UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM INFORMÁTICA

EVERTON VIEIRA MENDES

UM APLICATIVO PARA ANDROID VISANDO PROPORCIONAR MAIOR INTERAÇÃO DE UMA BANDA MUSICAL E SEUS SEGUIDORES

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2011

EVERTON VIEIRA MENDES

UM APLICATIVO PARA ANDROID VISANDO PROPORCIONAR MAIOR INTERAÇÃO DE UMA BANDA MUSICAL E SEUS SEGUIDORES

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Informática, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção de título de Tecnólogo.

Orientador: Prof. Tarlis Portela, Esp.

PATO BRANCO

2011

ATA Nº: **178**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **EVERTON VIEIRA MENDES.**

Às 13:55 hrs do dia 4 de julho de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Tarlis Tortelli Portela (Orientador), Beatriz Terezinha Borsoi (Convidada) e Marcelo Zanetti (Convidado), para avaliar o Trabalho de Diplomação do aluno Everton Vieira Mendes, matrícula 671770, sob o título **Um Aplicativo para Android Visando Proporcionar Maior Interação entre uma Banda Musical e seus Seguidores**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Coordenação de Informática. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:40 hrs foi encerrada a sessão.

Prof. Tarlis Tortelli Portela, Esp.
Orientador

Profa. Beatriz Terezinha Borsoi, Dr.
Convidada

Prof. Marcelo Zanetti, M.Sc.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

DEDICATÓRIA

Dedico o presente trabalho à minha mãe que foi quem sempre me apoiou para que o mesmo fosse realizado.

RESUMO

MENDES, Everton Vieira. Um Aplicativo para Android Visando Proporcionar Maior Interação de uma Banda Musical e seus Seguidores**.** 2011. 117 f. Monografia de Trabalho de Conclusão de Curso. Curso Superior de Tecnologia em Informática. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2011.

Atualmente o cenário musical está repleto de novas bandas independentes, as quais se esforçam para divulgar seus trabalhos. Com a grande popularização de dispositivos móveis com acesso a Internet, criou-se um ótimo meio de divulgação para essas bandas, possibilitando assim expandir o canal de comunicação com seus respectivos fãs. A plataforma Android atualmente é uma das mais utilizadas pelas empresas fabricantes de dispositivos móveis, tornando o desenvolvimento de um aplicativo para esta plataforma uma opção adequada para a expansão dos canais de comunicação da banda para com os seus fãs. Através o uso de várias ferramentas e tecnologias, demonstra-se as diversas etapas a serem superadas no desenvolvimento do projeto. Objetiva-se então a criação de um aplicativo para a plataforma Android com funcionalidades que envolvem agenda de *shows,* integração com a rede social Facebook e o uso de mapas. Desta maneira, obtém-se um aplicativo capaz de informar o usuário, e que possibilita o mesmo repassar essa informação.

Palavras-chave: Android; *Smartphone;* Java; Banda Musical; Facebook.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATIURAS E SIGLAS

ADT	*Android Development Tools*
API	*Application Programming Interface*
AVD	*Android Virtual Device*
IDE	*Integrated Development Environment*
JDK	*Java Development Kit*
JVM	*Java Virtual Machine*
OHA	*Open Handset Alliance*
OMG	*Object Management Group*
OMT	*Object Modeling Technique*
OOP	*Oriented Object Programming*
OS	*Operating System*
OSI	*Open Source Initiative*
PC	*Personal Computer*
RUP	*Rational Unified Process*
SO	Sistema Operacional
SSO	*Single Sign-on*
UML	*Unified Modeling Language*
UP	*Unified Process*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Com a crescente popularização dos dispositivos móveis e o constante aumento de suas respectivas funcionalidades, os chamados *Smartphones*[1] vem se tornando um objeto de desejo, e em muitos casos, instrumento necessário à rotina diária das pessoas.

Um *smartphone,* em sua essência, é um telefone celular, porém com funcionalidades estendidas por meio de programas executados em seu SO – Sistema Operacional ou OS do inglês: *Operating System*. Segundo Pereira e Silva (2009) o celular é atualmente o produto de consumo mais utilizado no mundo.

Atualmente um dos sistemas operacionais que são utilizados nesses aparelhos é o Android™, desenvolvido pela Google™. ”O Android™ é uma plataforma para tecnologia móvel completa, envolvendo um pacote com programas para celulares, já com um sistema operacional, *middleware*, aplicativos e interface do usuário.” (PEREIRA e SILVA, 2009, p. 03).

O sistema Android já é utilizado por muitos fabricantes de *smartphones,* contabilizando um crescente aumento no número de programas desenvolvidos especialmente para esta plataforma.

Desta forma, pretende-se mostrar o processo de desenvolvimento de um programa para o Android™, exemplificando as tecnologias e as ferramentas utilizadas nesse processo.

O desenvolvimento do programa em questão envolveu a utilização de conexão com a Internet juntamente com o uso de mapas e integração com uma rede social, constituindo-se em um *software* que proporciona funcionalidades de divulgação da agenda de shows de uma banda musical.

---

[1] Tradução livre do inglês: “Telefone inteligente”.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um aplicativo que estimule a interação de fãs com uma banda musical, através da divulgação de agenda de shows, e integração com mapas e redes sociais.

1.1.2 Objetivos Específicos

Para atender ao objetivo deste trabalho, pretende-se:

- Realizar levantamento bibliográfico acerca das teorias necessárias ao desenvolvimento do aplicativo proposto, envolvendo basicamente, conceitos de programação na plataforma Android através do uso da linguagem Java;
- Exemplificar o uso das ferramentas e das tecnologias empregadas no processo de desenvolvimento de um programa para a plataforma Android, com foco no uso de mapas e sistema de login do tipo SSO (*Single Sign-on*) com a rede social Facebook.
- Exemplificar o uso de mapas em aplicações para Android com a API (*Application Programming Interface* ou Interface de Programação de Aplicações) que a Google disponibiliza para desenvolvedores de programas para esta plataforma. Esta API permite o acesso a recursos do serviço de mapas chamado *Google Maps;*
- Exemplificar o uso do sistema de *login* SSO por meio da API do Facebook[2];

1.2 JUSTIFICATIVA

Atualmente o cenário musical está repleto de novas bandas independentes, que lutam diariamente para conseguir seu espaço na cena e atingir o *mainstream*[3]. Tais bandas precisam andar lado a lado com a tecnologia para proporcionar um

[2] Site de relacionamento – Rede Social.

[3] Popular ao público geral – "fluxo principal" (CARDOSO FILHO & JANOTTI JÚNIOR, 2006).

meio de comunicação eficaz e barato para a divulgação de seus trabalhos. Com a crescente popularização do acesso a Internet nos dispositivos móveis, os *smartphones* tornaram-se um ótimo meio de propagação de informações atualizadas que podem ser acessadas de praticamente qualquer lugar no mundo, constituindo um novo meio de comunicação de uma banda musical com os seus fãs.

Segundo Hashimi, Komatineni e MacLean (2010), os dispositivos móveis estão destinados a se tornarem o próximo PC (*Personal Computer* ou Computador Pessoal).

A plataforma Android apresenta-se cada vez mais como uma solução completa no mercado dos dispositivos móveis, tornando-se a opção de uso dos principais fabricantes de aparelhos celulares inteligentes.

A elaboração do presente trabalho justifica-se também por proporcionar conhecimento em tecnologias atuais, como o desenvolvimento de um software para uma plataforma móvel, e integração do mesmo ao uso de mapas e compartilhamento em redes sociais.

1.3 ESTRUTURA DO TRABALHO

Apresenta-se a seguir como será a estrutura do presente trabalho:

- Capítulo 1 - Apresenta uma breve introdução ao trabalho, contendo os objetivos e as justificativas do estudo;
- Capítulo 2 - Apresenta o referencial teórico, com os conceitos utilizados para o entendimento do estudo desenvolvido;
- Capítulo 3 - Descreve os materiais e métodos utilizados no desenvolvimento do sistema;
- Capítulo 4 – Encontram-se os resultados e discussões do trabalho, contendo a descrição, a modelagem e a implementação do sistema, além dos dados obtidos e consequentes discussões.

2 REFERENCIAL TEÓRICO

Serão abordados neste capítulo, conceitos de programação para a plataforma Android, por meio do uso da linguagem Java juntamente com o Android SDK – *Software Development Kit*. Por outro lado, também serão apresentados alguns aspectos do meio musical, enfatizando a necessidade de um aplicativo que possibilite uma melhor interação de uma banda com seus fãs.

2.1 BANDAS INDEPENDENTES NO MEIO MUSICAL

Atualmente existe um incalculável número de bandas no cenário musical independente. Tal cenário distingue-se principalmente por não criar vínculos com nenhuma gravadora, caracterizando-se como independente. Desta forma, as bandas que populam esse meio, utilizam-se principalmente da Internet para a divulgação de seus trabalhos, onde criam perfis em redes sociais, que se tornam um meio de comunicação constante com seus fãs.

2.2 ANDROID

Comprado em 2005 pela empresa Google, o Android foi anunciado oficialmente apenas em 5 de novembro de 2007. Na mesma ocasião a Google também anunciou a "*Open Handset Alliance* um grupo de empresas que ajudaria a desenvolvê-lo. Entre os membros da *Open Handset Alliance* estão operadoras de telefonia, desenvolvedores de softwares, fabricantes de dispositivos e fabricantes de componentes." (KARCH, 2010, pg. 01).

Conforme Pereira e Silva (2009), o Android é o primeiro projeto de uma plataforma *open source*[4] para dispositivos móveis em conjunto com a *Open Handset Alliance* (OHA). Segundo o mesmo autor, a plataforma Android™ foi desenvolvida com base no sistema operacional Linux e é composta por um conjunto de ferramentas que atua em todas as fases do desenvolvimento do projeto, desde a execução até a criação de softwares específicos. No entanto, o mesmo destaca que apesar de ser baseado em Linux não pode ser considerado um Linux.

[4] Significa "código aberto" – Criado pela OSI (*Open Source Initiative*).

A licença de uso do Android prega que a plataforma pode ser utilizada gratuitamente pelos fabricantes de dispositivos, podendo também ser customizada. Qualquer desenvolvedor pode criar programas para o Android e disponibilizá-los no *Android Market*[5] sem a necessidade da obtenção de uma permissão.

Os aplicativos para Android são desenvolvidos em Java, e executam na máquina virtual Dalvik, que é uma “máquina virtual otimizada para dispositivos móveis” (PEREIRA e SILVA, 2009, p. 9).

2.2.1 Estrutura da Plataforma Android

A seguir, a Figura 1 apresenta as camadas da plataforma Android.

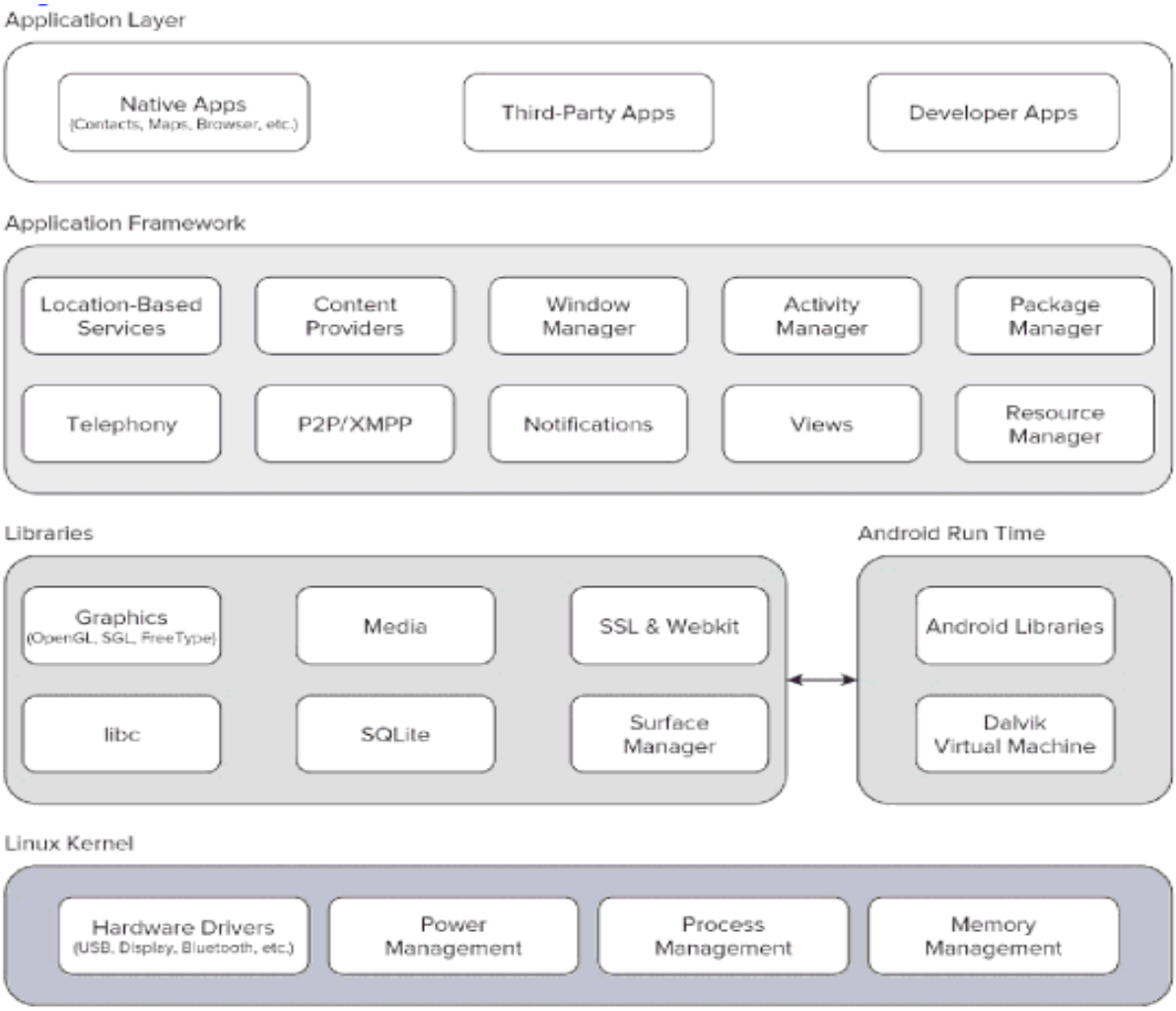

Figura 1 - Camadas do sistema Android

Fonte: Meyer (2010).

[5] Mercado online de aplicativos da plataforma Android.

Como ilustrado na Figura 1, pode-se destacar os seguintes aspectos:

- **Linux *Kernel***: Cuida do gerenciamento de drivers de hardware, processo, memória, segurança, rede e energia;
- ***Libraries***: Nesta camada situam-se bibliotecas de mídias, armazenamento, segurança, gerenciamento do *display*, entre outros;
- **Android *Run Time*:** Grupo de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem Java; Toda a aplicação Android roda em seu próprio processo, com sua própria instancia na máquina virtual Dalvik. Segundo Meyer (2010), a Android *Run Time* é a camada que realmente diferencia a plataforma Android de uma implementação Linux;
- ***Application Framework*:** Camada onde ficam as classes usadas para criar as aplicações no sistema operacional Android;
- ***Application Layer*:** Constam os aplicativos, nativos ou não, desenvolvidos através da linguagem Java. Conforme Meyer (2010), esses aplicativos rodam dentro da Android *Run Time*, e fazem uso das classes e serviços disponíveis na camada Android *Framework*.

2.3 ANÁLISE E PROJETO DE SISTEMAS ORIENTADOS A OBJETOS

Segundo Tonsig (2000, pg. 19), “a análise de sistemas consiste nos métodos e técnicas de investigação e especificação da solução de problemas, a partir dos requisitos levantados, para criação e implementação de software em algum meio que o suporte”.

De maneira mais específica, Silva (2007) afirma que o objetivo da análise é a modelagem do domínio do problema, em contraste com a modelagem computacional. A análise orientada a objetos consiste em produzir uma modelagem do domínio do problema, descrevendo-o como um conjunto de classes.

O autor ainda descreve que o objetivo da etapa de projeto é complementar ao da etapa de análise, porém o foco é o domínio da solução computacional e as classes definidas na etapa de análise sofrerão alterações para adequarem-se a um modelo computacional implementável.

2.4 CONCEITOS BÁSICOS DE ORIENTAÇÃO A OBJETOS

Programação orientada a objetos (ou *Oriented Object Programming* - OOP) é uma técnica de programação na qual o problema a ser abordado é modelado como sendo constituído por um conjunto de objetos que interagem entre si. Os conceitos básicos de orientação a objeto são (CAMPIONE e WALRATH, 1996):

- **Classe**: Uma classe é definida como um molde ou gabarito por meio do qual serão definidos os objetos. Assim, classe é um protótipo que define os métodos e atributos comuns a um conjunto de objetos de um mesmo tipo;
- **Objeto**: Um objeto é uma instância de uma classe. Ele representa uma entidade, conceito ou abstração individual pertinente ao domínio do problema sob análise;
- **Atributos**: Um atributo é uma propriedade do objeto. Eles representam a informação contida, na forma de variáveis ou constantes, dentro do objeto definido, registrando o estado atual do mesmo;
- **Métodos**: Representam o conjunto de operações que um objeto pode realizar. Basicamente, são sub-rotinas que manipulam variáveis locais, atributos próprios ou de outros objetos e parâmetros por passagem de valor;
- **Herança**: O mecanismo de herança permite que características comuns a um determinado conjunto de classes possam ser derivadas a partir de uma classe base, reaproveitando declarações e estruturas existentes.

Outros conceitos importantes segundo Silva e Videira (2001, p. 89) são:

- **Encapsulamento:** é o processo de "esconder" todos os detalhes de um objeto que não contribuem para as suas características essenciais nem para a disponibilização de funcionalidades para o seu exterior;
- **Abstração:** é a representação concisa de uma ideia ou objeto mais complexa, incidindo sobre as características essenciais do objeto;
- **Polimorfismo:** representa a capacidade de objetos diferentes responderem de forma diferente à mesma mensagem. Isso se da pelo fato de uma operação aceitar argumentos de tipos diferentes ou mesmo desconhecidos.

A Figura 2 apresenta a representação gráfica de uma classe, conforme a linguagem de modelagem unificada (UML - *Unified Modeling Language*), que divide a classe em três agrupamentos, sendo que o nome da classe é apresentado primeiramente. Em seguida, são listados os atributos (campos pertencentes aos objetos desta classe), explicitando o seu nome, tipo e, quando possível, seu valor inicial. Finalmente, são apresentados os métodos da classe, descrevendo o seu nome, lista de argumentos e tipo de valor retornado. Esses agrupamentos devem ser separados por uma linha horizontal.

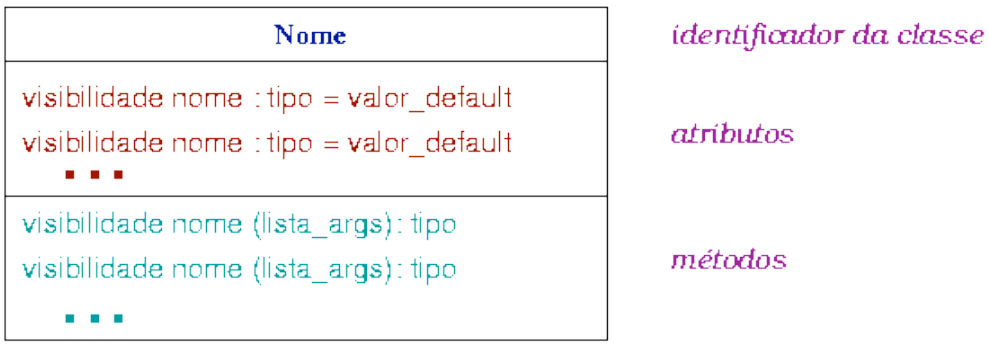

Figura 2 - Exemplo de uma classe
Fonte: UNICAMP WEBSITE.

Durante o desenvolvimento de projetos orientados a objetos, podem ser obtidos alguns modelos, dentre os quais o mais popular é o diagrama de classes, resultante da aplicação da técnica OMT - *Object Modeling Technique* (RUMBAUGH, 1991). Estes diagramas descrevem as funcionalidades e atributos inerentes ao problema para as diferentes classes, e são apresentados com maior detalhe na seção 4.4 que trata da UML.

2.5 PROCESSO UNIFICADO DE DESENVOLVIMENTO DE SOFTWARE

O UP (*Unified Process*, traduzido por Processo Unificado) é um processo estabelecido para o desenvolvimento de software. Jacobson et al. (1999) apresentam as origens do UP no processo *Objectory* (uma das metodologias pioneiras em relação à orientação a objeto), que teve sua primeira versão apresentada em 1987, passando pelas contribuições do processo *Rational Objectory* (1997), até chegar ao Processo Unificado da Rational – RUP (KRUCHTEN, 2000).

Jacobson et al. (1999) explica que o processo unificado de desenvolvimento de software é o conjunto de atividades necessárias para transformar requisitos do usuário em um sistema de software.

Baseando-se no autor Raul Sidnei Wazlawick, considera-se que a fase de concepção deve ser a primeira fase do processo de desenvolvimento de software, na qual se procura levantar os principais requisitos e compreender o sistema de forma abrangente (WAZLAWICK, 2004).

2.5.1 Fase de Concepção

De acordo com Wazlawick (2004, p. 32), “a fase de concepção consiste em uma etapa na qual o analista vai buscar as primeiras informações sobre o sistema a ser desenvolvido. Nessa etapa assume-se pouco conhecimento do analista sobre o sistema e uma grande interação com o usuário e cliente”.

As atividades da fase de concepção são: Levantamento de requisitos; Organização dos requisitos e Planejamento do desenvolvimento. As duas primeiras atividades serão abordadas neste trabalho.

A etapa de levantamento de requisitos compreende em buscar todas as informações possíveis sobre as funções que o sistema deve executar e as restrições que devem existir para um bom funcionamento do sistema. Nesta etapa são produzidos alguns artefatos como:

- A visão geral do sistema que é um texto corrido e deve descrever as principais ideias do sistema;
- Os requisitos funcionais e não-funcionais que registram os tópicos sobre o que o sistema deve fazer e sob que condições o sistema deve fazer as coisas.

2.5.2 Levantamento de Requisitos

Tanto Wazlawick (2004) como Larman (2004) comentam que requisitos são os objetivos ou as capacidades que o sistema deve atender.

Segundo Cysneiros e Leite (1997), os requisitos não funcionais, ao contrário dos funcionais, não expressam nenhuma função a ser realizada pelo software, e sim comportamentos e restrições que este software deve satisfazer.

Um exemplo da diferença entre os dois pode ser visto a seguir:

Requisito funcional:

- Calcular saldo a pagar de imposto de renda.

Requisito não-funcional:

- A declaração deve ser simples o suficiente para que o cálculo do imposto seja feito sem a necessidade do usuário pedir ajuda a um contador.

Os Quadros 1 e 2 mostram exemplos de requisitos funcionais e não-funcionais.

Quadro 1 - Exemplo de requisitos funcionais para controle acadêmico

Identificação	Descrição	Importância
[R1] Cadastrar Alunos	O sistema deve permitir que sejam cadastrados alunos, mediante inclusão de seus dados.	Essencial
[R2] Listar Alunos	O sistema deve permitir a listagem dos alunos cadastrados	Essencial
[R3] Inclusão de notas	O sistema deve permitir a inclusão (cadastro) de notas a partir da seleção de um item listado no requisito R2	Essencial

Quadro 2 - Exemplo de requisitos não-funcionais para controle acadêmico

Identificação	Descrição	Importância
[RN1] Confiabilidade	O sistema deve possuir mecanismos que garantam que o usuário não perca informações.	Essencial
[RN2] Listar Alunos	O sistema deve oferecer uma ferramenta de backup.	Essencial
[RN3] Segurança	Somente usuários autorizados devem ter acesso ao banco de dados.	Essencial

2.6 A LINGUAGEM DE MODELAGEM UNIFICADA

A UML (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) surgiu da união de métodos anteriores para análise e projeto de sistemas orientados a objetos, e em 1997 passou a ser aceita e reconhecida como um padrão potencial de notação para modelagem de múltiplas perspectivas de sistemas de informações pela OMG (*Object Management Group*) (BOOCH *et al.*, 2000).

Conforme Shalloway e Trott (2004), a UML é uma linguagem visual utilizada para criar modelos de programas, os quais podem ser entendidos como representação diagramática dos programas. Nela podem ser vistos os relacionamentos entre objetos no código.

A linguagem UML na versão 2.0 é composta por treze diagramas, classificados em duas categorias: os diagramas estruturais, que tratam o aspecto estrutural tanto do ponto de vista do sistema quanto das classes e os diagramas de comportamento que são voltados a descrever o sistema computacional modelado

quando em execução. Os diagramas de comportamento têm uma subcategoria, que são os diagramas de interação (SILVA, 2007, p. 84).

Os diagramas, conforme foram definidos em Silva (2007) estão descritos a seguir:

- Diagramas Estruturais: Diagrama de Classe, Diagrama de Pacotes, Diagrama de Componentes, Diagrama de Objetos, Diagrama de Estrutura Composta e Diagrama de Utilização.
- Diagramas de Comportamento: Diagrama de Casos de Uso, Diagrama de Maquina de Estados e Diagrama de Atividades.
- Diagramas de Interação: Diagrama de Sequência, Diagrama de Visão Geral de Interação, Diagrama de Comunicação e Diagrama de Temporização.

Neste trabalho serão definidos apenas os diagramas de caso de uso e atividades, por esta razão somente esses serão explicados a seguir.

2.6.1 Diagrama de Casos de Uso

"A visão do sistema sob modelagem dada pelo diagrama de casos de uso é o que se chama de *visão caixa-preta*, isto é, observa-se o que o sistema faz, mas não sua estrutura interna" (SILVA, 2007. p. 101).

O diagrama de casos de uso, exemplificado na Figura 3, é voltado à modelagem do conjunto de funcionalidades de um software. Tem por objetivo relacionar as funcionalidades do sistema sem detalhá-las.

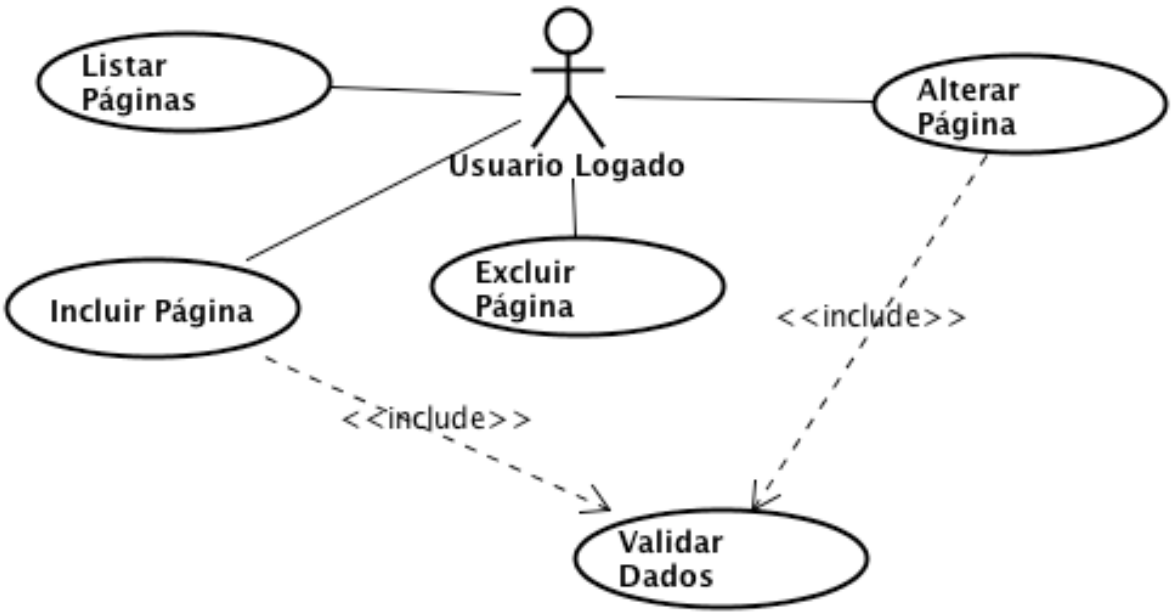

Figura 3 - Exemplo de diagrama de casos de uso

Atores e casos de uso são os principais elementos do diagrama. Um “caso de uso” é uma funcionalidade atômica do sistema, ou seja, ele representa uma funcionalidade completa, por exemplo, emissão de extrato em um terminal bancário é um caso de uso, já digitar a senha não, este segundo é um requisito. Já o “ator” representa uma entidade externa que interage com o software, por exemplo, o cliente do banco.

A Figura 4 apresentada abaixo descreve os elementos de um caso de uso:

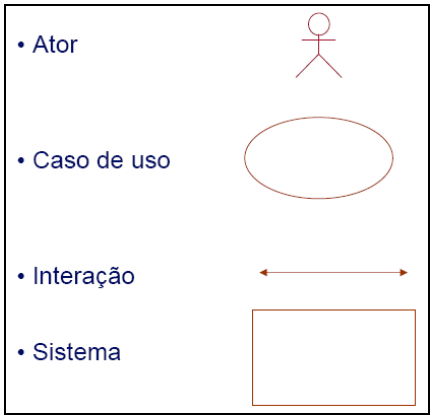

Figura 4 - Elementos de um diagrama de caso de uso

Os relacionamentos utilizados em diagramas de caso de uso podem ser classificados em:

- Relacionamento entre Caso de Uso e Ator (Comunica / Associa): Associa Casos de Uso e Atores indicando que há interação. É representada por uma linha ou seta unidirecional (Seta indica onde teve início a comunicação). A Figura 5 apresenta um exemplo deste relacionamento.

Figura 5 - Exemplo de relacionamento entre Caso de Uso e Ator

- Relacionamento entre Caso de Uso e Caso de Uso (Inclui/*Include*): Indica uma funcionalidade compartilhada por vários casos de uso. Se um caso de uso inicia ou inclui o comportamento de outro, considera-se que ele usa o outro. Propriedades básicas da inclusão: Realizar uma decomposição funcional; reduzir a complexidade de um caso de uso; o caso de uso básico não pode executar sem a inclusão; comportamento comum. A Figura 6 apresenta um exemplo deste relacionamento.

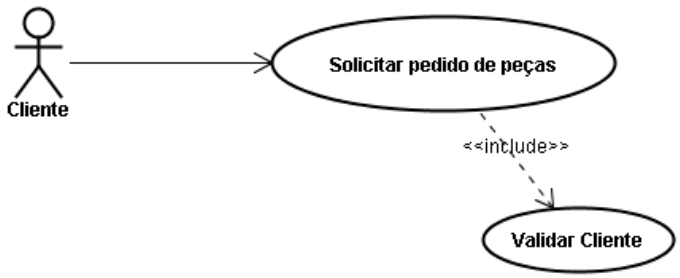

Figura 6 - Exemplo de relacionamento Inclui/*Include*
Fonte: Macoratti (2011).

- Relacionamento entre Caso de Uso e Caso de Uso (Estende/Extend): Descreve comportamento opcional de um caso de uso, executado sob certas condições. O caso de uso base pode ser executado mesmo sem a extensão. A Figura 7 apresenta um exemplo deste relacionamento.

Figura 7 - Exemplo de relacionamento Estende/*Extend*

- Relacionamento entre Caso de Uso e Caso de Uso (Generalização): Indica um caso de uso base que possui diferentes especializações e inclui comportamento ou sobrescreve o caso de uso base. A Figura 8 apresenta um exemplo deste relacionamento.

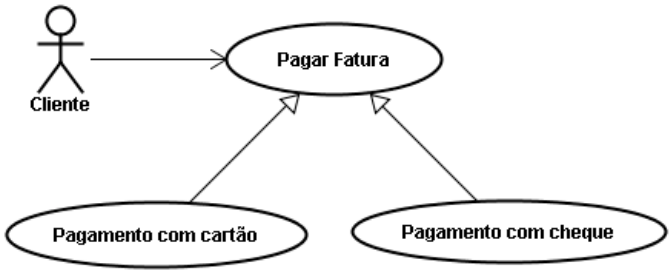

Figura 8 - Exemplo de relacionamento Generalização entre Casos de Uso

Fonte: Macoratti (2011).

- Relacionamento entre Ator e Ator (Generalização): Pode-se ter também os relacionamentos entre atores onde um ator especializado pode acessar os casos de uso de um Ator base. A figura 9 apresenta um exemplo deste relacionamento:

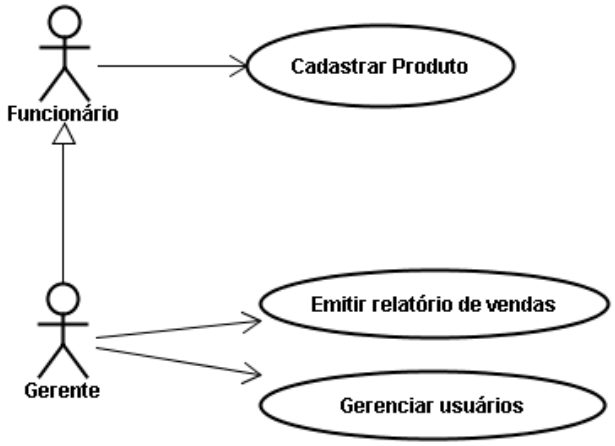

Figura 9 - Exemplo de relacionamento Generalização entre Atores

Fonte: Macoratti (2011).

2.6.2 Diagrama de Atividades

Segundo Schmuller (2004), a ideia de um diagrama de atividades é mostrar através de uma visão simplificada o que ocorre durante uma operação ou um processo. No diagrama de atividades, cada atividade é representada por uma espécie de retângulo com cantos arredondados, flechas que representam a transição de uma atividade para a próxima, e outros elementos para definir o início e fim da atividade, elementos de decisão, etc. A seguir, na Figura 10, verifica-se um exemplo que utiliza um elemento de decisão.

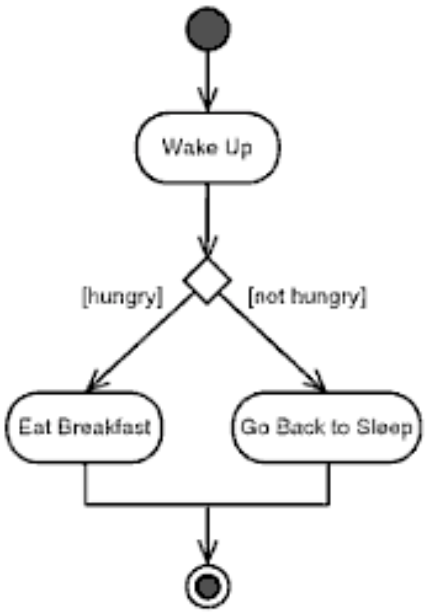

Figura 10 - Exemplo de um diagrama de atividades
Fonte: Schmuller (2004).

Ainda de acordo com Schmuller (2004), uma das grandes vantagens do diagrama de atividades é a habilidade de mostrar quem tem a responsabilidade por cada atividade no processo. Essa vantagem ocorre pelo uso das chamadas "*swimlanes*", que podem ser traduzidas como raias.

Na Figura 11 pode-se ver um exemplo de um diagrama de atividades que faz o uso das raias.

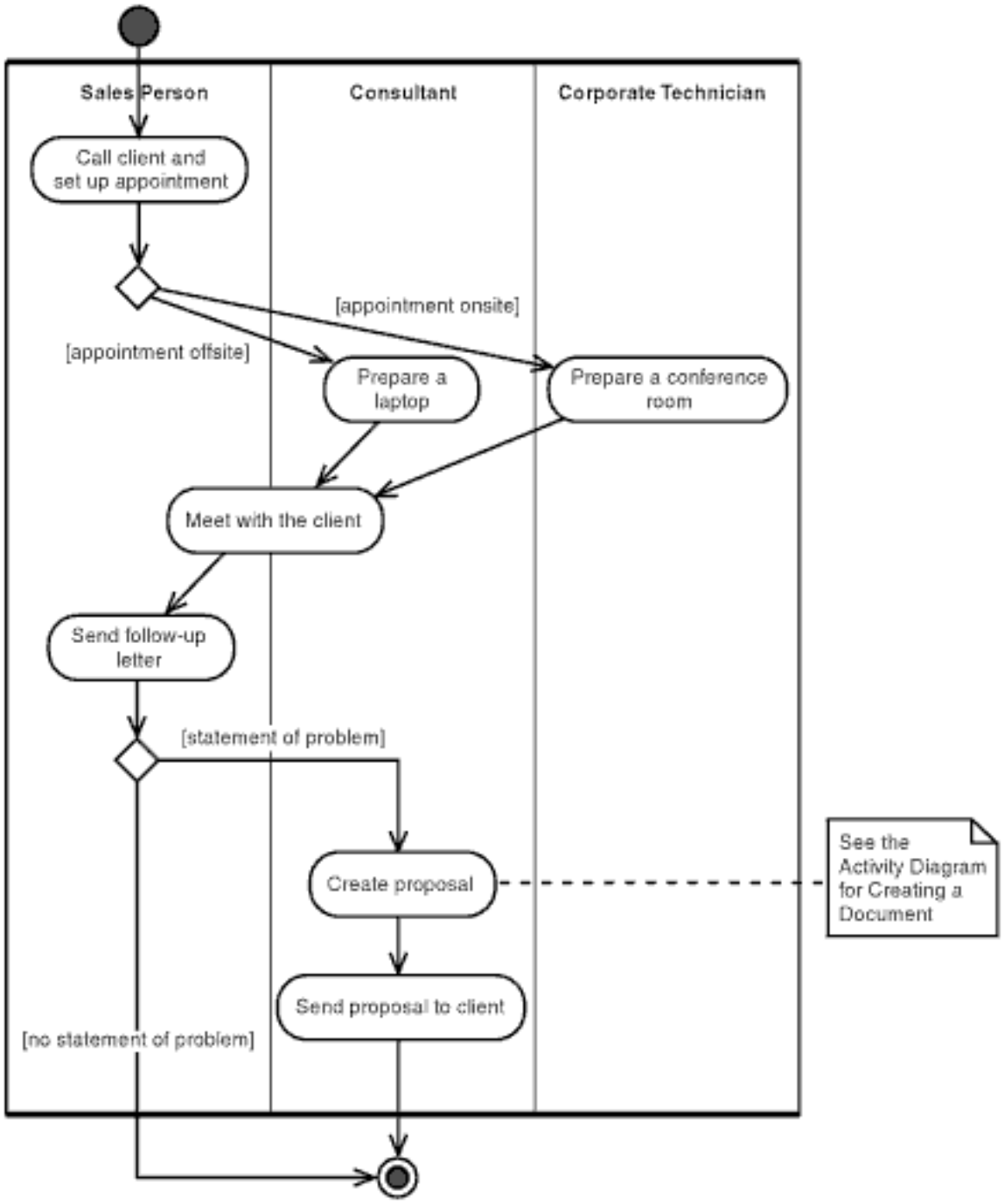

Figura 11 - Exemplo de um diagrama de atividades que faz uso de raias

FONTE: Schmuller (2004).

3 MATERIAIS E MÉTODOS

Este capítulo descreve os materiais e métodos utilizados no processo de elaboração deste trabalho.

3.1 MATERIAIS

A seguir está uma descrição dos materiais que foram necessários para a conclusão do trabalho proposto.

3.1.1 Java

De acordo com Schildt (2007), a linguagem Java foi concebida por James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan na empresa chamada Sun MicroSystems em 1991. Inicialmente era chamada de Oak, e só veio a receber o novo nome em 1995. É uma linguagem orientada a objetos.

O código gerado pelo compilador da linguagem não é um executável, e sim um *bytecode,* que segundo Schildt (2007) são um conjunto de instruções altamente otimizadas que são executadas na máquina virtual Java (JVM – *Java Virtual Machine*).

3.1.2 Android SDK

O desenvolvimento para Android é feito com o auxílio do Android SDK, o qual contém uma série de ferramentas e exemplos que auxiliam na criação de programas para a plataforma, juntamente com algumas bibliotecas nativas do Java, as quais o Android oferece suporte.

Inclusos no SDK, segundo Meyer (2010), estão:

- **APIs do Android:** Contêm as bibliotecas que possibilitam o desenvolvimento dos aplicativos para a plataforma. São as mesmas bibliotecas utilizadas pela Google no desenvolvimento das aplicações nativas;
- **Ferramentas de desenvolvimento:** Com essas ferramentas, o desenvolvedor pode compilar e depurar a aplicação;

- **Android SDK e AVD (*Android Virtual Device) Manager*:** *F*erramenta utilizada para a criação dos dispositivos virtuais (emuladores, Figura 12), e gerenciamento de APIs (Figura 13). Os emuladores utilizados para simular um dispositivo real com Android. Nele é possível a realização de testes, eliminando assim a necessidade do desenvolvedor possuir um dispositivo real para este propósito. No emulador, como no dispositivo físico, todas as aplicações rodam dentro da máquina virtual Dalvik.

Figura 12 - Emulador rodando o Android

Fonte: Jordan e Greyling (2011).

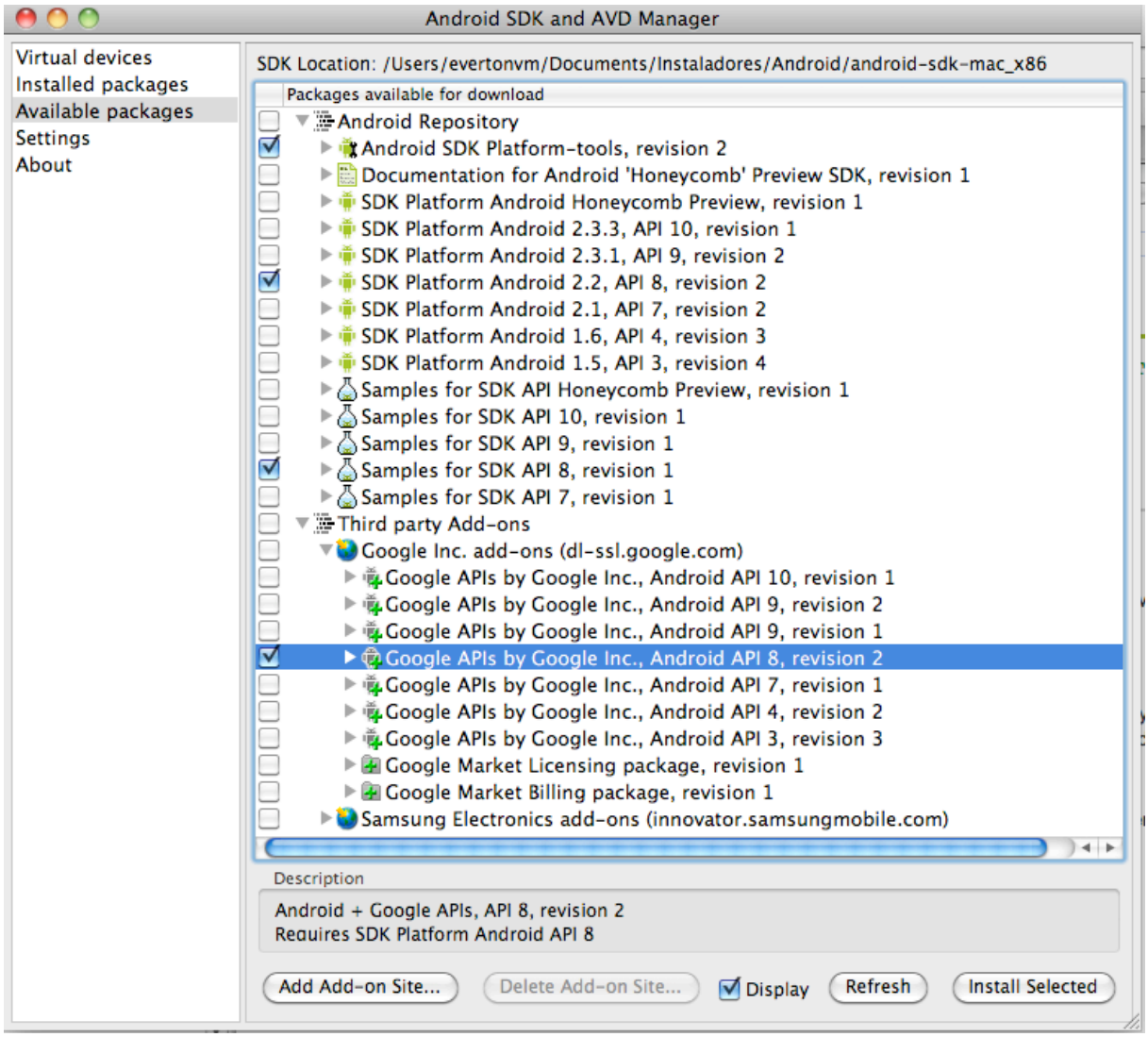

Figura 13 - Android SDK e AVD *Manager*

- **Documentação completa:** O SDK inclui uma extensa referência a nível de código, detalhando o que está incluso em cada pacote e classe e como utilizá-los;
- **Exemplos:** Uma seleção de aplicações que servem de exemplo para demonstrar algumas das possibilidades de desenvolvimento.

3.1.3 Eclipse IDE – *Integrated Development Environment*

"Eclipse é uma IDE para qualquer coisa e nada mais" (BURNETTE, 2005, pg. 01), significando que pode ser utilizada para o desenvolvimento de qualquer linguagem, e não apenas Java. Segundo Burnette (2005), ao final de 2001 o Eclipse

se tornou *open source*, e atualmente é controlado por uma organização sem fins lucrativos chama Eclipse *Foundation.* Essa ferramenta possui uma arquitetura baseada no uso de *plug-ins*[6].

A seguir, na 14, verifica-se uma imagem do ambiente de trabalho da Eclipse IDE.

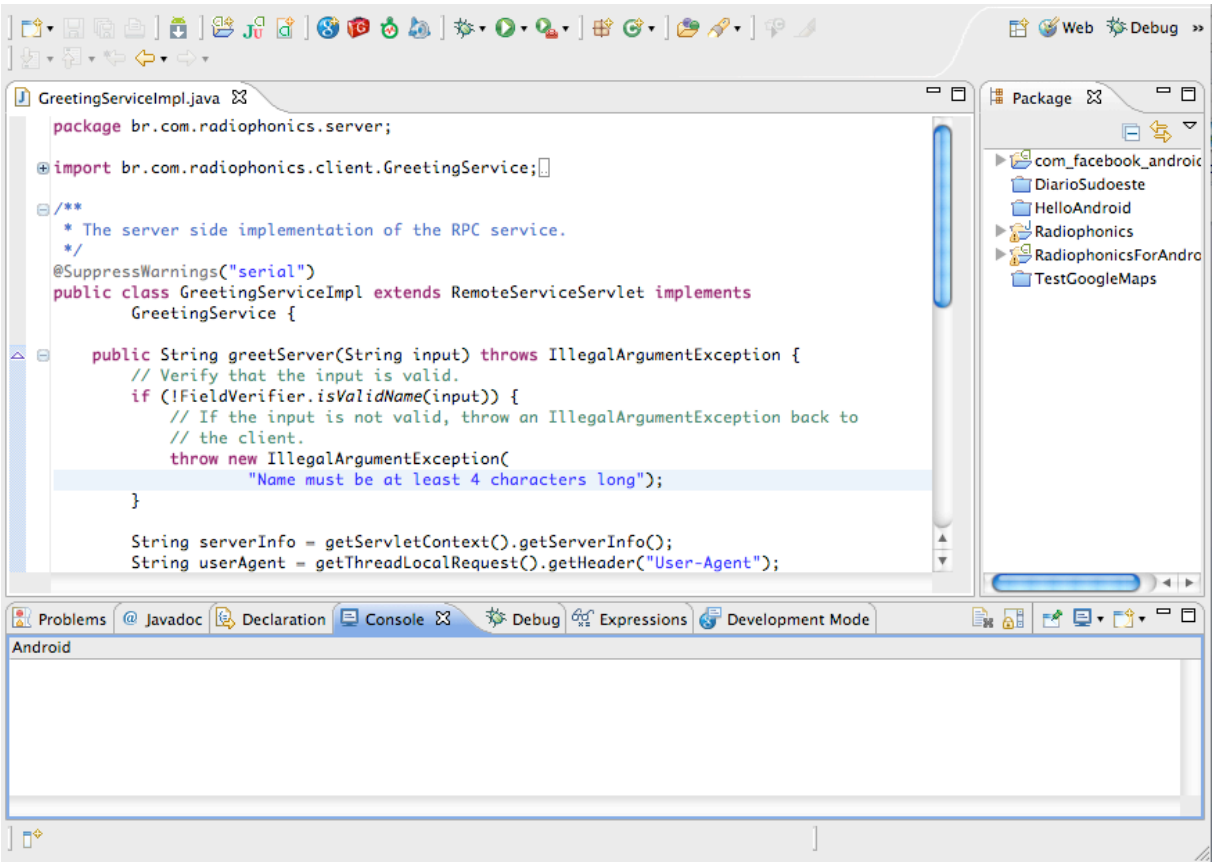

Figura 14 - Eclipse IDE

3.1.3.1 *Plug-in* ADT – *Android Development Tools*

Conforme Jordan e Greyling (2011), com o lançamento do Android SDK, a Google imediatamente disponibilizou o *plug-in* chamado ADT, que acrescenta à interface de desenvolvimento Eclipse funcionalidades que facilitam o processo de criação de aplicações para a plataforma Android neste ambiente.

Ainda segundo Jordan e Greyling (2011), o ADT é usado e mantido pelos desenvolvedores da plataforma Android.

---

[6] Programa que adiciona funcionalidades em programa maior.

Assim como a IDE Eclipse e o Android SDK, o ADT é *open source* e disponibilizado gratuitamente.

Verifica-se na Figura 15 a criação de um projeto para Android, fazendo uso do ADT no Eclipse.

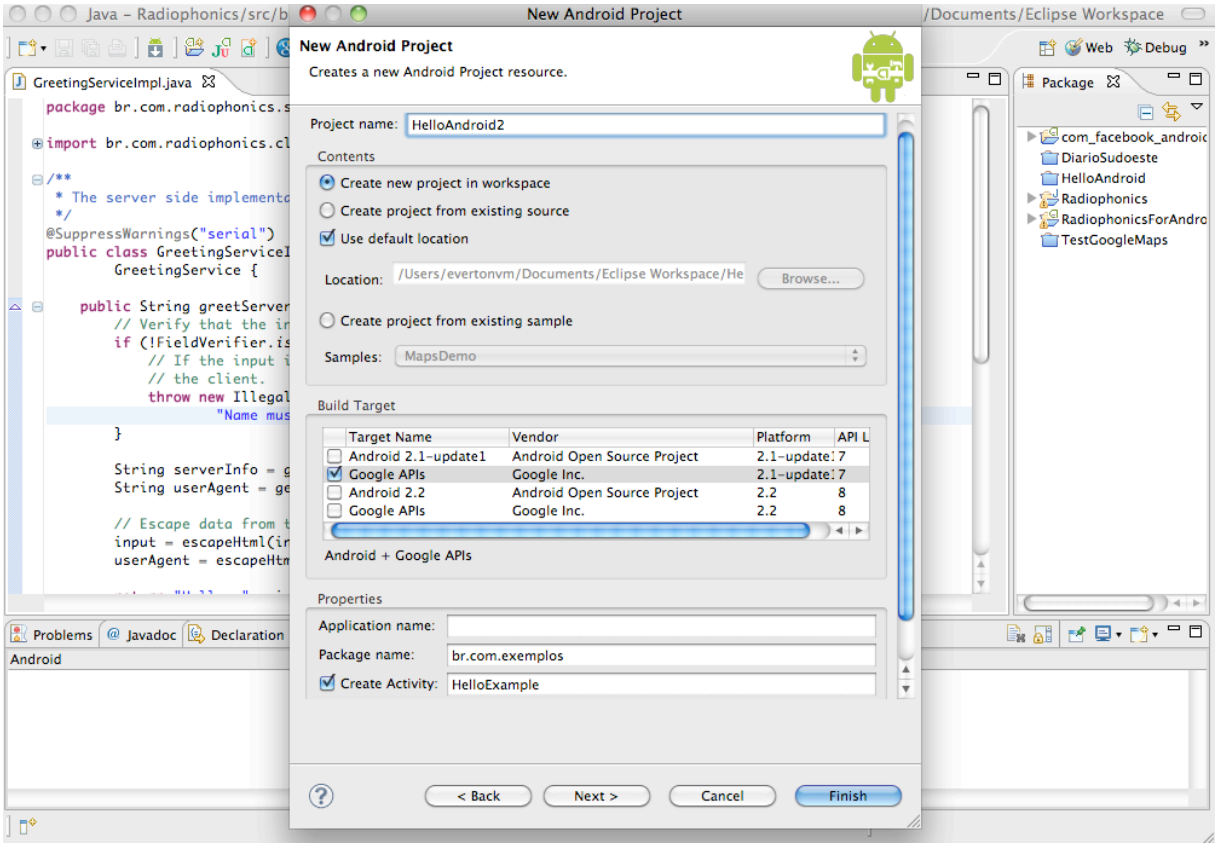

Figura 15 - Criando um projeto para Android no Eclipse através do *plug-in* ADT.

3.1.4 Google Maps

O Google *Maps* foi originalmente desenvolvido por dois irmãos dinamarqueses, Lars e Jens Rasmussen. Conforme explica Svennerberg (2010), os irmãos fundaram uma empresa chama *Where 2 Technologies*, que seria uma empresa dedicada a criação de soluções de mapeamento. Essa empresa foi adquirida pela Google em outubro de 2004, e então os irmãos criaram o Google *Maps*.

Segundo Svennerberg (2010), o serviço Google *Maps* foi apresentado em fevereiro do ano de 2005 e atualmente, entre tantos outros serviços de mapas, é o mais popular.

Na Figura 16 está um *print screen* da interface do Google *Maps* na tela de um dispositivo executando na plataforma Android.

Figura 16 - Google *Maps* rodando em um dispositivo com Android
Fonte: GOOGLE MAPS MOBILE WEBSITE

3.1.4.1 Google *Maps* API

Conforme Pereira e Silva (2009), o Android fornece um pacote chamado *com.google.android.maps* que contém classes relacionadas com a renderização, controle e informações sobre o Google *Maps* no dispositivo. Destacando-se a classe *MapView*, que mostra uma tela com o Google *Maps* quando adicionada no leiaute da aplicação. Possui também classes que possibilitam o controle do movimento do mapa na tela e a criação de itens que poderão ser sobrepostos no mapa.

3.1.5 Google *Navigation*

O Google *Navigation* é um aplicativo nativo da plataforma Android, pois o mesmo já vem por padrão instalado no sistema.

O aplicativo tem a função de traçar rotas de trânsito em um mapa para orientar o usuário. Possui recursos como sistema de navegação por voz, informações sobre o tráfego e seleção de rotas alternativas.

3.1.6 Facebook

O Facebook, fundado em 2004 por Mark Zuckerberg, é um site de relacionamento extremamente conhecido e de uso gratuito, ou seja, um modo de conectar-se com outras pessoas. E é um site que "combina o melhor dos blogs, fóruns *online* e grupos, compartilhamento de fotos e muito mais." (VEER, 2011, p. 01).

A Figura 17 mostra a página de login do Facebook.

Figura 17 - Site do Facebook – Página de login

3.1.6.1 Facebook SDK para Android

Segundo Soneff (2010), criador do Facebook SDK para Android, o projeto foi demonstrado pela primeira vez em maio de 2010 na Google I/O[7]. Segundo o autor, a biblioteca é extremamente simples e com apenas algumas linha de código Java é possível adicionar funcionalidades do Facebook a um aplicativo Android.

O SDK fornece acesso aos seguintes recursos da plataforma do Facebook:

- **Oauth 2.0:** "um protocolo aberto para permitir autorização de API segura de um modo simples e seguro por aplicações desktop e da web" (OAUTH WEBSITE, 2011). O uso do protocolo permite, por exemplo, o acesso a dados de um determinado usuário do Facebook a um outro aplicativo sem a necessidade do mesmo pedir pelas credenciais desse usuário;
- **Graph API:** Fornece acesso ao gráfico social do Facebook, que contém informações sobre os usuários e suas conexões sociais na rede;
- Publicação de mensagens no mural Facebook do usuário.

3.1.7 SSO - *Single Sign-On*

O termo *Single Sign-On,* que pode ser traduzido como "login único", de acordo com o Bertino e Takahashi (2011), refere-se ao uso de um único nome de *login* para conectar múltiplos sistemas. O sistema de login SSO pede que o usuário faça a autenticação utilizando suas credenciais apenas uma vez, não requisitando essas informações novamente nos próximos acessos.

3.1.8 JSON – *Javascript Object Notation*

Como explica Sampaio (2007), o JSON é um formato de troca de dados que também serve para serializar objetos. Este formato é amplamente utilizado em *Mashups*[8] para troca de dados entre funções de APIs existentes na Web.

---

[7] Google I/O é um evento que reúne milhares de desenvolvedores, focado na construção da próxima geração de aplicativos para *web*, dispositivos móveis e soluções empresariais utilizando tecnologias abertas para desenvolvimento *web*, entre elas o Android. (GOOGLE I/O WEBSITE).

[8] Diferentes serviços que podem funcionar simultaneamente, em cooperação.

“Um Objeto JSON é definido como um conjunto de nomes e valores separados por dois pontos e delimitados por vírgulas” (SAMPAIO, 2007, pg. 115).

Segundo Sampaio (2007), existem duas vantagens imediatas de usar JSON:

- É fácil de entender;
- É Javascript Não precisa de nada para poder transformar em objeto;

Na Figura 18 está um exemplo de objeto no formato JSON:

```
{
   "id": "660137427",
   "name": "Everton Mendes",
   "first_name": "Everton",
   "last_name": "Mendes",
   "link": "http://www.facebook.com/evertonvm",
   "username": "evertonvm",
   "gender": "male",
   "locale": "pt_BR"
}
```

Figura 18 - Objeto JSON

Fonte: FACEBOOK GRAPH WEBSITE

3.1.9 Ferramenta de Modelagem JUDE

JUDE é uma ferramenta de modelagem de dados UML (*Unified Modeling Language*) voltada para a criação de diagramas, entre eles estão: diagramas de classe, atividades, sequência e casos de uso.

A ferramenta foi criada pela companhia japonesa “ChangeVision” e a versão Community é de uso livre, porém com funcionalidades reduzidas comparada a versão Professional.

No JUDE é possível trabalhar com vários diagramas, são eles: o diagrama de classes, caso de uso, atividades, estado, sequência, implantação. comunicação e componentes.

Atualmente o *software* passou a ser chamado de “Astah”, devido a compra de direitos por uma outra empresa. Apesar da mudança o *software* mantém suas características originais, e ainda sustenta a versão “*Community*” porém com recursos reduzidos comparados a versão antiga. Alem da versão o Astah conta com as versões: Astah UML, Astah Professional e Astah share.

3.2 MÉTODOS

O desenvolvimento do presente trabalho foi dividido nas seguintes etapas:

- **Levantamento de requisitos** – recolhimento das informações necessárias para definir quais funcionalidades o sistema deve possuir;
- **Análise** – a partir dos requisitos foi será realizada a análise do sistema, criação do diagrama de Caso de Uso através da ferramenta *JUDE*;
- **Projeto** – após a conclusão da documentação, foi definida a forma de desenvolvimento do sistema e as ferramentas utilizadas no processo;
- **Implementação** – com todas as etapas anteriores finalizadas, a implementação começa com a utilização da linguagem Java juntamente com o Android SDK, Facebook API para Android e Google *Maps* API no ambiente de desenvolvimento Eclipse IDE;
- **Testes** – os testes forma realizados paralelamente ao desenvolvimento do sistema através da ferramenta AVD, que emula um dispositivo com o Android.

4 RESULTADOS E DISCUSSÕES

Nesse capítulo são apresentadas as etapas realizadas no decorrer do desenvolvimento do sistema, abrangendo desde a descrição do programa passando pela modelagem, implementação e testes.

4.1 DESCRIÇÃO DO SISTEMA

Trata-se de um sistema de arquitetura simples para dispositivos que executam na plataforma Android em sua versão 1.6 ou superior.

O sistema mostra a agenda de *shows* de uma banda musical específica, com informações como local, data e uma descrição do *show*. Nessa agenda, é permitido ao usuário selecionar um dos deles e interagir com ele das seguintes formas:

- Visualizando o local do *show* em um mapa (Google *Maps*);
- Visualizando a rota do local do usuário até o local do *show* (Google *Navigation*);
- Compartilhando informações do *show* selecionado no mural de seu perfil no Facebook.

Para o funcionamento de todos os recursos o sistema requer uma conexão com a Internet, porém se não houver uma conexão disponível será buscada uma versão local da agenda para mostrar ao usuário. Esta versão local é obtida a partir da última versão *online* da agenda acessada pelo usuário. Sendo assim, em seu primeiro acesso ao sistema o usuário deverá ter uma conexão com a Internet disponível, ou não será possível visualizar a agenda.

4.1.1 Requisitos do sistema

A seguir nos Quadros 3 e 4 estão os requisitos funcionais e não-funcionais do sistema.

Quadro 3 - Requisitos funcionais

Identificação	Descrição	Dependência	Importância
[RF1] Agenda de *shows*	O sistema deve permitir a visualização da agenda de *shows*	[RNF3] para primeiro acesso	Essencial
[RF2] Seleção de *show* para informações detalhadas	O sistema deve permitir a seleção de um determinado *show* na agenda para uma visualização detalhada.	[RF1]	Essencial
[RF3] Local do *show* no mapa	O sistema deve permitir a visualização do local do *show* selecionado no mapa.	[RF2], [RNF3] e [RNF6]	Essencial
[RF4] Rota até o local do *show*	O sistema deve permitir a visualização da rota partindo do local onde o usuário se encontra até o local onde o *show* selecionado será realizado.	[RF2], [RF3], [RNF3], [RNF4] e [RNF7]	Essencial
[RF5] *Login* SSO	O sistema deve permitir que o usuário faça *login* com suas credenciais do Facebook e autorize a aplicação.	[RNF3]	Essencial
[RF6] Compartilhamento de informações	O sistema deve permitir que o usuário compartilhe as informações do *show* selecionado no mural em seu perfil do Facebook.	[RF2], [RF5] e [RNF3]	Essencial
[RF7] Deslogar-se do sistema	O sistema deve permitir ao usuário deslogar-se do sistema.	[RF5] e [RNF3]	Essencial

Quadro 4 - Requisitos não-funcionais

Requisitos do Produto		
Identificação	**Descrição**	**Importância**
[RNF1] Usabilidade	O sistema deve ser simples o suficiente para ser utilizado em sua plenitude sem a necessidade de treinamento ou de um tópico de ajuda.	Essencial
	O sistema deverá guardar uma cópia local da agenda de *shows* sempre que a mesma for atualizada a partir de uma versão *online*.	Essencial
Requisitos de Interface		
Identificação	**Descrição**	**Importância**
[RNF2] Visualização em diferentes tamanhos	O sistema deverá oferecer uma interface adequada conforme o tamanho da tela de cada dispositivo.	Essencial
Requisitos de Hardware		
Identificação	**Descrição**	**Importância**
[RNF3] Conexão com a Internet	O dispositivo deve possuir a habilidade de conectar-se com a Internet.	Essencial
[RNF4] Sistema de	O sistema de localização do dispositivo deve estar ativado.	Desejável

GPS		
Requisitos de Software		
Identificação	**Descrição**	**Importância**
[RNF5] Android	Sistema operacional Android versão 1.6 ou superior.	Essencial
[RNF6] Google *Maps*	O aplicativo Google *Maps* (nativo) deve estar instalado no sistema operacional.	Essencial
[RNF7] Google *Navigation*	O aplicativo Google *Navigation* (nativo) deve estar instalado no sistema operacional.	Essencial
[RNF8] Aplicativo do Facebook	Ter o aplicativo oficial do Facebook instalado no sistema operacional.	Desejável

4.2 MODELAGEM DO SISTEMA

A modelagem do sistema foi focada principalmente nos diagramas da UML de caso de uso e o de atividades, pois constatou-se que tais diagramas ofereceriam uma melhor visão dos processos do sistema.

O diagrama de classes, apesar de ser considerado um dos mais importantes da UML, devido a simplicidade da estrutura do sistema, julgou-se desnecessário o seu uso.

4.2.1 Diagrama de Caso de Uso

No diagrama de caso de uso procurou-se apresentar uma visão simples e geral do sistema. Na Figura 19 está o diagrama de casos de uso utilizado na modelagem do sistema.

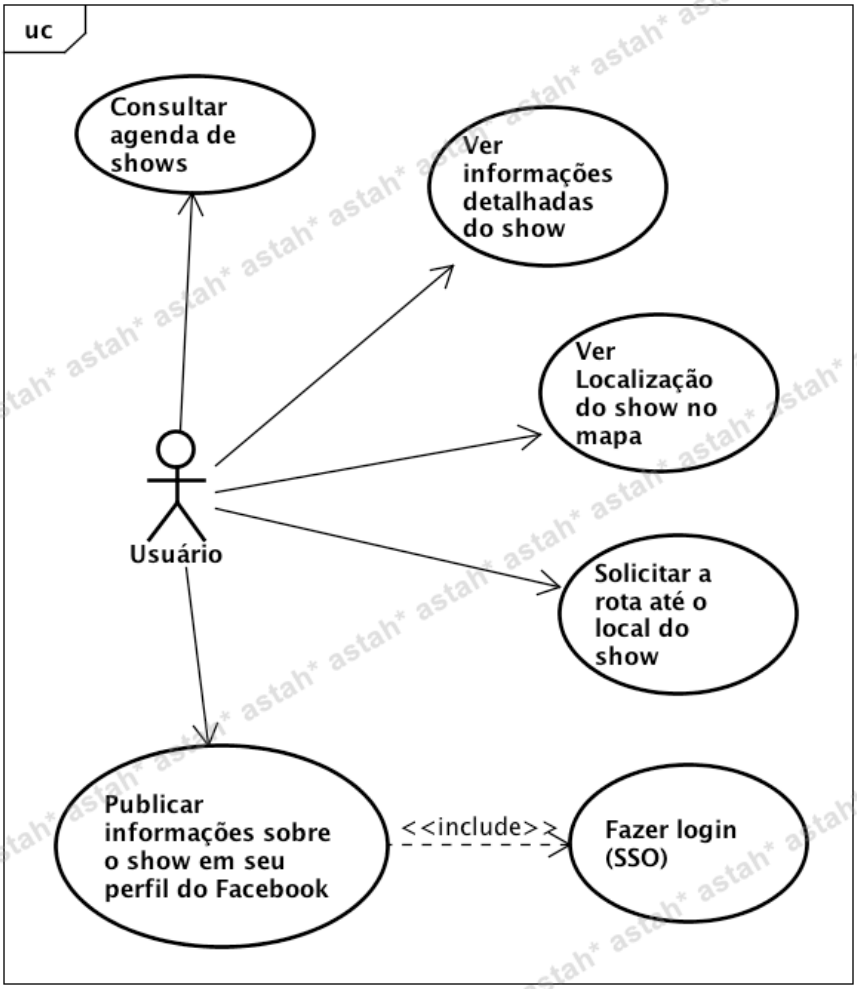

Figura 19 - Diagrama de Caso de Uso

A seguir está descrito de forma mais detalhada os casos de uso do sistema, constantes na Figura 19:

- **Consultar agenda de *shows*:** Ao carregar o sistema é apresentada a agenda de *shows* da banda e é permitido a seleção de um determinado *show* para obter mais detalhes;
- **Ver informações detalhadas do *show*:** O usuário tem acesso a informações como: local, data e descrição;
- **Ver localização do *show* no mapa:** Por meio dessa função é levado apresentada uma tela com um mapa com a marcação do local onde o *show* será realizado;
- **Solicitar a rota até o local do show:** A partir da visão do mapa é possível, clicando-se na marcação do local, selecionar a opção “como

chegar?”. Essa função leva o usuário a aplicação Google *Navigation*, que fica responsável por localizar o usuário e traçar a rota até o local;

- **Publicar informações sobre o *show* em seu perfil do Facebook:** Possibilita que o usuário divulgue o *show* selecionado em seu perfil do Facebook. Esse caso de uso possui uma relação do tipo *include* com o caso de uso “Fazer login (SSO)“, tornando o *login* obrigatório ao usuário para utilizar-se da função de publicação no perfil;
- **Fazer login (SSO):** Permite o usuário fazer *login* na aplicação utilizando-se de suas credenciais do Facebook. Com o *login* do tipo SSO o usuário deve autorizar algumas permissões de acesso a informações de seu perfil no Facebook que forem solicitadas pela aplicação.

4.2.2 Diagrama de Atividades

Com os diagramas de atividade foi possível obter uma visão mais aprofundada, porém não complexa, do funcionamento do sistema. Na Figura 20 está uma visão das operações que ocorrem na abertura do sistema.

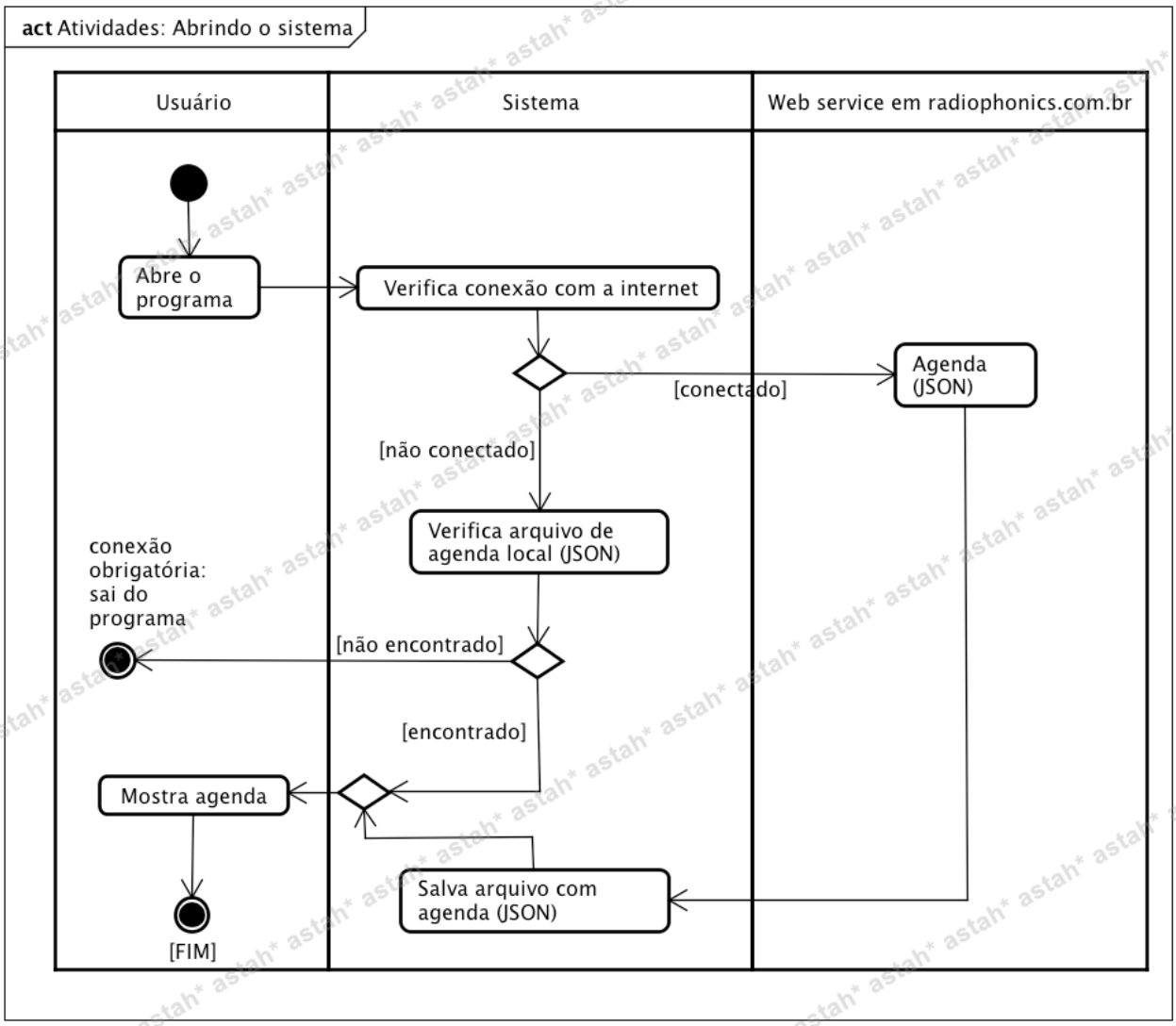

Figura 20 - Diagrama de atividades que ocorrem no processo de abertura do sistema

As atividades apresentadas no diagrama da Figura 20 mostram que ao encontrar uma conexão com a Internet o sistema busca no *web service* uma versão atualizada da agenda e antes de mostrar ao usuário e as salva localmente em um arquivo. Deste modo é possível que nos próximos acessos o usuário abra o sistema e consulte a agenda sem a necessidade de estar conectado à Internet.

Na Figura 21, observam-se as atividades que são realizadas quando o usuário decide publicar as informações sobre o *show* em seu perfil da rede social Facebook.

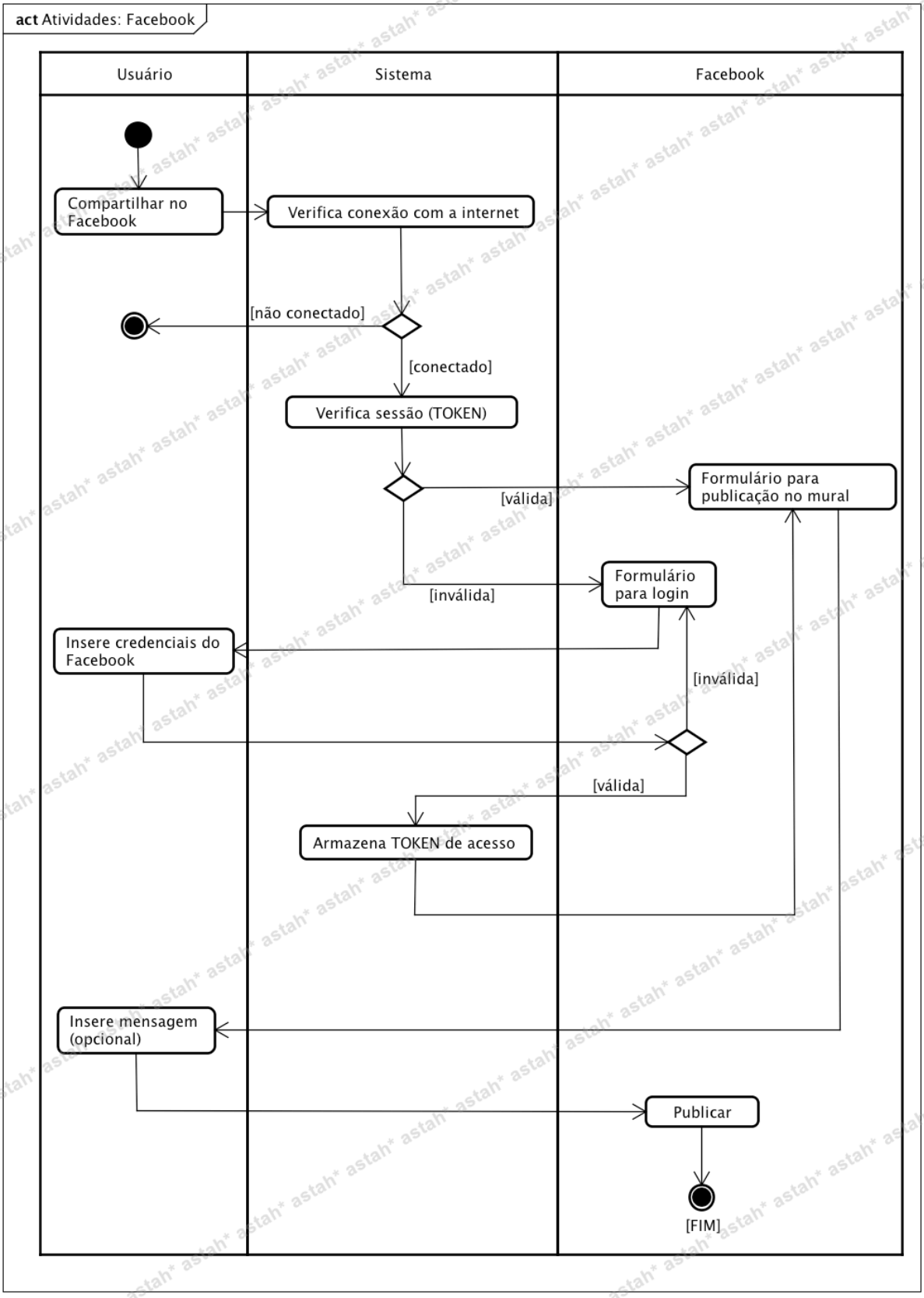

Figura 21 - Diagrama de atividades de publicação em perfil do Facebook

Por meio do diagrama da Figura 21 é possível perceber que a função de publicação é interrompida se o usuário não possuir uma conexão ativa com a Internet. Outro aspecto importante é que quando as credenciais inseridas pelo usuário são aceitas pelo Facebook, é retornado um *token* de acesso. Esse *token*, garante o acesso as informações do usuário quando solicitadas ao sistema do Facebook. Observa que o sistema armazena o *token* quando o mesmo é obtido.

4.3 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção será apresentada uma visão mais aprofundada e técnica do funcionamento do sistema. Isso é feito pela apresentação de elementos como telas e partes de códigos.

4.3.1 Preparando o Ambiente de Trabalho

A seguir será descrito de forma sequencial as etapas necessárias à preparação do ambiente de trabalho.

a) Verificar a existência do kit de desenvolvimento Java (JDK), o qual deve estar instalado para possibilitar o desenvolvimento com a linguagem Java;

b) Realizar a instalação da IDE Eclipse;

c) Download do Android SDK em formato *zip*, versão para Mac OS X (Intel);

d) Instalação do ADT *Plugin* para Eclipse;

e) Adição de componentes ao Android SDK: No Android SDK alguns componentes devem ser instalados separadamente, para tal fim utiliza-se da ferramenta AVD *Manager*, que busca por versões atualizadas dos componentes e pode ser iniciada facilmente com o uso do ADT *Plugin*. Dentre os componentes que foram instalados pelo AVD *Manager* estão:

e.1) Android SDK *Tools* (*revision* 11) e *Platform-tools* (*revision* 5), que são ferramentas para tarefas como compilação e depuração das aplicações. A Android SDK *Tools* já é instalada juntamente com o SDK, porém recebe atualizações constantes e deve ser atualizada sempre que possível;

e.2) SDK *Platform* Android 1.6 (API 4, *revision* 3): Contém as bibliotecas para desenvolvimento. Apesar de ser considerada uma versão antiga do Android, foi optado pela versão 1.6, na tentativa de restringir ao mínimo o uso

da aplicação;

e.3) Google APIs *by* Google Inc. (Android API 4, *revision* 2): API com bibliotecas para o uso de mapas nativamente na aplicação.

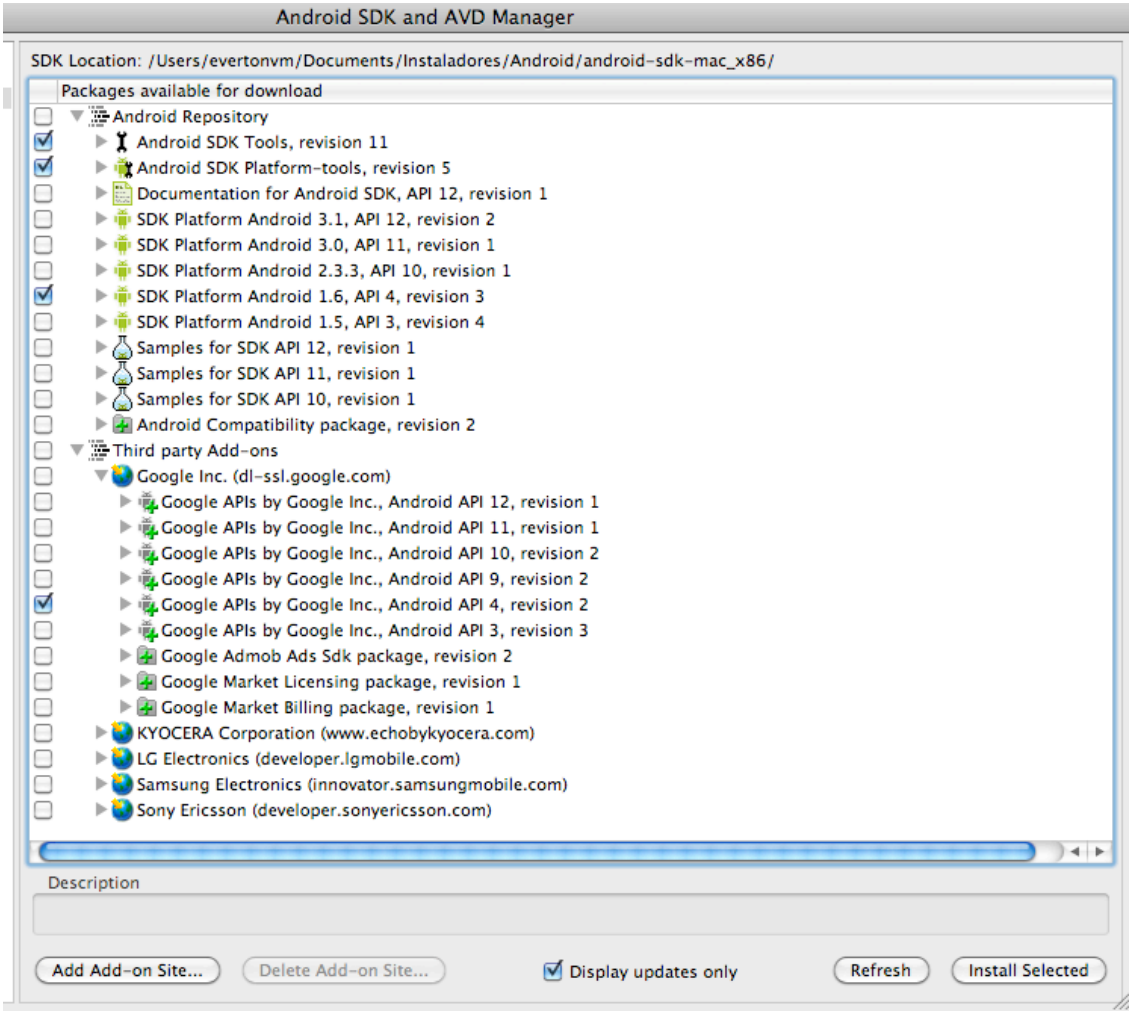

Figura 22 - AVD *Manager* com os pacotes instalados selecionado

Após as etapas descritas acima, o ambiente se encontra pronto para o início do desenvolvimento.

4.3.2 Estrutura do Projeto

Na Figura 23, observa-se a estrutura das classes e pacotes do projeto.

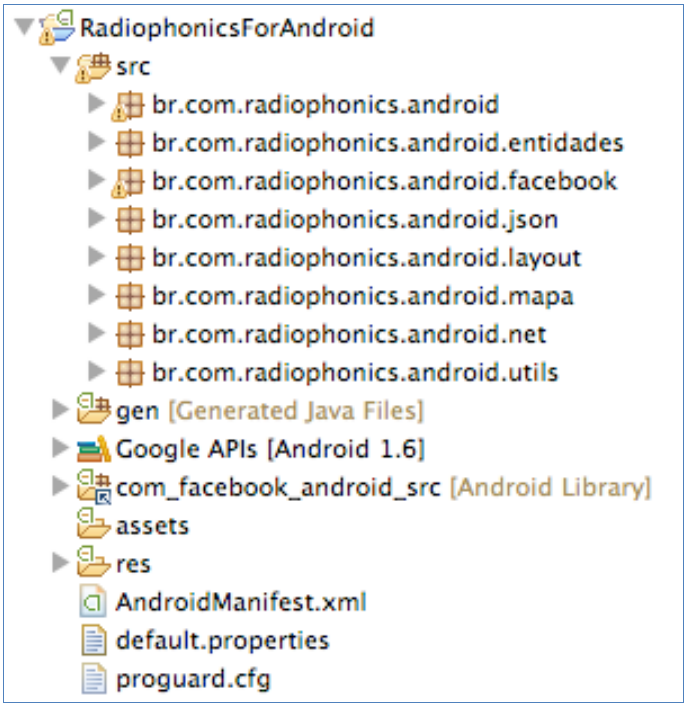

Figura 23 - Estrutura do projeto

A seguir é apresentado uma breve descrição do conteúdo de cada pacote:

a) br.com.radiophonics.android: neste pacote encontram-se as activities que são as classes que representam as telas (interfaces) em um sistema Android;

b) br.com.radiophonics.android.entidades: Possui classes que definem o formato dos objetos como: Agenda, Show e Local;

c) br.com.radiophonics.android.facebook: Contém uma classe chamada FacebookHelper. Essa classe herda de uma classe externa chamada Facebook que pertence ao projeto Facebook (Figura 24) que está contido no Facebook SDK.

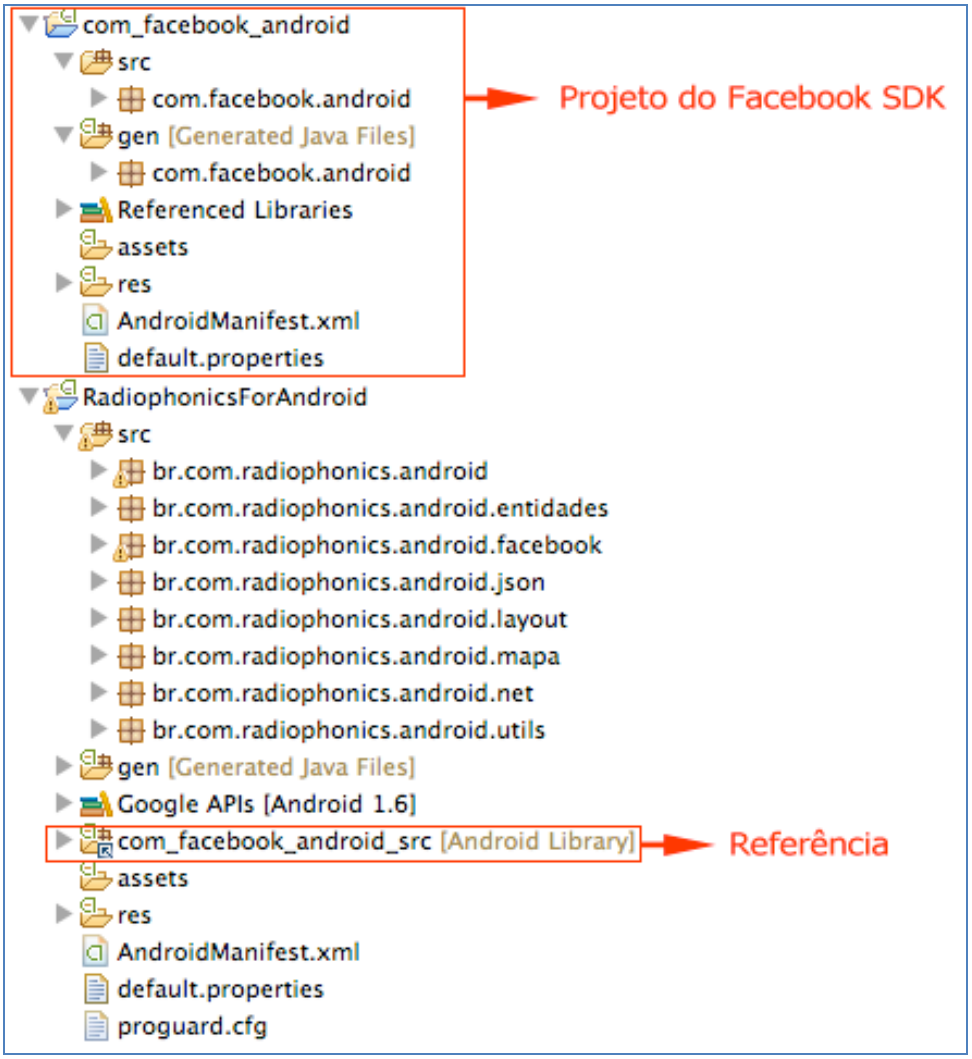

Figura 24 - Estrutura do projeto indicando referência a outro projeto

d) br.com.radiophonics.android.json: Trata da conversão da string no formato JSON para um array de objetos do tipo Show;

e) br.com.radiophonics.android.layout: Responsável por inflar a agenda de shows com objetos do tipo Show;

f) br.com.radiophonics.android.mapa: Possui classes que auxiliam o uso de mapas pela aplicação adicionando funcionalidades como marcação de um determinado ponto no mapa;

g) br.com.radiophonics.android.net: Encarregado das comunicações com o web *service*;

h) br.com.radiophonics.android.utils: Classes com diversa funcionalidades, como por exemplo a de facilitar a exibição de mensagens na tela.

4.3.3 Interfaces e Códigos

No Android, as interfaces são criadas no formato XML (*Extensible Markup Language*), no entanto também há suporte para a criação em tempo de execução. Na aplicação desenvolvida, optou-se por utilizar sempre que possível o formato XML, deixando de fora apenas os conteúdos dinâmicos, os quais devem ser controlados em tempo de execução.

Na Figura 25 está um exemplo da inserção de uma imagem na interface utilizando o formato XML, e logo após um exemplo de como é feito o acesso e alteração das propriedades dessa imagem no código Java (tempo de execução).

Arquivo XML

```
<ImageView android:id="@+id/imgFacebookLogout"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_alignParentBottom="true"
           android:layout_alignParentRight="true"
           android:src="@drawable/facebook_logout"
           android:onClick="imgFacebookLogout_OnClick"/>
```

Classe Java

```
imgFacebookLogout = (ImageView) findViewById(R.id.imgFacebookLogout);
imgFacebookLogout
        .setVisibility(fbHelper.isSessionValid(this) ? View.VISIBLE
                : View.INVISIBLE);
```

Figura 25 - Criação de elemento de interface XML e acesso via classe

Na Figura 25 verifica-se que o código na classe Java faz referência a imagem criada em outro arquivo que está no formato XML, então em uma cláusula condicional define a visibilidade da imagem, ou seja, se a imagem deverá ou não estar visível na interface. Neste caso em particular, trata-se de uma imagem que é utilizada como botão para que o usuário possa fazer o *logout* do sistema.

4.3.3.1 Tela de Carregamento ou Abertura do Programa

Ao abrir o sistema será exibido ao usuário uma espécie de tela de abertura, a qual exibe uma imagem do logotipo da banda. Observa-se essa tela na Figura 26.

Figura 26 - Tela de abertura do programa

A tela de abertura, além de exibir uma imagem ao usuário, também tem a função de carregar a agenda de *shows*. Suas atividades podem ser melhor entendidas observando-se o diagrama exibido na Figura 20.

A seguir, na Figura 27, uma amostra do código de inicialização da tela.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.setContentView(R.layout.splash);

    boolean hasConnection = Util.hasInternetConnection(this);
    new AgendaAsync(hasConnection).start();
}

private class AgendaAsync implements Runnable {
```

Figura 27 - Amostra de código da tela de abertura do sistema

Na Figura 28 observa-se uma chamada a classe *AgendaAsync*, que é uma classe interna que implementa a interface *Runnable*, permitindo uma execução assíncrona. Desta forma, a tela com a imagem é exibida ao usuário enquanto o processo de carregamento da agenda ocorre em uma outra *thread*.

Ao término com sucesso das operações nesta tela, o usuário será automaticamente direcionado para a tela com a agenda de *shows*.

Na Figura 28 é possível observar o código que faz a chamada da tela com a agenda. Na chamada é passado como parâmetro um *array* dos objetos do tipo *Show* que foram obtidos. Este código pertence a classe *AgendaAsync*.

```
Intent i = new Intent(getApplicationContext(),
        ListaShows.class);
i.putExtra("shows", agenda.getShows());
startActivity(i);
finish();
```

Figura 28 - Código de chamada para a tela de agenda

4.3.3.1.1 Processo de Busca e Conversão da Agenda de Shows

A responsável pela maioria operações nesse processo é a classe *Agenda* que está contida no pacote br.com.radiophonics.entidades.

Dentre os métodos presentes nesta classe, destaca-se o método “carregaAgenda(), o qual é encarregado da comunicação com o *web service* para a busca da agenda atualizada, ou o carregamento do arquivo local de agenda, dependendo da existência ou não de conexão com a Internet.

O método “carregaAgenda()“ ao verificar a presença de conexão com a Internet, faz uso de uma função da classe *WebService* pertencente ao pacote br.com.radiophonics.android.net. Na Figura 29, pode-se observar esta função, que faz a conexão com o *web service* retornando uma resposta em forma de *string*.

```
public static String get(String url) throws Exception {
    HttpGet httpGet = new HttpGet(url);

    HttpParams httpParameters = new BasicHttpParams();

    //seta os timeouts para o caso de algum problema
    //em alcançar o host de destino
    HttpConnectionParams.setConnectionTimeout(httpParameters, 3000);
    HttpConnectionParams.setSoTimeout(httpParameters, 5000);

    HttpResponse response = new DefaultHttpClient(httpParameters)
            .execute(httpGet);

    return EntityUtils.toString(response.getEntity());

}
```

Figura 29 - Função da classe WebService

No caso da busca pela agenda, a função apresentada na Figura 29, retornará uma *string* no formato JSON.

Sempre que é adquirida uma versão online da *string*, a mesma é armazenada, possibilitando acessos futuros sem uma conexão com a Internet estar disponível.

A Figura 30 mostra um exemplo de uma JSON *string* que pode ser retornada.


```
[{"ShowId":48,
      "Data":"\/Date(1308711600000)\/",
      "Descricao":"",
      "Local":{
            "LocalShowId":3,
            "Nome":"Empório São Francisco",
            "Endereco":"Rua Presidente Carlos Cavalcanti, 1138",
            "Cidade":"Curitiba",
            "Estado":"PR",
            "Pais":"Brasil"}
},
{"ShowId":66,
      "Data":"\/Date(1308970800000)\/",
      "Descricao":"",
      "Local":{
            "LocalShowId":6,
            "Nome":"Sheridans Irish Pub",
            "Endereco":"Rua Bispo Dom José, 2315",
            "Cidade":"Curitiba",
            "Estado":"PR",
            "Pais":"Brasil"}
}]
```


Figura 30 - Exemplo de retorno da função carregaAgenda

A *string* mostrada na Figura 30 é, neste caso, a representação no formato JSON de uma agenda com apenas dois *shows*. Essa *string* deve então ser convertida em um coleção de objetos do tipo *Show*.

A classe responsável pela conversão da *string* no formato JSON para uma coleção de *shows* é a *AgendaParser* que está contida no pacote br.com.radiophonics.android.json. Essa classe faz uso de classes do pacote org.json que está presente no Java para transformar porções de uma JSON *string* em objetos que então poderão ser convertidos para os tipos corretos.

4.3.3.2 Tela da Agenda de shows

Após o carregamento do sistema o usuário é automaticamente levado a tela com a agenda de *shows*, a qual pode ser observada na Figura 31.

Figura 31 - Tela com agenda de shows

Um ponto importante a ser observado nesta tela, é que se a agenda mostrada for uma versão desatualizada, ou seja, carregada a partir de um arquivo local, ao se retornar a tela, se for detectada uma conexão ativa com a Internet, a agenda será automaticamente atualizada, e um novo arquivo será salvo com a versão obtida. Na Figura 32, observa-se uma imagem da agenda sendo atualizada.

Figura 32 - Atualizando a agenda

4.3.3.3 Tela de Detalhes do Show Selecionado

Ao selecionar um determinado show na tela de agenda, o usuário é levado a uma nova tela com os detalhes do show selecionado, como é possível observar na Figura 33.

Figura 33 - Tela mostrando detalhes do show

A partir da tela apresentada na Figura 33 é possível acessar funcionalidades do sistema como, compartilhar no Facebook e mostrar a localidade no mapa. O botão de *logout* do Facebook que é visualizado no canto inferior direito da Figura 33, que tem sua visibilidade definida conforme o usuário se encontra ou não conectado com suas credenciais do Facebook, ou seja, se o usuário não tiver feito *login* no Facebook o botão ficará invisível. A solicitação de *login* no Facebook ocorre quando o usuário clica no botão “Compartilhar no Facebook” e o mesmo não está logado.

Na tela de detalhes do *show* também é possível, ao clicar-se no ícone do mapa no canto superior direito, ser levado a uma outra tela com uma visualização do local do *show* em um mapa.

As respectivas funções de compartilhamento no Facebook e visualização no mapa serão apresentadas em maiores detalhes nos itens 4.3.3.4 e 4.3.3.5 a seguir.

4.3.3.4 Tela de Compartilhamento no Facebook

Na Figura 34 observa-se a tela de *login* no Facebook. É importante ressaltar que a interface e o funcionamento desta tela são controlados pelo Facebook SDK para Android, o qual o projeto faz referência.

Figura 34 - Tela de *login* do Facebook no Android

Outro ponto a se observar é que se o usuário tiver instalado o aplicativo oficial do Facebook disponível no Android *Market*, a tela de *login* é mostrada de maneira nativa e não fazendo uso de uma *webview* como mostrado na Figura 34.

Na Figura 35 está uma imagem da tela nativa de *login* no Facebook.

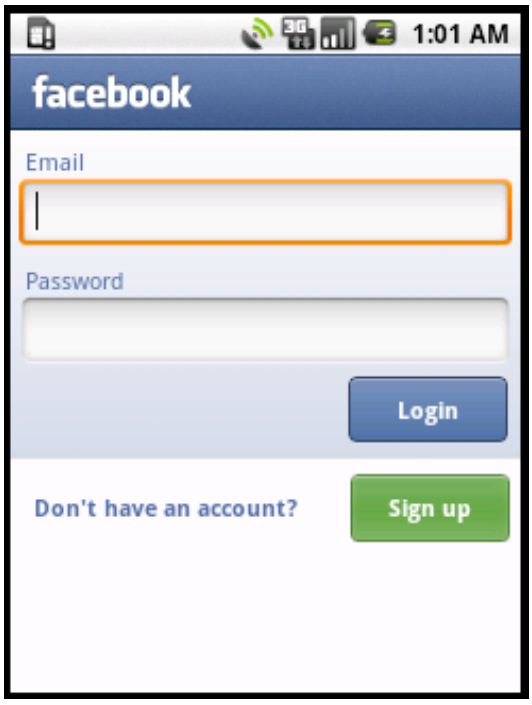

Figura 35 - Tela nativa de *login* no Facebook

Ao fazer o *login* o usuário é levado uma outra tela na qual deve autorizar o acesso aos seus dados pela aplicação. Essa tela pode ser vista na Figura 36.

Figura 36 - Tela de autorização do Facebook[9]

Por padrão a tela de autorização pede apenas pelo acesso às informações básicas do usuário, no entanto, ao chamar a função de autorização é possível pedir autorização para acessar outros dados.

Observa-se na Figura 37 o trecho de código que faz a chamada do formulário de autorização.

```
this.authorize(activity, new String[] { "email", "publish_stream", "offline_access" },
        new DialogListener() {
```

Figura 37 - Chamada ao formulário de autorização do Facebook

Após o usuário ter concedido as devidas autorizações à aplicação, ele é finalmente levado a tela que permite a publicação no perfil do Facebook. Nesta tela

[9] Imagem editada para possibilitar a visualização de todas as autorizações.

é permitido ao usuário inserir uma mensagem personalizada ou deixar a mensagem padrão que é definida pelo aplicativo e que consiste na frase: “Eu vou!!“.

A seguir na Figura 38 uma imagem da tela de publicação no perfil.

Figura 38 - Tela de publicação no perfil

4.3.3.5 Tela de Visualização no Mapa

A partir da tela de detalhes do *show* é possível o usuário optar por visualizar a localização do mesmo em um mapa, sendo assim levado à tela mostrada a seguir na Figura 39.

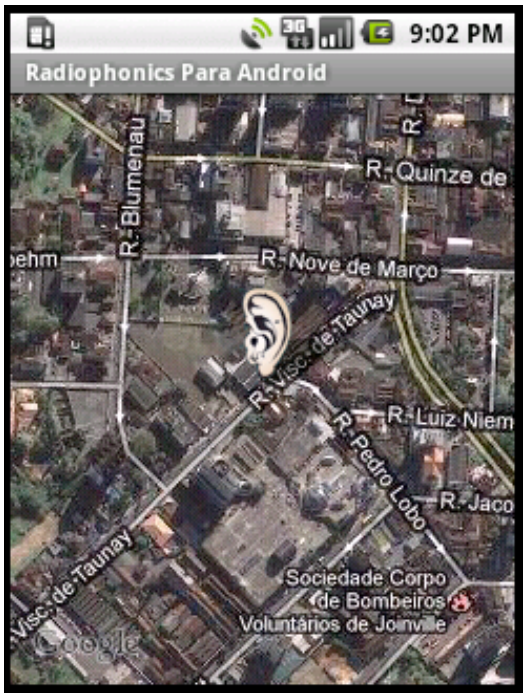

Figura 39 - Tela de Visualização no mapa

A classe responsável pela marcação do ponto no mapa é a *MapaUtil*, que está localizada no pacote br.com.radiophonics.android.mapa. Essa classe faz uso de várias outras classes do pacote com.google.android.maps, inclusive a referência a esse pacote se faz necessária para a visualização de mapas em uma tela nativa.

A seguir, a Figura 40 mostra a função que faz a conversão de um endereço no mapa para um objeto do tipo com.google.android.maps.GeoPoint, o qual é necessário para a marcação do local no mapa.

```
public GeoPoint getGeoPoint(final String endereco) throws IOException {
    Geocoder gCoder = new Geocoder(this.activity, Locale.getDefault());

    List<Address> enderecos = gCoder.getFromLocationName(endereco, 1);
    if (enderecos.size() > 0) {
        GeoPoint gPoint = new GeoPoint((int) (enderecos.get(0)
                .getLatitude() * 1E6), (int) (enderecos.get(0)
                .getLongitude() * 1E6));

        return gPoint;
    }
    return null;
}
```

Figura 40 - Função recebe um endereço e retorna um objeto *GeoPoint*

A partir da obtenção do objeto *GeoPoint* é possível, por exemplo, marcar o local no mapa com alguma imagem. As imagens no Android são definidas pela classe *Drawable*. Sendo assim, na Figura 41 observa-se uma função que recebe por parâmetro uma imagem do tipo *Drawable*, um *Geopoint* e um *AlertDialog*, que é a caixa de dialogo que será mostrada quando o usuário tocar na imagem que marca o local.

```
private MyItemizedOverlay criaOverlay(Drawable icon, GeoPoint gPoint,
        AlertDialog dialog) {
    MyItemizedOverlay itmOverlay = new MyItemizedOverlay(icon, dialog);
    // cria o overlay
    OverlayItem ovItem = new OverlayItem(gPoint,
            this.activity.getString(R.string.map_overlay_title), "");

    itmOverlay.addOverlay(ovItem);

    return itmOverlay;
}
```

Figura 41 - Criação do *overlay* para marcação de local no mapa

A função demonstrada acima retorna o *overlay* que contém a imagem que será adicionada ao mapa.

Quando a solicitação de marcação de um local é feita pela tela de visualização do mapa, é criado um o objeto do tipo *AlertDialog* que é passado como parâmetro nessa solicitação. O *AlertDialog* é uma caixa de dialogo com a opção “Como chegar?”, que ao ser selecionada pelo usuário, o mesmo é direcionado a um aplicativo nativo do Android chamado Google *Navigation.* Essa aplicativo é responsável por localizar o usuário e definir a rota até o local do *show*.

Na Figura 42 observa-se a chamada ao aplicativo Google *Navigation*.

```
private void showNavigation() {
    Intent i = new Intent(Intent.ACTION_VIEW,
            Uri.parse("google.navigation:q=" + this.show.getEnderecoMapa()));
    this.startActivity(i);
}
```

Figura 42 - Chamada ao aplicativo Google *Navigation*

4.4 DISCUSSÃO

No decorrer do desenvolvimento do projeto, os testes foram realizados principalmente em emulador, sempre utilizando o mesmo tamanho de tela. Desta maneira constata-se que existe grande possibilidade do aplicativo desenvolvido necessitar de alterações para se adequar aos diferentes tamanhos de telas encontrados em dispositivos com Android. No entanto, essas possíveis modificações limitam-se a questões de interface, não interferindo diretamente nas funções desempenhadas pela aplicação.

Por meio de alguns testes realizados em um dispositivo real, constatou-se que a velocidade de processamento ao executar o aplicativo é extremamente maior do que a vista nos testes com o emulador.

Importante ressaltar que atualmente o Google não fornece uma API do Google *Navigation* para o Android. No entanto, existem algumas maneiras de invocar a aplicação Google *Navigation* a partir de outro aplicativo Android. A maneira utilizada no desenvolvimento desse projeto não é documentada pelo Google, e portanto pode-se dizer que não tem seu uso recomendado. Entretanto, optou-se pelo uso do método por oferecer simplicidade de acesso.

5 CONCLUSÃO

Apesar de se tratar de um sistema com arquitetura relativamente simples, a etapa de implementação do sistema apresentou-se como um grande obstáculo ao desenvolvimento do projeto. Durante esta etapa, deparou-se com várias dificuldades, principalmente na integração das funcionalidades do Facebook, o qual pode-se dizer que ainda não possui um SDK sólido para Android, no entanto, foi de extrema ajuda para realizar a integração com o aplicativo.

O emulador do sistema Android apresentou-se como uma das principais ferramentas ao desenvolvimento do projeto. Nas poucas oportunidades de testar o aplicativo em um dispositivo real, pôde-se concluir que o emulador se fez uma opção extremamente confiável para ambiente de testes. Descartando, assim, a grande diferença de desempenho, onde o dispositivo real realiza as tarefas muito mais rapidamente, porém não interferindo em nada em suas funcionalidades.

O desenvolvimento do presente trabalho tornou-se possível graças ao ótimo suporte oferecido pela documentação para desenvolvedores da plataforma Android, e ao grande número de livros disponíveis que tratam do assunto. Desta maneira acredita-se ter alcançado o objetivo proposto neste trabalho.

6 PROSPECÇÕES FUTURAS

Futuramente pretende-se acrescentar algumas funcionalidades ao aplicativo desenvolvido, visando proporcionar uma melhor experiência ao usuário. A seguir observa-se algumas das principais modificações a serem feitas:

- Adequação da interface do sistema para a adaptação a diferentes tamanhos de telas e resoluções;
- Integração com outras redes sociais;
- Aumentar a integração com a rede social do Facebook, possibilitando ao usuário ver quais de seus amigos na rede estão utilizando o aplicativo.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BERTINO, Elisa; TAKAHASHI, Kenji. **Identity management:** Concepts, Technologies, and Systems. Artech House, 2011.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do usuário.** Rio de Janeiro: Campus, 2000, 7ª edição.

BURNETTE, Ed. **Eclipse IDE:** Pocket Guide. O'Reilly Media, Inc., 2005.

CAMPIONE, M. and WALRATH, K. **The Java™ ttorial - object-oriented programming for the internet**, Addison-Wesley, 1996.

CARDOSO FILHO, Jorge; JANOTTI JÚNIOR, Jeder. **A Música popular massiva, o mainstream e o underground: trajetórias e caminhos da músicas na cultura midiática.** IN: FREIRE FILHO, João e JANOTTI JUNIOR, Jeder. ***Comunicação e Música Popular Massiva***. Salvador: Edufba, 2006.

CYSNEIROS, L.M.; LEITE, J.C.S.P. **Definindo requisitos não funcionais**, 1997.

FACEBOOK GRAPH WEBSITE. Disponível em: < https://graph.facebook.com/evertonvm >. Acessado em: 28/05/2011.

FACEBOOK Developers. **Graph API.** Disponível em: < http://developers.facebook.com/docs/reference/api/ >. Acessado em: 30/05/2011.

GOOGLE I/O WEBSITE. Disponível em: < http://www.google.com/events/io/2011/about.html >. Acessado em: 29/05/2011.

GOOGLE MAPS MOBILE WEBSITE. Disponível em: <http://www.google.com/mobile/maps/>. Acessado em: 30/05/2011.

HASHIMI, Sayed; KOMATINENI, Satya; MACLEAN, Dave. **Pro Android 2.** Apress, 2010.

JACOBSON, Ivar, BOOCH, Grady, RUMBAUGH, James **The unified software development process**. Indianapolis: Editora Addison-Wesley, 1999.

JORDAN, Lucas; GREYLING, Pieter. **Pratical android projects.** Apress, 2011.

KARCH, Marziah. **Android for work:** Productivity for Professionals. Apress, 2010.

KRUCHTEN, Philippe Kruchten; **The rational unified process** , 2a edição. Addison Wesley, 2000.

LARMAN, Craig. **Utilizando UML e padrões–uma introdução à análise e ao projeto orientado a objetos e ao processo unificado**. Porto Alegre: Editora Bookman, 2004.

MACORATTI **Modelando sistemas em UML - casos de uso.** Disponível em: <http://www.macoratti.net/net_uml2.htm>. Acesso em 17/06/2011.

MEYER, Reto. **Professional android 2 application development.** Indianapolis, Wiley Publishing, 2010.

OAUTH WEBSITE. Disponível em: < http://oauth.net/ >. Acessado em: 30/05/2011.

PEREIRA, Lucio Camilo Oliveira; SILVA, Michel Lourenço da. **Android para desenvolvedores.** Rio de Janeiro: Brasport, 2009.

SAMPAIO, Cleuton. **Web 2.0 e mashups:** Reinventando a Internet. Brasport, 2007.

SCHILDT, Herbert. **Java:** A Beginner's Guide, Fourth Edition. McGraw-Hill, 2007.

SCHMULLER, Joseph. **Sams teach yourself uml in 24 hours,** Third Edition**.** Sams Publishing, 2004.

SHALLOWAY, Alan; TROTT, James R. **Explicando padrões de projeto**: uma nova perspectiva em projeto orientado a objeto. Porto Alegre: Bookman, 2004.

SILVA, Alberto Manuel Rodrigues da; VIDEIRA, Carlos Alberto Escaleira. **UML, Metodologias e ferramentas CASE**. Prod. Centro Atlântico, 2001, ISBN: 972-8426-36-4.

SILVA, Ricardo Pereira. **UML 2: modelagem orientada a objetos**. Florianópolis: Visual Books, 2007.

SONEFF, Steven. **The official facebook SDK for android.** Publicado em 27/05/2010. Disponível em: <http://developers.facebook.com/blog/post/385/>. Acessado em 29/05/2011.

SVENNERBERG, Gabriel. **Beginning google maps API 3.** Apress, 2010.

TONSIG, Sérgio Luiz. **Apostila: métodos orientados a objetos.** São Paulo. 2000.

UNICAMP WEBSITE. Disponível em: < http://www.dca.fee.unicamp.br/cursos/PooJava/classes/umlclass.html >. Acessado em: 05/06/2011.

VEER, E. A. Vander. **Facebook: the missing manual**, Third Edition**.** O'Reilly, 2011.

WAZLAWICK, Raul S. **Análise e projeto de sistemas de informação orientados a objetos.** 5 ed. Rio de Janeiro: Elsevier, 2004.