.viseduandroid.webserviceclient`



A camada `model` da aplicação é independente, usada para modelar os objetos usados como representação da informação da base de dados. A camada `notification` é independente, usada para configurar o comportamento da aplicação quando a mesma recebe uma notificação. A camada `webserviceclient` é independente, mas é utilizada pela camada `view` para buscar informações no servidor. A camada `adapter` é dependente da camada `view`,

onde que a camada `view` popula a lista de itens da `ListView` de acordo com a configuração do `adapter` escolhido. Por sua vez, a camada `view` é a camada principal do aplicativo cliente, pois, é onde fica a interface com o usuário e a partir dela são iniciadas as chamadas no *webservice*. A Figura 14 apresenta o diagrama de classes da aplicação cliente onde é possível identificar a relação entre as camadas apresentadas anteriormente.

Figura 14 – Diagrama de classes do aplicativo cliente



3.3.4 Diagrama de sequência

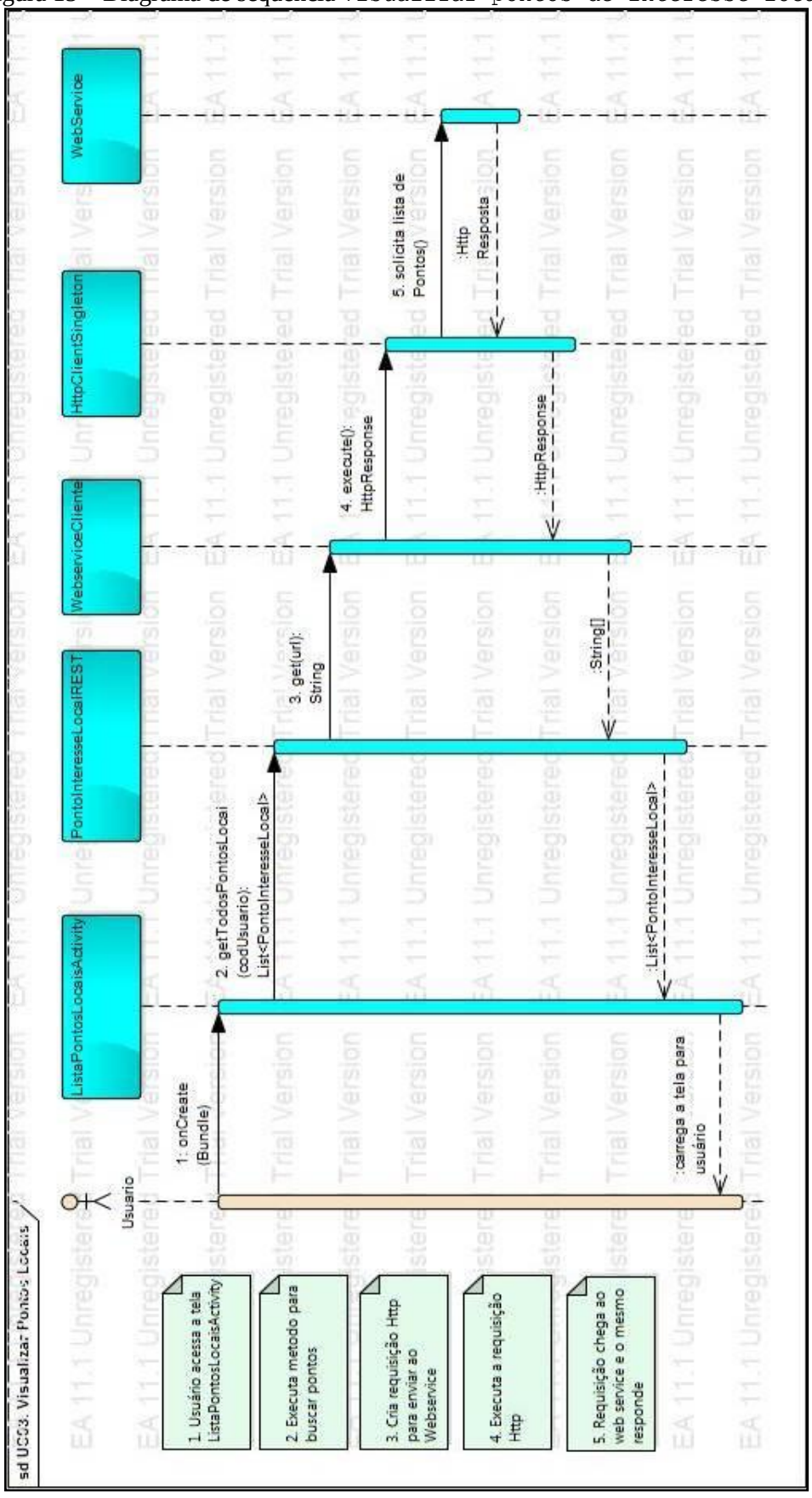
Os diagramas de sequência têm como finalidade demonstrar a troca de mensagem entre as classes criadas, facilitando a implementação dos casos de uso. Nesta seção são apresentados os diagramas de sequência para os casos de uso `Visualizar pontos de interesse local (UC03)` e `Visualizar amigos (UC05)`.

3.3.4.1 Diagrama de sequência visualizar pontos de interesse local

O caso de uso `Visualizar pontos de interesse local (UC03)` inicia quando o usuário abre a tela de `Pontos de Interesse Local`, pois a aplicação cliente executará o método `onCreate()` para popular os dados ao criar a tela. Nesse instante será executado o método `getTodosPontosLocais()` da classe `PontoInteresseLocalREST`. Este, por sua vez, instancia a classe `WebserviceCliente` e executa o método `get()` passando uma `String`, que é formada pelo caminho do *webservice*, concatenando o método *webservice* desejado. A resposta desta chamada ao *webservice* trará uma *String* em formato JSON contendo a lista de pontos de interesse local que serão convertidos e retornados para o método que iniciou o processo. Caso ocorra algum erro no lado do servidor quando for realizada a chamada do

webservice, será lançado uma exceção no método `get()` da classe `WebserviceCliente` e este será tratado no método que chamou a classe. A Figura 15 apresenta o diagrama de sequência do caso de uso `Visualizar pontos de interesse local (UC03)`.

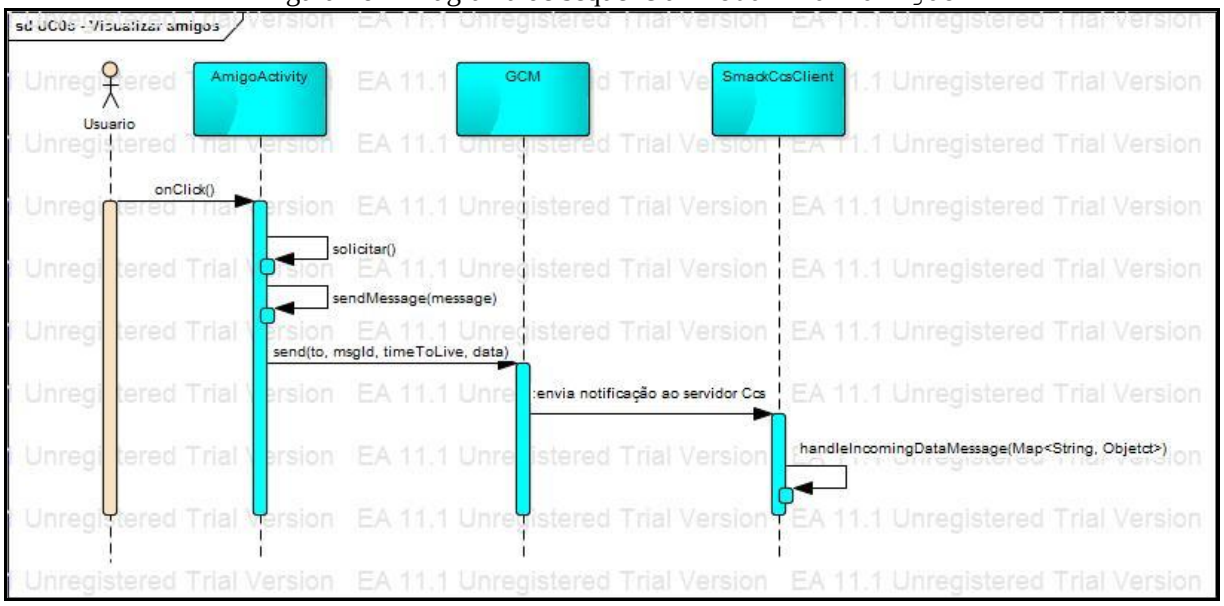
Figura 15 – Diagrama de sequência `Visualizar pontos de interesse local`



3.3.4.2 Diagrama de sequência Visualizar amigos

O caso de uso `Visualizar amigos (UC05)` começa quando o usuário acessa o cadastro de um amigo e seleciona o `checkbox "solicitar"` fazendo com que seja enviada uma notificação para o amigo perguntando se o mesmo permite que o usuário visualize a posição do amigo em questão. Ao clicar em solicitar a aplicação cliente executa o método `solicitar` da classe `AmigoActivity`. Neste momento é executado o método `sendMessage` para que este use a instancia do GCM para executar o método `send` para enviar a mensagem para o servidor do GCM. Este por sua vez, encaminhara a notificação para o serviço CCS. O serviço CCS tratará a notificação com o método `handleIncomingDataMessage` da classe `SmackCssClient`. A Figura 16 apresenta o diagrama de sequência `Visualizar Amigos`.

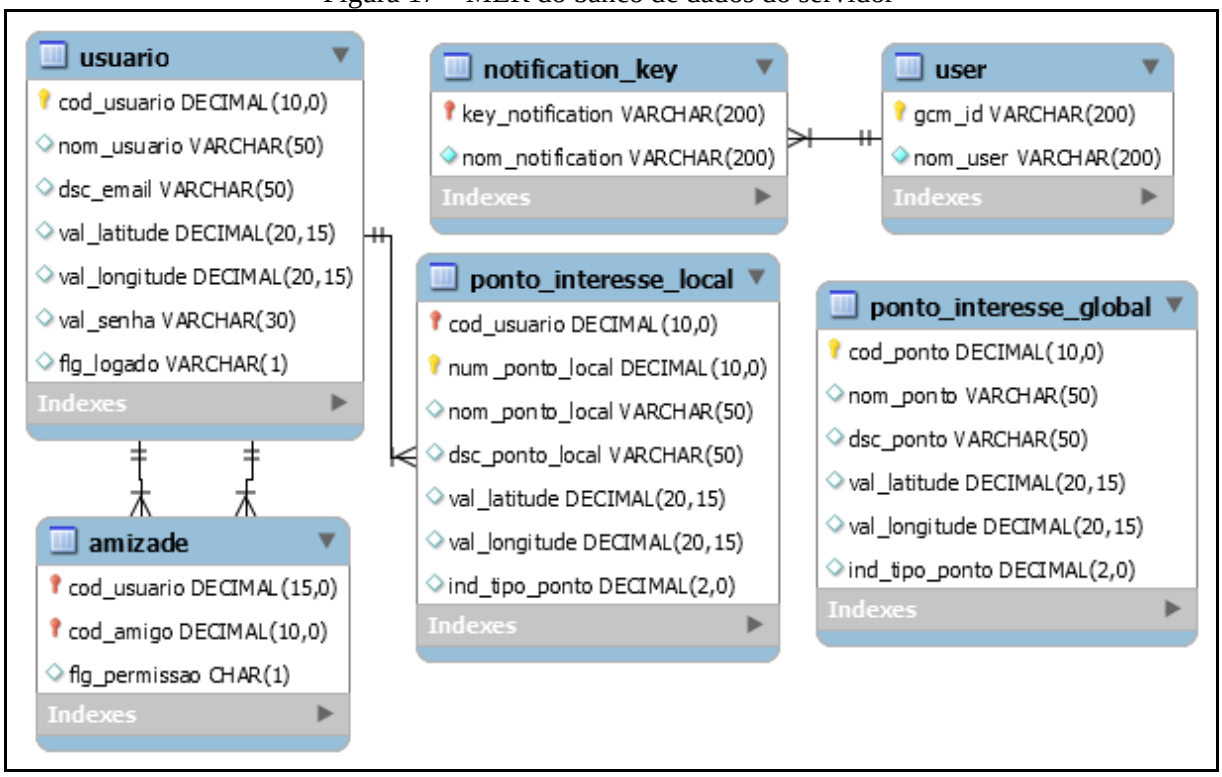
Figura 16 – Diagrama de sequência `Visualizar amigos`



3.3.5 Banco de Dados

As informações básicas do sistema foram persistidas em banco de dados apenas no servidor. O aplicativo cliente utiliza sempre o *webservice* para buscar ou enviar informações para serem persistidas na base de dados. O servidor utiliza uma base de dados MySQL para persistir e recuperar informações. A Figura 17 apresenta o MER do banco de dados do aplicativo servidor.

Figura 17 – MER do banco de dados do servidor



A tabela `Usuario` possui os dados de cadastro do mesmo. Ela possui também a posição atual do usuário nos campos `val_latitude` e `val_longitude`. Apenas uma classe acessa esta tabela, a classe `UsuarioDAO`. A tabela `Amizade` possui o registro da relação de amizade entre dois usuários. Ela foi criada para que fosse possível manter o registro permitindo o amigo visualizar a posição do usuário. A tabela é acessada através da classe `AmizadeDAO`. A tabela `Ponto_Interesse_Local` possui os registros dos pontos de interesse de cada usuário. Esta tabela foi criada para que cada usuário possa criar seus pontos de interesse e visualizá-los no mapa. Esta tabela é acessada pela classe `PontoInteresseLocalDAO`. A tabela `Ponto_Interesse_Global` possui os dados dos pontos de interesse cadastrados pela universidade que são visualizados por todos os usuários. A tabela é acessada apenas pela classe `PontoInteresseLocalDAO`, classe que possui todos os comando SQL relacionados a tabela. A tabela `User` armazena todos os dispositivos que se registraram no GCM. A tabela é acessada pela classe `UserDAO`. A tabela `Notification_Key` mantem os registros para que o serviço `SmackCcsClient` possa enviar uma notificação para um grupo de dispositivos. A tabela é acessada pela classe `NotificationKeyDAO`.

3.4 IMPLEMENTAÇÃO

A seguir serão apresentadas as técnicas e ferramentas utilizadas e a operacionalidade da aplicação.

3.4.1 Técnicas e ferramentas utilizadas

Na seção de técnicas e ferramentas será apresentada a implementação da camada DAO, para que se tenha um entendimento de como foi desenvolvido todas as classes DAO e entender como funciona este processo. Será apresentada a camada *webservice* do aplicativo servidor para identificar a forma de acesso aos métodos. As camadas de notificação do aplicativo cliente e servidor. Será apresentado também o Google Maps usado para mostrar a posição atual do usuário e também os pontos de interesse.

3.4.1.1 Camada DAO

O aplicativo servidor foi implementado de forma que centralize o acesso ao banco de dados e para acessá-lo foi implementado o padrão DAO. Este padrão é a camada responsável por realizar a conexão com o banco de dados, executar o comando SQL e retornar dados quando necessário. O Quadro 10 apresenta a estrutura da classe `ConnectionManager`, classe responsável por criar a conexão com o banco de dados.

Quadro 10 – Código da classe `ConnectionManager`

```
13  public class ConnectionManager {
14
15      private static ConnectionManager conexcaoSingleton;
16      private final String CONFIG_PATH = "C:\\Program Files\\Apache Software Foundation\\"
19      private String url;
20      private String user;
21      private String pass;
22      private Connection conn;
23
24      private ConnectionManager(){
57
58      public static ConnectionManager getInstance(){
59          if(conexcaoSingleton == null){
60              conexcaoSingleton = new ConnectionManager();
61          }
62          return conexcaoSingleton;
63      }
64
65      public Connection getConexao(){
66          return this.conn;
67      }
68
69      public void fechaConexao(Connection conn, PreparedStatement stm, ResultSet res){
```

Em seguida, é apresentado o código do método construtor da mesma classe, que é privado, pois foi optado por ser implementado o padrão de projeto `Singleton` para garantir que a aplicação use uma única instancia da classe `ConnectionManager` e garantir apenas uma conexão no banco. O Quadro 11 mostra o construtor da classe.

Quadro 11 – Método construtor da classe `ConnectionManager`

```
24    private ConnectionManager(){
25
26        try{
27            Class.forName("com.mysql.jdbc.Driver");
28        }catch(Exception e){
29            System.out.println("Classe nao encontrada!");
30            System.exit(1);
31        }
32
33        Properties props = new Properties();
34        FileInputStream in = null;
35        File f = null;
36
37        try {
38            f = new File(CONFIG_PATH);
39            in = new FileInputStream(f);
40            props.load(in);
41        } catch (IOException ex) {
42            System.out.println("Deu erro na recuperação das propriedades!");
43        }
44        try{
45            this.url = props.getProperty("database");
46            this.user = props.getProperty("db.user");
47            this.pass = props.getProperty("db.password");
48
49            conn = DriverManager.getConnection(this.url, this.user, this.pass);
50
51            System.out.println("Conexao realizada com Sucesso!");
52        }catch ( SQLException e){
53            System.out.println(e.getMessage());
54        }
55
56    }
```

As classes DAO utilizam a classe de conexão como variável de método nos métodos que fazem conexão com o banco de dados. A implementação das classes DAO parte do princípio que todas as classes terão comportamentos em comum e para isto foi criado a interface `IDAO`. Ela utiliza o tipo genérico do Java para permitir que as classes que a implementem possam especificar o tipo usado pela classe. As classes implementam os métodos `inserir(T)`, `atualizar(T)`, `remover(T)` e `retornaTodos()`. O Quadro 12 mostra a classe `UsuarioDAO` que representa a implementação de todas as outras classes DAO. O Quadro 13 traz o complemento da classe `UsuarioDAO` que é método `inserir`, responsável por inserir um usuário na tabela `Usuario`.

Quadro 12 – Classe `UsuarioDAO`

```
23 public class UsuarioDAO implements IDAO<Usuario> {
24
25     private static UsuarioDAO instance;
26
27     private UsuarioDAO() {
28     }
29
30     /**
31      * Metodo responsavel por controlar o acesso a instancia da classe UsuarioDAO
32      * @return the instance
33      */
34     public static UsuarioDAO getInstance() {
35         if(instance == null){
36             instance = new UsuarioDAO();
37         }
38         return instance;
39     }
40
```

Quadro 13 – Termino do método `inserir` da classe `UsuarioDAO`

```
41     @Override
42     public void inserir(Usuario t) throws DAOException, SQLException {
43         Connection conn = null;
44         PreparedStatement stm = null;
45
46         conn = ConnectionManager.getInstance().getConexao();
47         String sql = "INSERT INTO usuario "
48                 + " (cod_usuario,"
49                 + " nom_usuario,"
50                 + " dsc_email,"
51                 + " val_latitude,"
52                 + " val_longitude,"
53                 + " val_senha)"
54                 + " VALUES ( ?, ?, ?, ?, ?, ?)";
55
56         stm = conn.prepareStatement(sql);
57
58         t.setCodigo(this.getMaxCod());
59
60         stm.setInt(1, t.getCodigo());
61         stm.setString(2, t.getNome());
62         stm.setString(3, t.getEmail());
63         stm.setDouble(4, t.getLocalizacao().getLatitude());
64         stm.setDouble(5, t.getLocalizacao().getLongitude());
65         stm.setString(6, t.getSenha());
66
67         stm.execute();
68     }
```

Na linha 25 do Quadro 12, está a referencia da própria classe `UsuarioDAO`. Em seguida tem-se o construtor privado da classe e o método `getInstance()` que garante a implementação do padrão de projetos *Singleton*, para que outras classes acessem sempre a mesma instancia da classe DAO. Dando sequência, a partir da linha 42 inicia o método `inserir`. O método recebe um `Usuário` como parâmetro, recupera a conexão vigente na classe `ConnectionManager`. Na linha 47 é criado o SQL para inserir o usuário. No Quadro 13 é demostrado o carregamento do `PreparedStatement`, associação dos respectivos valores do

select com os dados do usuário e na linha 67 executado o *select* com o comando `stm.execute()`, para executar o comando *select* no banco de dados.

3.4.1.2 Camada Webservice

A camada *webservice* foi implementada para receber as requisições HTTP, acessar o banco de dados através da camada DAO e retornar as informações para quem fez a requisição. Um exemplo dessa camada é a classe `UsuarioResource` que utiliza de anotações para configurar os métodos acessíveis. O Quadro 14 apresenta a implementação usada na classe.

Quadro 14 – Implementação da classe `UsuarioResource`

```
26 @Singleton
27 @Path("usuario")
28 public class UsuarioResource {
29
30     public UsuarioResource(){
31     }
32
33     @POST
34     @Path("/inserir")
35     @Produces(MediaType.APPLICATION_JSON)
36     @Consumes(MediaType.APPLICATION_JSON)
37     public String insereUsuario(Usuario usuario){
38         boolean ret = new UsuarioController().inserirUsuario(usuario);
39         //System.out.println("Chamou innsereUsuario!");
40         if(ret){
41             return "Usuario inserido com sucesso!";
42         }else{
43             return "Ocorreu erro na inserção do usuário";
44         }
45     }
46
47     @GET
48     @Path("buscar/{id}")
49     @Produces(MediaType.APPLICATION_JSON)
50     public Usuario getUsuario(@PathParam("id") int id){
51         //System.out.println("Chamou getUsuario!");
52         Usuario usuario = new UsuarioController().retornarUsuario(id);
53         if(usuario == null){
54             throw new NoContentException("Usuario não encontrado!");
55         }
56         return usuario;
57     }
```

As linhas 26 e 27 do Quadro 14 mostram as duas anotações usadas em nível de classe. A anotação `@Singleton` faz com que exista apenas uma instancia da classe, fazendo com que o aplicativo servidor trate a instancia da classe com o padrão de projetos *Singleton*. A anotação `@Path("usuario")` especifica a URL que deve ser usada para acessar os recursos desta classe. Mais abaixo, nas linhas 33 a 36, tem-se as anotações em nível de método. A anotação `@POST` configura o método como o método `POST` do protocolo HTTP. A anotação `@Path`, agora usada em nível de método, serve para poder diferenciar o método que será usado

na URL chamada. As anotações `@Produces` e `@Consumes` são usadas para informar o tipo de retorno e o tipo de consumo de dados respectivamente. O método `inserirUsuario` recebe como parâmetro um objeto `Usuario` que será persistido na base de dados. Para realizar a persistência, usa-se uma instancia da classe `UsuarioController`, que por sua vez instancia um objeto `UsuarioDAO` e insere o objeto na base de dados. Em seguida, da linha 47 a 49 tem-se as anotações `@GET`, que configura o método como método `GET` do protocolo HTTP, `@Path("buscar/{id}")` que especifica a URL que deve ser usada para acessar o recurso da classe, incluindo o parâmetro `id` e a anotação `@Produces` que especifica o tipo de retorno do método. Na linha 50, onde é especificado o cabeçalho do método, encontra-se a anotação `@PathParm`, usada para especificar que o parâmetro `int id` está associado ao parâmetro `{id}` da anotação `@Path` do método. O método `getUsuario` executa o método `retornaUsuario` da classe `UsuarioController` para recuperar o usuário informado pelo parâmetro id. Conforme mostrado nas linhas 53 e 54, caso não exista o usuário informado, é lançada uma exceção `NoContentException` para que a resposta da chamada HTTP seja configurada como erro e tratada adequadamente no aplicativo cliente.

3.4.1.3 Notificações

Na seção responsável por explicar a implementação de notificações no ambiente Android foi separado em quatro itens, isto porque, para que o servidor envie notificações é necessária uma implementação no Android e para que o dispositivo móvel envie notificações é necessário uma implementação no servidor. Na primeira seção será tratado do envio e recebimento de notificações do servidor para o dispositivo móvel, também conhecido como servidor GCM. Neste caso será tratado envio e recebimento juntos, pois o servidor GCM deve receber uma notificação antes de enviar notificações. Na segunda seção será tratado o recebimento das notificações pelo dispositivo móvel. A terceira seção está reservada para mostrar o funcionamento do envio de notificações por parte do dispositivo móvel e por fim, a quarta seção trará a implementação que permite o servidor receber notificações,

3.4.1.3.1 Envio e recebimento de notificações do servidor

Para que o servidor envie e receba notificações é necessário implementar um servidor que se conecte com o GCM através de uma conexão XMPP e gerencie o envio de informações para os aplicativos, conforme informado pela Google (ANDROID DEVELOPERS, 2014c). De acordo com o aplicativo proposto, o servidor não inicia envio de notificações e sim, recebe notificação do dispositivo, trata e envia notificação de retorno ao

usuário ou a outro usuário, se necessário. Para enviar notificações ao GCM, o servidor deve se conectar a ele usando uma conexão XMPP, conforme a Quadro 15.

Quadro 15 – Conexão XMPP e o método `send`

```
175     XMPPConnection connection;
176
177     /**
178      * Sends a downstream GCM message.
179      */
180     public void send(String jsonRequest) {
181         Packet request = new GcmPacketExtension(jsonRequest).toPacket();
182         connection.sendPacket(request);
183     }
```

Conforme pode ser visto no Quadro 15, é necessária apenas a conexão com o GCM e uma *String* contendo a requisição que pode ser enviado informações aos dispositivos móveis. A configuração da conexão está representada na Quadro 16.

Quadro 16 – Configuração da conexão XMPP

```
332⊕     * Connects to GCM Cloud Connection Server using the supplied credentials.
340⊖    public void connect(String username, String password) throws XMPPException {
341         config = new ConnectionConfiguration(GCM_SERVER, GCM_PORT);
342         config.setSecurityMode(SecurityMode.enabled);
343         config.setReconnectionAllowed(true);
344         config.setRosterLoadedAtLogin(false);
345         config.setSendPresence(false);
346         config.setSocketFactory(SSLSocketFactory.getDefault());
347
348         // NOTE: Set to true to launch a window with information about packets
349         // sent and received
350         config.setDebuggerEnabled(true);
351
352         // -Dsmack.debugEnabled=true
353         XMPPConnection.DEBUG_ENABLED = true;
354
355         connection = new XMPPConnection(config);
356         connection.connect();
357
358⊖        connection.addConnectionListener(new ConnectionListener() {
359
360⊖            @Override
361             public void reconnectionSuccessful() {
362                 logger.info("Reconnecting..");
363             }
364
365⊖            @Override
366             public void reconnectionFailed(Exception e) {
367                 logger.log(Level.INFO, "Reconnection failed.. ", e);
368             }
369
370⊖            @Override
371             public void reconnectingIn(int seconds) {
372                 logger.log(Level.INFO, String.format("Reconnecting in %d secs", seconds));
373             }
```

O Quadro 17 apresenta a continuação do método `conect()` que faz a configuração da conexão e realiza a conexão com o GCM.

Quadro 17 – Continuação do método `conect()`

```
375            @Override
376            public void connectionClosedOnError(Exception e) {
377                logger.log(Level.INFO, "Connection closed on error.");
378            }
379
380            @Override
381            public void connectionClosed() {
382                logger.info("Connection closed.");
383            }
384        });
385
386        // Handle incoming packets
387        connection.addPacketListener(new PacketListener() {
388
389            @Override
390            public void processPacket(Packet packet) {
391                logger.log(Level.INFO, "Received: " + packet.toXML());
392                Message incomingMessage = (Message) packet;
393                GcmPacketExtension gcmPacket = (GcmPacketExtension) incomingMessage
394                        .getExtension(GCM_NAMESPACE);
395                String json = gcmPacket.getJson();
396                try {
397                    @SuppressWarnings("unchecked")
398                    Map<String, Object> jsonObject = (Map<String, Object>) JSONValue
399                            .parseWithException(json);
400
401                    // present for "ack"/"nack", null otherwise
402                    Object messageType = jsonObject.get("message_type");
403
404                    if (messageType == null) {
405                        // Normal upstream data message
406                        handleIncomingDataMessage(jsonObject);
407
408                        // Send ACK to CCS
409                        String messageId = jsonObject.get("message_id")
```

Na linha 387 do Quadro 17, apresenta-se o inicio do *listner* que processará todos os pacotes recebidos. Na linha 406 é executado o método `handleIncomingDataMessage`, responsável por tratar todos os tipos de mensagens enviadas pelo aplicativo móvel. O Quadro 18 mostra a continuação do método `conect()`.

Quadro 18 - Continuação do método `conect`

```
410                                 .toString();
411                         String from = jsonObject.get("from").toString();
412                         String ack = createJsonAck(from, messageId);
413                         send(ack);
414                     } else if ("ack".equals(messageType.toString())) {
415                         // Process Ack
416                         handleAckReceipt(jsonObject);
417                     } else if ("nack".equals(messageType.toString())) {
418                         // Process Nack
419                         handleNackReceipt(jsonObject);
420                     } else {
421                         logger.log(Level.WARNING,
422                                 "Unrecognized message type (%s)",
423                                 messageType.toString());
424                     }
425                 } catch (ParseException e) {
426                     logger.log(Level.SEVERE, "Error parsing JSON " + json, e);
427                 } catch (Exception e) {
428                     logger.log(Level.SEVERE, "Couldn't send echo.", e);
429                 }
430             }
431         }, new PacketTypeFilter(Message.class));
432 
433         // Log all outgoing packets
434         connection.addPacketInterceptor(new PacketInterceptor() {
435             @Override
436             public void interceptPacket(Packet packet) {
437                 logger.log(Level.INFO, "Sent: {0}", packet.toXML());
438             }
439         }, new PacketTypeFilter(Message.class));
440 
441         connection.login(username, password);
442     }
```

O Quadro 18 apresenta o fim do método `conect` e como são tratados os tipos de notificações enviados. Na linha 441 do Quadro 18 é realizado o *login* no GCM informando o `username` e `password` que são, respectivamente, o identificador do projeto cadastrado na Google, mais o *token* `@gcm.googleapis.com` e a *hash key* do projeto gerado pela Google. A aplicação agora recebe notificações, porém, é preciso entender as notificações recebidas e escrever da maneira especificada pela Google (ANDROID DEVELOPERS, 2014c).

Porém as informações tem uma estrutura e para criar esta estrutura, foi criado a classe `GcmPacketExtension`, responsável por formatar a notificação para que o GCM entenda e processe a mesma. O Quadro 19 apresenta o código que faz a formatação da mensagem a ser enviada. Em seguida o Quadro 20 mostra o fim da classe `GcmPacketExtension`.

Quadro 19 – Código da classe `GcmPacketExtension`

```
90    /**
91     * XMPP Packet Extension for GCM Cloud Connection Server.
92     */
93    class GcmPacketExtension extends DefaultPacketExtension {
94        String json;
95
96        public GcmPacketExtension(String json) {
97            super(GCM_ELEMENT_NAME, GCM_NAMESPACE);
98            this.json = json;
99        }
100
101       public String getJson() {
102           return json;
103       }
104
105       @Override
106       public String toXML() {
107           return String.format("<%s xmlns=\"%s\">%s</%s>", GCM_ELEMENT_NAME,
108                   GCM_NAMESPACE, json, GCM_ELEMENT_NAME);
109       }
110
111       public Packet toPacket() {
112           return new Message() {
113               // Must override toXML() because it includes a <body>
114               @Override
115               public String toXML() {
116                   StringBuilder buf = new StringBuilder();
117                   buf.append("<message");
118                   if (getXmlns() != null) {
119                       buf.append(" xmlns=\"").append(getXmlns()).append("\"");
120                   }
121                   if (getLanguage() != null) {
122                       buf.append(" xml:lang=\"").append(getLanguage())
123                               .append("\"");
124                   }
```

Quadro 20 – Fim da classe GcmPacketExtension

```
125                   if (getPacketID() != null) {
126                       buf.append(" id=\"").append(getPacketID()).append("\"");
127                   }
128                   if (getTo() != null) {
129                       buf.append(" to=\"")
130                               .append(StringUtils.escapeForXML(getTo()))
131                               .append("\"");
132                   }
133                   if (getFrom() != null) {
134                       buf.append(" from=\"")
135                               .append(StringUtils.escapeForXML(getFrom()))
136                               .append("\"");
137                   }
138                   buf.append(">");
139                   buf.append(GcmPacketExtension.this.toXML());
140                   buf.append("</message>");
141                   return buf.toString();
142               }
143           };
144       }
145   }
```

Agora para que seja enviada qualquer mensagem para o GCM basta criar uma mensagem adicionar como atributo na classe `GcmPacketExtension`, chamar o método

`toXml()` e usar o método `sendPacket` da conexão XMPP, conforme feito no Quadro 15. Um exemplo deste envio é quando a classe `SmackCcsClient` recebe uma notificação e envia uma mensagem `Ack` como resposta ao remetente da mensagem. O Quadro 21 mostra o código utilizado para realizar tal operação.

Quadro 21 – Envio de mensagem `Ack` para quem enviou notificação

```
404                     if (messageType == null) {
405                         // Normal upstream data message
406                         handleIncomingDataMessage(jsonObject);
407
408                         // Send ACK to CCS
409                         String messageId = jsonObject.get("message_id")
410                                 .toString();
411                         String from = jsonObject.get("from").toString();
412                         String ack = createJsonAck(from, messageId);
413                         send(ack);
414                     } else if ("ack".equals(messageType.toString())) {
415                         // Process Ack
416                         handleAckReceipt(jsonObject);
417                     } else if ("nack".equals(messageType.toString())) {
418                         // Process Nack
419                         handleNackReceipt(jsonObject);
420                     } else {
421                         logger.log(Level.WARNING,
422                                 "Unrecognized message type (%s)",
423                                 messageType.toString());
424                     }
```

3.4.1.3.2 Recebimento de notificações no dispositivo móvel

Os dispositivos móveis que usam Android recebem notificações de maneira implícita, porém, para que sua aplicação receba estas informações é necessário especificar no arquivo `Manifest.xml` da aplicação, uma entidade responsável por tratar as notificações. O Quadro 22 apresenta o trecho do arquivo manifesto que contém a configuração da entidade.

Quadro 22 – Configuração da entidade que trata notificações na aplicação cliente

```
77      <receiver
78          android:name="br.com.viseduandroid.notification.broadcastreceiver.GcmBroadcastReceiver"
79          android:permission="com.google.android.c2dm.permission.SEND" >
80          <intent-filter>
81              <action android:name="com.google.android.c2dm.intent.RECEIVE" />
82              <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
83
84              <category android:name="br.com.viseduandroid" />
85          </intent-filter>
86      </receiver>
```

A propriedade `android:name` especifica a classe responsável por tratar as notificações. Esta classe precisa estender os comportamentos da classe `BroadcastReceiver` para receber a informação como objeto `Intent` e poder processar da maneira necessária. Para isto foi criado as classes do pacote `br.com.visedu.notification`. São as classes responsáveis por configurar o recebimento de notificações, implementar o `BroadcastReceiver` e o serviço `IntentService` que agem quando a aplicação recebe uma

notificação. A classe `Globals` armazena os valores utilizados pelo GCM, como `GCM_SENDER_ID`, que é o identificador usado pelo GCM para identificar quem está enviando a mensagem e enviar para o servidor correspondente que também possui este identificador. O Quadro 23 mostra a classe `Globals`.

Quadro 23 – Classe `Globals`

```
1  package br.com.viseduandroid.notification;
2
3  /**
4   * Classe responsavel por manter as variaveis globais do GCM CCS
5   * @author Bruno André Kestring - kestringbak@gmail.com
6   * @since 17/09/2014
7   * @version 1.0
8   */
9  public class Globals {
10
11     public static final String TAG = "GCM DEMO";
12
13     public static final String GCM_SENDER_ID = "914807375209";
14
15     public static final String PREFS_NAME = "VISEDUANDROID";
16     public static final String PREFS_PROPERTY_REG_ID = "registration_id";
17
18     public static final long GCM_TIME_TO_LIVE = 60L * 60L * 24L * 7L * 4L; // 4 Weeks
19
20 }
```

A classe `GcmBroadcastReceiver` estende de `WakefulBroadcastReceiver` para que seja reservado um *lock* para a aplicação poder tratar a notificação. Este *lock* é usado para impedir que o processador do Android pare de processar mesmo quando o usuário venha a bloquear o dispositivo. Nesta classe também é definido o serviço que processará a notificação. O Quadro 24 apresenta a implementação da classe `GcmBroadcastReceiver`.

Quadro 24 – Classe `GcmBroadcastReceiver`

```
1  /**
4  package br.com.viseduandroid.notification.broadcastreceiver;
5
6  import br.com.viseduandroid.notification.intentservice.GcmIntentService;
12
13 /**
14  * Classe responsavel por implementar o comportamento de criar wake lock
15  * para manter o aparelho "acordado" enquanto recebe a notificacao
16  *
17  * @author Bruno André Kestring - kestringbak@gmail.com
18  * @since 17/09/2014
19  * @version 1.0
20  */
21 public class GcmBroadcastReceiver extends WakefulBroadcastReceiver {
22
23     @Override
24     public void onReceive(Context context, Intent intent) {
25         // Explicitly specify that GcmIntentService will handle the intent.
26         ComponentName comp = new ComponentName(context.getPackageName(),
27                 GcmIntentService.class.getName());
28         // Start the service, keeping the device awake while it is launching.
29         startWakefulService(context, (intent.setComponent(comp)));
30         setResultCode(Activity.RESULT_OK);
31     }
32 }
```

Conforme pode ser visualizado no Quadro 24, ao receber uma notificação o método `onReceive` recebe uma `Intent` como parâmetro e usa esta para passar para o *service* através da instancia da classe `ComponentName`. Após isto, a classe efetua a requisição do *lock* através do método `startWakefulService`. Por sua vez, classe `GcmIntentService` é a classe que trata a notificação recebida pelo `GcmBroadcastReceiver`. A classe estende o comportamento da classe `IntentService`, necessário para implementar o comportamento do método `onHandleIntent` que recebe a notificação e no final libera o *lock* do processador adquirido pela classe `GcmBroadcastReceiver`. O Quadro 25 apresenta a implementação do método `onHandleIntent`.

Quadro 25 – Método `onHandleIntent` da classe `GcmIntentService`

```
40    @Override
41    protected void onHandleIntent(Intent intent) {
42        Bundle extras = intent.getExtras();
43        GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
44        // The getMessageType() intent parameter must be the intent you received
45        // in your BroadcastReceiver.
46        String messageType = gcm.getMessageType(intent);
47
48        if (!extras.isEmpty()) // has effect of unparcelling Bundle
49        {
50            /*
51             * Filter messages based on message type. Since it is likely that
52             * GCM will be extended in the future with new message types, just
53             * ignore any message types you're not interested in, or that you
54             * don't recognize.
55             */
56            if (GoogleCloudMessaging.MESSAGE_TYPE_SEND_ERROR
57                    .equals(messageType)) {
58                sendNotification("Send error: " + extras.toString(), null);
59            } else if (GoogleCloudMessaging.MESSAGE_TYPE_DELETED
60                    .equals(messageType)) {
61                sendNotification(
62                        "Deleted messages on server: " + extras.toString(),
63                        null);
64            } else if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE
65                    .equals(messageType)) {
66                // Post notification of received message.
67                String message = ((intent.getExtras() == null) ? "Empty Bundle"
68                        : intent.getExtras().getString("message"));
69                if (intent.getExtras() != null
70                        && "br.com.visedu.gcm.CLEAR_NOTIFICATION"
71                                .equals(intent.getExtras().getString("action"))) {
72                    clearNotification();
73                } else {
74                    sendNotification("Received: " + message, extras);
```

Neste método é instanciado um objeto da classe `GoogleCloudMessage` para efetuar a verificação do tipo da mensagem. Caso seja uma notificação conhecida pela aplicação e não seja do tipo `br.com.viseduandroid.CLEAR_NOTIFICATION` é executado o método `sentNotification` que criará uma notificação para o usuário visualizar. No final do método é executado o método `GcmBroadcastReceiver.completeWakefulIntent(Intent)` para que o *lock* adquirido pela aplicação para manter o processador do dispositivo executando seja liberado.

3.4.1.3.3 Envio de notificações do dispositivo móvel

O envio de notificações por parte do dispositivo móvel é realizado através de uma instância da classe `GoogleCloudMessage` disponibilizado pela biblioteca `google_play_service`. Na implementação deste trabalho, ficou decidido que a notificação seria enviada no momento em que o usuário solicita a permissão de visualizar a posição do amigo. A classe responsável por enviar a notificação ao servidor CCS é a `AmigoActivity`. O Quadro 26 mostra a implementação do método `sendMessage`.

Quadro 26 – Método `sendMessage` da classe `AmigoActivity`

```
147     /**
148      * Upstream a GCM message up to the 3rd party server
149      *
150      * @param message
151      */
152     private void sendMessage(String message) {
153         if (regid == null || regid.equals("")) {
154             Toast.makeText(this, "You must register first", Toast.LENGTH_LONG)
155                     .show();
156             return;
157         }
158         new AsyncTask<String, Void, String>() {
159             @Override
160             protected String doInBackground(String... params) {
161                 String msg = "";
162                 try {
163                     Bundle data = new Bundle();
164                     data.putString("message", params[0]);
165                     data.putString("action", "br.com.visedu.gcm.NOTIFICATION");
166                     String id = Integer.toString(msgId.incrementAndGet());
167                     gcm.send(Globals.GCM_SENDER_ID + "@gcm.googleapis.com", id,
168                             Globals.GCM_TIME_TO_LIVE, data);
169                     msg = "Solicitação enviada!";
170                 } catch (IOException ex) {
171                     msg = "Error :" + ex.getMessage();
172                 }
173                 return msg;
174             }
175
176             @Override
177             protected void onPostExecute(String msg) {
178                 Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
179             }
180         }.execute(message);
181     }
```

Como pode ser visualizado no Quadro 26, na linha 158 instancia-se um objeto da classe `AsyncTask` para que o envio da notificação seja realizado sem interromper o funcionamento da aplicação. Foi instanciado um objeto `Bundle` na linha 163 para armazenar os dados da notificação. Gerado um identificador de mensagem é possível realizar o envio da notificação na linha 167 com o método `gcm.send()`. Os parâmetros usados no método servem para identificar o destino. A constante `Globals.GCM_SENDER_ID` possui o identificador do

cadastro do projeto na Google e o *token* adicional serve para o GCM identificar o servidor destino, que neste caso é o nosso servidor CCS. O parâmetro `id` é o identificador da mensagem. O parâmetro `Globals.GCM_TIME_TO_LIVE` serve para informar ao GCM o tempo de vida da notificação, caso ele não consiga encontrar o destino e o parâmetro `data` possui as informações da notificação.

3.4.1.4 Mapa da Aplicação

A tela principal do sistema possui um mapa em que aparece a posição atual do usuário, os seus pontos de interesse e seus amigos. Para fazer isto, foi utilizado a nova versão do Google Maps, o Google Maps Android API v2. Para utilizar essa versão foi necessário o *download* da biblioteca `google_play_service`. Para adicionar o mapa na tela configura-se o XML da tela conforme a Quadro 27.

Quadro 27 – Configuração do XML `activity_mapa` da tela do mapa

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3    android:layout_width="match_parent"
4    android:layout_height="match_parent"
5    android:orientation="vertical">
6
7      <Spinner
8          android:id="@+id/spinnerMapa"
9          android:layout_width="fill_parent"
10         android:layout_height="44dp"
11         android:prompt="@string/mapa_destino" />
12
13     <fragment
14       android:id="@+id/map"
15       android:layout_width="match_parent"
16       android:layout_height="match_parent"
17       class="com.google.android.gms.maps.SupportMapFragment"/>
18
19 </LinearLayout>
```

Como pode ser visto na Quadro 27, a *tag* `<fragmente` serve para adicionar o mapa na tela e o atributo `class="com.google.android.gms.maps.SupportMapFragment"` faz com que o mapa seja suportado por dispositivos móveis com Android versão de número 8 ou superior. A classe `MapaActivity` é responsável por carregar o layout criado no XML do Quadro 27 no método `onCrate` da `Activity`. Executam-se os métodos responsáveis por manter as posições dos pontos de interesse do usuário e também dos seus amigos. No Quadro 28 pode ser visualizado os métodos responsáveis por controlar os marcadores que aparecerão no mapa.

Quadro 28 – Métodos responsáveis por manter os pontos de interesse atualizados

```
173    @Override
174    protected void onResume() {
175        super.onResume();
176        setUpMapIfNeeded();
177        setUpLocationClientIfNeeded();
178        locationClient.connect();
179    }
180
181    private void setUpMapIfNeeded() {
196
197    private void setUpMap() {
198        map.setMyLocationEnabled(true);
199        map.setOnMyLocationButtonClickListener(myLocationListner);
200        map.setOnMarkerClickListener(markerClickListner);
201        map.setOnMapLongClickListener(mapLongClickListner);
202        atualizarPontos();
203    }
204
205    private void atualizarPontos(){
206        new AsyncTask<Void, Void, Void>() {
207
208            @Override
209            protected Void doInBackground(Void... params) {
210                drawAmigos();
211                drawWayPointsGlobais();
212                drawWayPointsLocais();
213                return null;
214            }
215        }.execute();
216    }
```

O método `setUpMapIfNeeded` verifica a necessidade de instanciar um objeto da classe `SupportMapFragment` e então executa o método `setUpMap` que é usado para configurar os *listners* usados na classe. No mesmo método `setUpMap`, é executado também o método `atualizarPontos`, responsável por manter atualizado os pontos do usuário. Ele executa uma tarefa assíncrona para garantir que os processos serão executados, mas não travarão a aplicação até terminarem o processo. Para demonstrar como os pontos de interesse são desenhados no mapa, apresenta-se o Quadro 29.

Quadro 29 – Método que desenha os amigos no mapa

```
281    private void drawAmigos() {
282        carregaListaAmigos();
283        if (listaAmigos != null && !listaAmigos.isEmpty()) {
284            for (Amizade a : listaAmigos) {
285                Marker marker = listaMarkersAmigos
286                        .get(a.getAmigo().getCodigo());
287                if (marker != null) {
288                    LatLng latlng = new LatLng(a.getAmigo().getLocalizacao()
289                            .getLatitude(), a.getAmigo().getLocalizacao()
290                            .getLongitude());
291                    marker.setPosition(latlng);
292                } else {
293                    marker = map.addMarker(new MarkerOptions()
294                            .position(
295                                    new LatLng(a.getAmigo().getLocalizacao()
296                                            .getLatitude(), a.getAmigo()
297                                            .getLocalizacao().getLongitude()))
298                            .title(a.getAmigo().getNome())
299                            .snippet(a.getAmigo().getEmail())
300                            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN))
301                            );
302                    listaMarkersAmigos.put(a.getAmigo().getCodigo(), marker);
303                }
304            }
305        }
306    }
```

Conforme pode ser visualizado no Quadro 29, executa-se o método `carregarListaAmigos()` na 282 para recuperar a lista de amigos do *webservice*. Está lista é percorrida na linha 284 para que seja verificado se cada registro já possui um marcador na tela. Caso já exista um marcador atualiza-se apenas a posição deste marcador. Caso contrario, usa-se a instancia `map` da classe `GoogleMap` para adicionar um marcador no mapa, conforme apresentado na linha 293 a 301, adicionando esse marcador na lista de marcadores.

A atualização da posição do usuário no mapa acontece também utilizando a classe GoogleMap. O Quadro 30 apresenta o trecho de código que mostra o método que trata a mudança de posição do usuário.

Quadro 30 – Método que trata a mudança de posição do usuário

```
243    @Override
244    public void onLocationChanged(Location location) {
245        // é chamado a cada 5 segundos de acordo com a configuracao do
246        // LocationRequest
247        // aqui será capturada a posicao atual do usuario
248        Localizacao localizacao = new
249                Localizacao(location.getLatitude(),
250                location.getLongitude());
251        usuario.setLocalizacao(localizacao);
252        LatLng latLng = new LatLng(location.getLatitude(),
253                location.getLongitude());
254        if (markerUsuario == null) {
255            markerUsuario = map.addMarker(new MarkerOptions()
256            .position(latLng)
257            .title(usuario.getNome())
258            .snippet("Estou aqui!")
259            .icon(BitmapDescriptorFactory.fromResource(R.drawable.livros)));
260        } else {
261            markerUsuario.setPosition(latLng);
262        }
263        atualizarPosicaoAtual();
264    }
```

Para que o método do Quadro 30 seja utilizado a classe `MapaActivity` precisa implementar o comportamento da interface `LocationListner`. Foi utilizado também uma instancia da classe `LocationRequest` para configurar o intervalo entre cada leitura, a frequência da leitura e também a qualidade da requisição. Isto foi feito para que o método `onLocationChanged` executasse a cada 5 segundos e recuperasse a posição da melhor forma possível. Com a localização recuperada, atualiza-se a posição atual do usuário e, em seguida, o método `atualizarPosicaoAtual` é executado para enviar a posição do usuário para a base de dados.

3.4.2 Operacionalidade da implementação

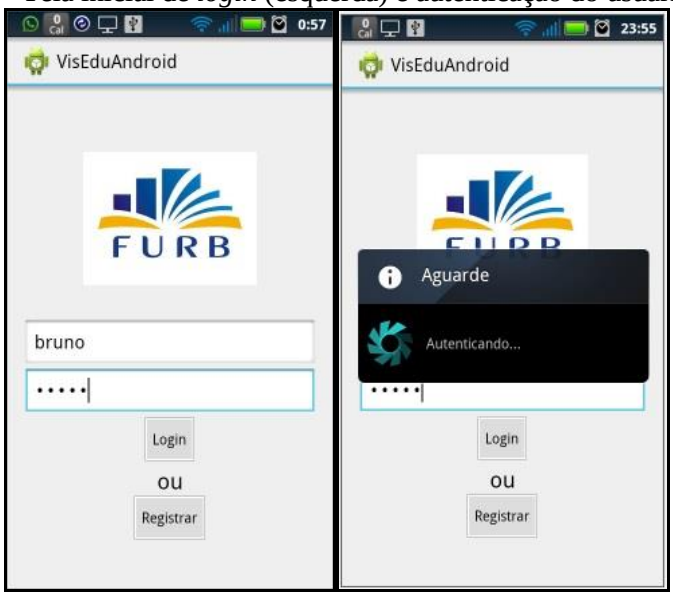
Nesta seção serão apresentadas as funcionalidades das telas utilizadas pelo usuário. Esta seção apresenta a tela principal do aplicativo e seus componentes, a visualização das informações de amizades, pontos de interesse global e local, e o mapa com as informações. Para as demonstrações a seguir, foi utilizado o dispositivo Motorola Defy + com o Android 2.3.5 instalado.

3.4.2.1 Tela inicial do aplicativo

A `tela inicial` do aplicativo é apresentada para que o usuário possa realizar o *login* no aplicativo. O botão `Login` serve justamente para realizar o *login* no aplicativo para usuários já cadastrados. Conforme já previsto, usuários que não tem vínculo nenhum com a

FURB podem realizar o cadastro normalmente através do botão `Registrar`. A Figura 18 mostra a `tela inicial` do aplicativo.

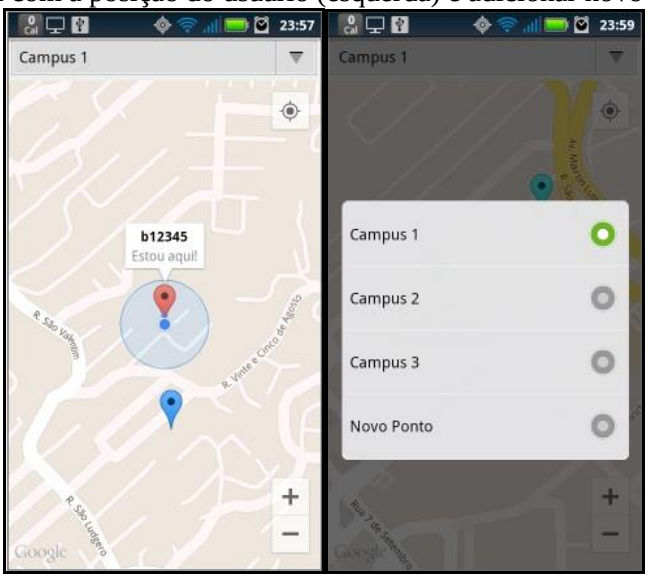
Figura 18 – Tela inicial de *login* (esquerda) e autenticação do usuário (direita)



3.4.2.2 Tela principal do aplicativo

A `tela Mapa` é a tela principal do aplicativo, pois nela é possível ver a posição atual do usuário, dos seus amigos, dos seus pontos de interesse locais e dos pontos de interesse globais. A Figura 19 mostra a `tela Mapa` com a posição atual do usuário (ponto vermelho) e um ponto de interesse local (ponto azul escuro). Através desta tela é possível mover a câmera até as dependências dos três campis da FURB e adicionar novos pontos de interesse locais, como pode ser visto na Figura 19.

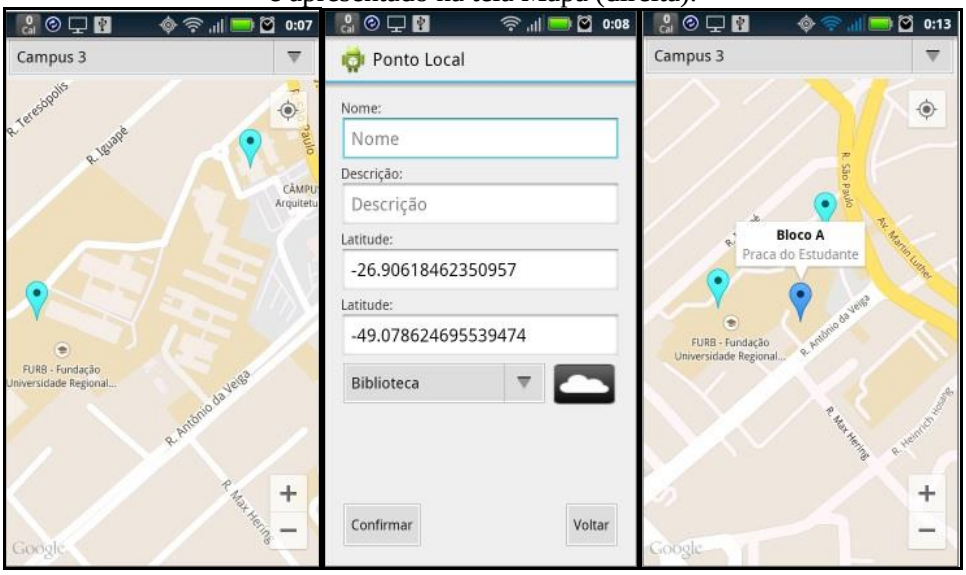
Figura 19 – Tela Mapa com a posição do usuário (esquerda) e adicionar novo ponto local (esquerda)



3.4.2.3 Adicionando novo ponto local a partir da tela Mapa

A adição de um novo ponto local pode acontecer de duas formas. Primeiro, o usuário realiza um longo click no mapa o aplicativo mostrará a tela de cadastro de ponto com a posição selecionada. Já a segunda forma é selecionar o `Spinner` na `tela Mapa` e selecionar a opção `Novo Ponto`. Esta opção também trará a tela de cadastro de ponto, mas a posição mostrada será a posição atual do usuário. A Figura 20 mostra os passos realizados para adicionar um ponto local.

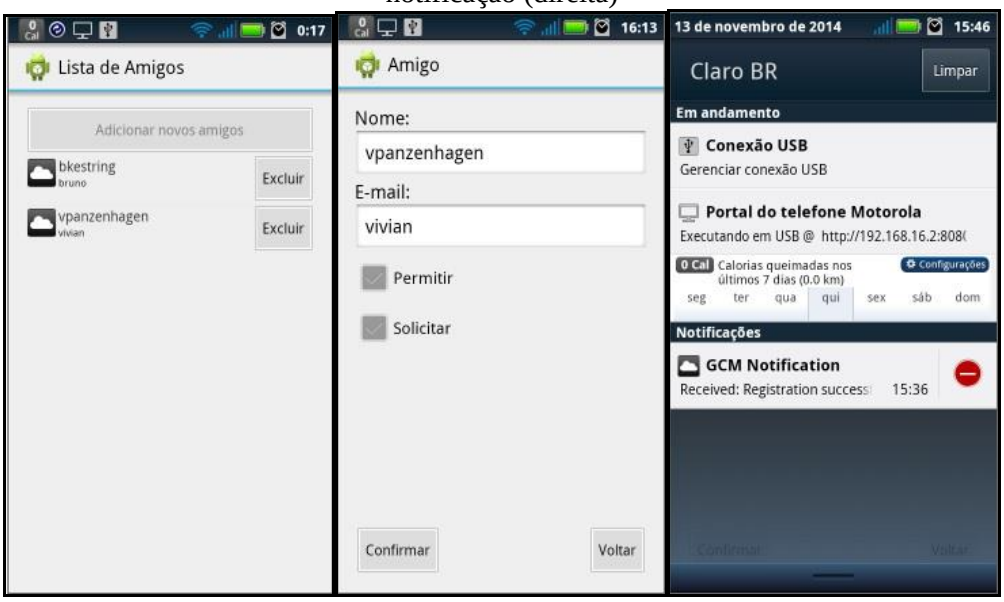
Figura 20 – Na `tela Mapa` (esquerda), tela de cadastro de Ponto Local (centro) e finalizando o ponto é apresentado na tela Mapa (direita).



3.4.2.4 Lista de Amigos

A `tela de Lista de Amigos` mostra os amigos que o usuário possui. Somente amigos cadastrados poderão ser visualizados no Mapa, porém eles devem permitir que o usuário visualize-os. Através desta tela é possível excluir e adicionar novos amigos. Ao acessar o registro de um amigo, é realizado o registro no servidor GCM CCS para que o dispositivo móvel envie notificações ao servidor de notificações. A Figura 21 mostra a `tela Lista de Amigos`, o registro de um amigo e o recebimento de uma notificação no dispositivo móvel.

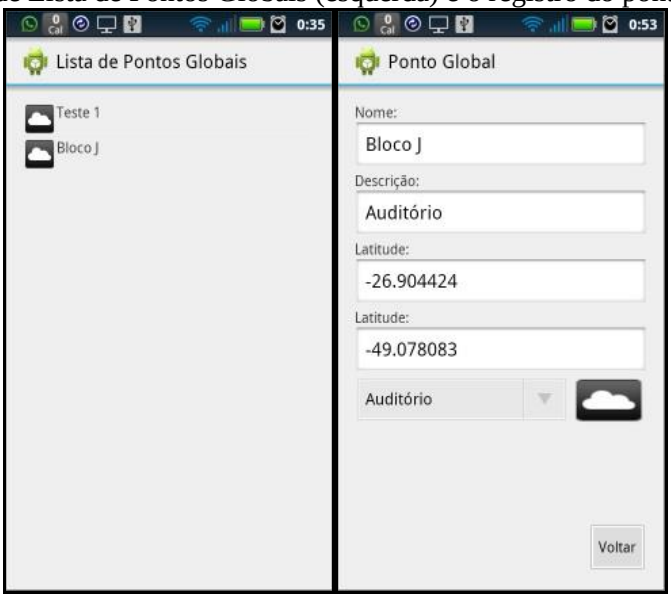
Figura 21 – Tela Lista de amigos (esquerda), o registro do amigo (centro) e o recebimento de notificação (direita)



3.4.2.5 Lista de Pontos Globais

A `tela Lista de Pontos Globais` apresenta os pontos cadastrados pela FURB. Eles podem ser acessados por todos os usuários e não podem ser modificados pelos mesmos. A Figura 22 mostra a tela de Lista de Pontos Globais da aplicação e o registro de um ponto global.

Figura 22 – Tela de Lista de Pontos Globais (esquerda) e o registro do ponto global (direita).



3.5 RESULTADOS E DISCUSSÃO

O presente trabalho teve como objetivo inicial desenvolver um aplicativo para a plataforma Android, que permitisse que usuários participantes do evento Interação FURB pudessem se localizar na universidade, encontrar oficinas, salas, eventos agendados pela organização do evento e encontrar outros participantes. Para encontrar participantes, o usuário precisa adicionar este participante como amigo e então solicitar a permissão de visualizar a posição atual do mesmo. Para isto, implementou-se uma classe no aplicativo servidor capaz de enviar notificações para os dispositivos móveis chamado Google Cloud Message, porém, percebe-se que a ação partia dos usuários e que seria necessário que os dispositivos móveis enviassem notificações, coisa que o servidor GCM não permite. Para suprir essa necessidade, alterou-se a implementação do aplicativo servidor de GCM para Cloud Connection Server (CCS), implementação que permite o servidor GCM receber e enviar notificações.

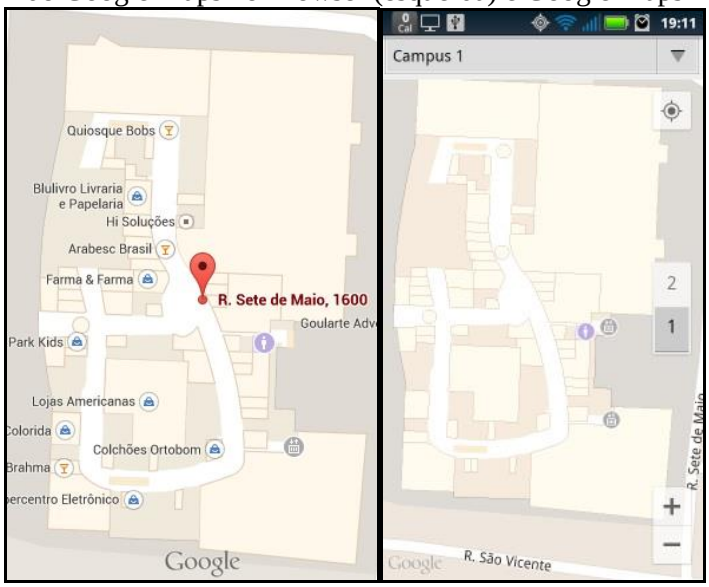
Inicialmente a implementação do trabalho foi realizado com a IDE Android Studio. De acordo com Google Developers (2014d, tradução nossa), esta IDE é um novo ambiente de desenvolvimento Android com base no IntelliJ IDEA. Ele fornece novos recursos e melhorias sobre o Eclipse ADT e será a IDE oficial Android quando ele estiver pronto. Porém, como a

IDE ainda está em fase beta, existem algumas inconsistências e não existe muita documentação, mas já existe uma comunidade utilizando essa IDE.

Inicialmente havia sido cogitado a possibilidade de realizar a autenticação dos usuários pela API do Facebook, para que apenas usuários cadastrados na rede social pudessem utilizar o aplicativo e o controle de amizades fosse feito pela rede social, mas ficou decidido deixar essa funcionalidade para extensões.

Durante a fase de fundamentação teórica identificou-se que o Google permite o cadastro de mapas Indoors, que são as plantas de lugares como shoppings. Nestes locais podem ser cadastrados pontos de interesse como lojas, banheiros, escadas, etc. Porém, ao iniciar a implementação do aplicativo cliente, foi identificado que pontos visualizados nos mapas nos browser de computadores não apareciam no Google Maps do Android. A Figura 23 mostra a diferença entre os dois mapas. Eles estão posicionados no mesmo local, neste caso escolheu-se o Shopping Park Europeu da cidade de Blumenau.

Figura 23 – Imagem do Google Maps no Browser (esquerda) e Google Maps no aplicativo cliente



Fonte: Google Maps (2014).

Por este motivo, notou-se a necessidade de implementar os Pontos de Interesse Global, para que a universidade cadastre pontos que são de interesse a todos os usuários, como salas, laboratórios, auditórios, entre outros.

Foram realizados testes de desempenho para identificar a quantidade de tempo levada para aplicação executar alguns algoritmos com quantidade de registros diferenciados.

3.5.1 Teste de Desempenho

Os testes de desempenho foram realizados na tela Mapa, pois esta é a tela principal da aplicação, é onde o aplicativo buscará informações de amigos, pontos de interesse global,

pontos de interesse local e também a posição atual do próprio usuário. Mais especificamente, os testes foram realizados nas rotinas que fazem a atualização da posição atual do usuário e da atualização dos outros pontos. Foi escolhida esta parte para testar o desempenho, porque quanto mais registros, mais a aplicação demora em apresentar os registros no mapa e em casos extremos a aplicação trava. A Tabela 1 mostra os resultados dos testes realizados.

Tabela 1 – Testes de desempenho

Nº de registros por usuário	Tempo de resposta em segundos	Taxa de sucesso em porcentagem
20	0,364 s	100%
100	0,670 s	100%
200	2,303 s	70%
300	5,500 s	15%
400	12,000 s	0%

Para obter os resultados visualizados na tabela acima, realizou-se várias coletas com os mesmos números de registro para excluir os resultados que ficavam fora da média e então realizar a média entre os valores. Como pode ser observado, o tempo de resposta aumenta consideravelmente na medida em que o número de registros é acrescentado. Outro ponto relevante é a taxa de sucesso nos resultados na medida em que o numero de registros aumenta. Foi realizado esta averiguação por notar que a aplicação estava travando devido a demora excessiva nas consultas com muitos registros.

3.6 COMPARATIVO DOS TRABALHOS CORRELATOS

Esta seção é responsável por realizar a comparação entre as funcionalidades dos trabalhos descritos na seção anterior. O Quadro 31 apresenta a relação da existência das características nos trabalhos apresentados. O presente trabalho está identificado como VISEDU, o trabalho descrito na seção 2.4.1 foi apresentado como KUEHL (2012), o aplicativo descrito na seção 2.4.2 foi apresentado como HARVARD (2014) e o trabalho descrito na seção 2.4.3 foi apresentado como ANGELI (2010). O trabalho que contiver contribuição para o item em questão será marcado com “Sim” e o que não tiver será marcado com “Não”.

Quadro 31 – Quadro comparativo entre os trabalhos correlatos

	KUEHL (2012)	HARVARD (2014)	ANGELI (2010)	VISEDU
Mundo Virtual	**Não**	**Não**	**Sim**	**Não**
Geolocalização do usuário	**Sim**	**Sim**	**Não**	**Sim**
Interface Gráfica com Mapa	**Não**	**Sim**	**Não**	**Sim**
Cálculo de proximidade	**Sim**	**Sim**	**Não**	**Não**
Notificação	**Não**	**Não**	**Não**	**Sim**
Persistência de dados	**Sim**	**Não**	**Sim**	**Sim**

Como pode ser visto no Quadro 31, as características dos trabalhos correlatos são distintas, onde que trabalhos que possuem informações em mapas, não possuem mundos virtuais para interação. Os trabalhos de Kuehl (2012) é interessante, pois utiliza da geolocalização do usuário para calcular a distância entre outros usuários para determinar se eles estão dentro do raio especificado pelo usuário. Porém ele não possui interação com mapas e nem um mundo virtual. O aplicativo disponibilizado por Harvard é interessantes, pois utilizam da geolocalização do usuário e cálculos matemáticos para definir a proximidade entre pontos de ônibus e o usuário, e também apresenta interação com mapas mostrando a posição atual do usuário e a posição dos ônibus selecionados. Porém, não apresenta persistência de dados e não possui mundo virtual. O trabalho de Angeli (2010) é interessante, pois apresenta a utilização de *frameworks* que possibilitam a criação de mundos virtuais para a interação entre usuários e possui persistência de dados. Porém, não utiliza da geolocalização do usuário, não realiza cálculos com a geolocalização do usuário e também não apresenta interação com mapas. O presente trabalho possui interação com mapas, utiliza da geolocalização do usuário e apresenta persistência de dados. Porém, não realiza cálculos com a geolocalização do usuário e não possui mundo virtual.

4 CONCLUSÕES

O presente trabalho apresenta o desenvolvimento de um aplicativo para a plataforma Android, que utiliza de geolocalização, mapas, notificações e interface gráfica para a utilização de novas ferramentas disponibilizadas pela Google, como o Google Maps Android API v2 e o envio de notificações GCM Cloud Connection Server. Para suprir as necessidades deste aplicativo, foi realizado o desenvolvimento de um aplicativo servidor que atua como *webservice*, capaz de receber requisições `POST` e `GET` do protocolo HTTP, processar as informações, acessar o banco de dados MySQL e retornar as informações ao aplicativo cliente via HTTP.

O cenário utilizado como base para o estudo realizado foi a necessidade de auxiliar os visitantes da FURB a encontrar locais desejados e por isto foi escolhido o evento Interação FURB, pois os participantes possuem pouco ou nenhum conhecimento de onde ficam os lugares na FURB, como por exemplo, o complexo Esportivo do Campus I. Para isto, o aplicativo permite o cadastro de qualquer tipo de usuário, sem a necessidade de vínculo estudantil com a universidade.

Quanto à implementação, inicialmente a falta de conhecimento e experiência com o ambiente de desenvolvimento Android aumentou o desafio na realização do trabalho. Outro ponto que somou ao desafio foi à busca de informações quanto às novas API's disponibilizadas pela Google para o Android, quanto ao Google Maps e o serviço de Notificação. Especificamente, o serviço de Notificações não possui muita documentação e poucos exemplos e por este motivo foi um desafio maior conseguir implementar sua funcionalidade.

Foi encontrado dificuldade na implementação do mapa no Android, porque em testes prévios, identificou-se que ao cadastrar uma planta de uma construção no Google Indoor, é possível cadastrar pontos de interesse no mapa e estes estarão visíveis a todos que visualizarem a estrutura cadastrada, porém, para a versão disponibilizada para desenvolvimento no Android, estes pontos de interesse não eram apresentados. Para solucionar este problema foi implementado os Pontos de Interesse Global onde que a aplicação fica responsável por desenhar estes pontos no mapa.

Por fim, a integração entre as várias tecnologias empregadas se mostrou de grande valia devido às possibilidades que as ferramentas trazem. Apesar de algumas limitações na implementação principalmente devido a dificuldade de encontrar materiais de estudo, foi

possível implementar as funções de notificação que permitem os usuários trocarem informações sem a necessidade do servidor *webservice*.

4.1 EXTENSÕES

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

a) realizar a autenticação de usuários através de uma rede social, como o Facebook, para que o aplicativo fique mais sociável;
b) implementar um algoritmo que distingue a altitude atual do usuários com a altitude dos pontos apresentados no mapa, para que o usuários consiga distinguir se o ponto está acima ou abaixo do mesmo (diferentes andares);
c) disponibilizar formas de realizar gincanas entre os participantes do Interação FURB;
d) criar uma base de dados local no aplicativo para reduzir o número de chamadas ao *webservice*;
e) realizar um sincronismo dos dados contidos na base de dados local do dispositivo com a base de dados no *webservice*;
f) utilizar um ambiente virtual com as dimensões da FURB para que os usuários possam visualizar informações ao deslocar-se pela universidade;
g) criar tipos de Amizades que expiram com o tempo. Exemplo: por tempo ou uma região em particular;
h) criar uma forma de diferenciar usuários que estão logados e deslogados, para que o usuário consiga distinguir se seu amigo está logado na aplicação;
i) melhorar a rotina de atualização de pontos no mapa;
j) incluir uma opção nas telas de pontos de interesse (Global e Local) que permita visualizar o ponto no mapa.

REFERÊNCIAS

ALBUQUERQUE, Antonia L. VELHO, Luiz. **Mundos virtuais e jogos por computador:** Pong – Um Estudo de Caso. Instituto de Matemática Pura e Aplicada. [S.l.], 2001. Disponível em: <http://www.visgraf.impa.br/Data/RefBib/PS_PDF/pong2001/reportech.pdf>. Acesso em: 04 abr. 2014.

ANDROID DEVELOPERS . **SENSOR**. [S.l.], 2014a. Disponível em: <http://developer.android.com/reference/android/hardware/Sensor.html>. Acesso em: 07 abr. 2014.

_____. **Google cloud message for Android**. [S.l.], 2014b. Disponível em: <http://developer.android.com/google/gcm/index.html>. Acesso em: 11 nov. 2014.

_____. **GCM cloud connection server (XMPP)**. [S.l.], 2014c. Disponível em: <http://developer.android.com/google/gcm/ccs.html>. Acesso em 13 nov. 2014.

_____. **Android Studio**. [S.l.], 2014d. Disponível em: <https://developer.android.com/sdk/installing/studio.html>. Acesso em 13 nov. 2014.

ANGELI, Flaviano L. **Protótipo de mundo virtual para relacionamento com participantes do Interação FURB**. 2010. 65. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DEVMEDIA. **Android**: Mapas e geolocalização – Revista mobile magazine 37. [S.l.], 2014. Disponível em: <http://www.devmedia.com.br/android-mapas-e-geolocalizacao-revista-mobile-magazine-37/22041>. Acesso em: 03 abr. 2014.

GEOREFERENCE. **Sistema de coordenadas UTM**. [S.l.], 2014. Disponível em: <http://georeference.blogspot.com.br/2010/02/sistema-de-coordenadas-utm.html>. Acesso em: 03 abr. 2014.

Google. **Google Maps**.[S.l], 2014. Disponível em: <https://www.google.com.br/maps/>. Acesso em: 14 nov. 2014.

GRAZIADIO E-LEARNING. **History of geolocation**. [Los Angeles], 2011. Disponível em: <https://wikis.pepperdine.edu/display/GSBME/History+of+Geolocation>. Acesso em: 03 abr. 2014.

HARVARD. **Mobile web application**. [S.l.], 2014. Disponível em: <http://m.harvard.edu/info/#>. Acesso em: 26 mar. 2014.

KUEHL, Cesar A. **Protótipo de sistema móvel na plataforma Android para compartilhamento de arquivos e mensagens entre dispositivos baseado em proximidade geográfica**. 2012. 115 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SÓ GEOGRAFIA. **Cartografia**. [S.l.], 2014. Disponível em: <http://www.sogeografia.com.br/Conteudos/GeografiaFisica/Cartografia/>. Acesso em: 03 abr. 2014.

TECMUNDO. **História**: a evolução do celular. [S.l.], 2014. Disponível em: <http://www.tecmundo.com.br/celular/2140-historia-a-evolucao-do-celular.htm> Acesso em 04 abr. 2014.

UNIVERSIDADE REGIONAL DE BLUMENAU. **Nossa história:** como tudo começou. Blumenau, [2014a?]. Disponível em: <http://www.furb.br/web/1317/institucional/a-furb/nossa-historia>. Acesso em: 03 abr. 2014.

_____. **Interação FURB**. Blumenau, [2014b?]. Disponível em: <http://www.furb.br/web/2818/interacao-furb-2014>. Acesso em: 21 maio 2014.