

**LIGA DE ENSINO DO RIO GRANDE DO NORTE  
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RN – FARN  
Especialização em Desenvolvimento de Sistemas Corporativos**

**DIEGO MORAIS DE MEDEIROS RODRIGUES**

**USO DE DISPOSITIVOS MÓVEIS COM ANDROID E WIFI  
PARA REALIZAÇÃO DE AVALIAÇÕES ACADÊMICAS**

Natal / RN  
2011

DIEGO MORAIS DE MEDEIROS RODRIGUES

**USO DE DISPOSITIVOS MÓVEIS COM ANDROID E WIFI  
PARA REALIZAÇÃO DE AVALIAÇÕES ACADÊMICAS**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos, da Faculdade Natalense para o Desenvolvimento do RN – FARN, como requisito para a obtenção do título de Especialista.

Orientador: Rômulo Fagundes, Msc.

Natal / RN  
2011

Rodrigues, Diego Morais de Medeiros  
Uso de Dispositivos Móveis com Android e WIFI para Realização de  
Avaliações Acadêmicas / Diego Morais de Medeiros Rodrigues. – Natal,  
2011.  
63 f.

Orientador: Rômulo Fagundes (Msc.)  
Monografia (Especialização em Desenvolvimento de Sistemas  
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio  
Grande do Norte.

1. Sistemas de Informação - Monografia. 2. Dispositivos Móveis –  
Monografia. 3. Aplicativos - Avaliações Acadêmicas – Monografia.  
I. Título.

RN/FARN/BC

CDU 004.7

DIEGO MORAIS DE MEDEIROS RODRIGUES

**USO DE DISPOSITIVOS MÓVEIS COM ANDROID E WIFI  
PARA REALIZAÇÃO DE AVALIAÇÕES ACADÊMICAS**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos, da Faculdade Natalense para o Desenvolvimento do RN – FARN, como requisito para a obtenção do título de Especialista.

Aprovada em 5 de Outubro de 2011, pela banca examinadora composta pelos seguintes membros:

**Prof. Msc. Rômulo Fagundes Cantanhede**

Faculdade Natalense para o Desenvolvimento do RN – FARN

**Orientador**

**Prof. Msc. Gleydson de Azevedo Ferreira Lima**

Faculdade Natalense para o Desenvolvimento do RN - FARN

**Membro Examinador**

**Prof. Itamir de Moraes Barroca Filho**

Faculdade Natalense para o Desenvolvimento do RN - FARN

**Membro Examinador**

## **AGRADECIMENTOS**

Em primeiro lugar, minha gratidão a Jesus pelo amor, cuidado e seu auxílio em mais essa empreitada; recebidos sem nenhum merecimento.

À minha amável família e à companhia indispensável de Camila. É infrutífero tentar expressar toda gratidão por seus atos, compreensão e amor.

Ao professor Rômulo por sua orientação paciente e informações preciosas. Consegui compilar o \*.jar professor! O problema era a plataforma 64bits.

*“Aquele que diz: Eu conheço-o e não guarda os seus mandamentos é mentiroso, e nele não está a verdade. Mas qualquer que guarda a sua Palavra, o amor de Deus está nele verdadeiramente aperfeiçoado; nisto conhecemos que estamos nele. Aquele que diz que está nele também deve andar como ele andou”.*

*1 João 2.4-6*

## RESUMO

Explora o uso de dispositivos móveis, com o sistema operacional Android embarcado, na realização de avaliações acadêmicas sobre a infra-estrutura de uma rede computacional WiFi para a comunicação de dados com um aplicativo Java Desktop no papel de servidor. Aborda o desenvolvimento de aplicações com o uso de técnicas da Engenharia de Software e *Unified Modeling Language* – UML. Expõe o uso de tecnologias como persistência de dados com o uso de banco dados SQLite e o padrão DAO (Data Access Object) para o desenvolvimento de aplicações e acesso a dados. Demonstra o uso do padrão *singleton* de desenvolvimento e sua utilidade em aplicativos Android para facilitar a troca de mensagens entre seus objetos. Discute detalhes da implementação das aplicações Desktop e Android propostas e aspectos da comunicação por meio do uso de uma rede *ad-hoc* WiFi através do uso de um protocolo exclusivo e gerenciamento dos dispositivos conectados para realização de broadcast seletivo de dados para os dispositivos. Aponta para os benefícios do uso de uma infra-estrutura composta por um computador Desktop e dispositivos móveis com Android embarcado para o professor e alunos, respectivamente, na realização de avaliações acadêmicas.

**Palavras chave:** Java. Android. WiFi.

## ABSTRACT

It explores the use of mobile devices, with the operational system Android embedded, in the realization of academic tests, over the infra-structure of a WiFi computational network to communication of data to a Java Desktop application in the server role. It approaches the application development with Software Engineering and Unified Modeling Language – UML techniques. It exposes the use of data persistence through the use of a SQLite database and the Data Access Object software development and data access pattern. It demonstrates the use of the singleton software development pattern and its utility in Android applications to facilitate the exchange of messages between its objects. It discusses details in the implementation of Java Desktop and Android applications purposed and communication aspects using an *ad hoc* WiFi network through exclusive protocol and connected devices management to realize a selective broadcast of data to the devices. It points to the benefits of using an infra-structure made by a Desktop computer and mobile devices with Android embedded to the teacher and students, respectively, in the realization of academic tests.

**Key-words:** Java. Android. WiFi.

## LISTA DE ILUSTRAÇÕES

|                                                                                             |    |
|---------------------------------------------------------------------------------------------|----|
| Figura 1. Visão geral das aplicações propostas por este trabalho .....                      | 14 |
| Figura 2. Gráfico com a participação dos OS para dispositivos móveis .....                  | 19 |
| Figura 3. Ciclo de vida de uma <i>Activity</i> .....                                        | 22 |
| Figura 4. Ciclo de vida de <i>services</i> vinculados e não vinculados .....                | 23 |
| Figura 5. Exemplo do arquivo <i>AndroidManifest</i> (XML) .....                             | 24 |
| Figura 6. Esquema da infra-estrutura que proporciona a portabilidade do Java .....          | 26 |
| Figura 7. A integração promovida pela linguagem XML.....                                    | 27 |
| Figura 8. Exemplo de um arquivo em linguagem XML.....                                       | 28 |
| Figura 9. Arquitetura cliente/servidor de um SGBD tradicional .....                         | 29 |
| Figura 10. Arquitetura SQLite de conexão a dados .....                                      | 30 |
| Figura 11. Logo e ícone das aplicações desenvolvidas .....                                  | 35 |
| Figura 12. Diagrama de Classes de Domínio do aplicativo Saara Desktop.....                  | 38 |
| Figura 13. Esquema de atribuição de um <i>MAC Address</i> a uma Interface de Rede.....      | 39 |
| Figura 14. Diagrama dos principais Casos de Uso do Saara Desktop.....                       | 40 |
| Figura 15. Diagrama de Classes de Domínio da Aplicação Saara Android .....                  | 42 |
| Figura 16. Diagrama dos principais Casos de Uso do Saara Android .....                      | 43 |
| Figura 17. Exemplo da estrutura de uma classe DAO .....                                     | 44 |
| Figura 18. Janela de Cadastro de Avaliações.....                                            | 45 |
| Figura 19. Tela principal do Saara Desktop .....                                            | 46 |
| Figura 20. Estrutura da classe <i>ServidorSaara.java</i> .....                              | 47 |
| Figura 21. Estrutura da classe <i>GerenciadorDeConexoes.java</i> .....                      | 48 |
| Figura 22. Tela inicial da aplicação Saara Android .....                                    | 49 |
| Figura 23. Mensagem quando uma aplicação Android deixa de responder.....                    | 50 |
| Figura 24. Trecho de código com a criação de um thread em Android .....                     | 51 |
| Figura 25. Exemplo da estrutura XML de uma avaliação do Saara .....                         | 52 |
| Figura 26. Estrutura da Classe Repositório de Mensagens .....                               | 54 |
| Figura 27. Fragmento da classe <i>SaaraDatabase</i> .....                                   | 55 |
| Figura 28. Exemplo de uma consulta ao Banco de Dados em uma aplicação Android .....         | 56 |
| Figura 29. Estrutura da Classe <i>RespostaCursor</i> .....                                  | 56 |
| Figura 30. Diagrama das atividades de interação entre as aplicações Desktop e Android ..... | 57 |

## SUMÁRIO

|                                                                    |           |
|--------------------------------------------------------------------|-----------|
| <b>1. INTRODUÇÃO .....</b>                                         | <b>12</b> |
| <b>2. REFERENCIAL TEÓRICO.....</b>                                 | <b>15</b> |
| <b>2.1. ENGENHARIA DE SOFTWARE.....</b>                            | <b>15</b> |
| 2.1.1. Unified Modeling Language – UML.....                        | 15        |
| 2.1.2. Ferramentas CASE (Computer-Aided Software Engineering)..... | 16        |
| <b>2.2. WIFI .....</b>                                             | <b>18</b> |
| <b>2.3. ANDROID.....</b>                                           | <b>19</b> |
| 2.3.1. Componentes de um Aplicativo Android .....                  | 20        |
| 2.3.2. O Ciclo de Vida de uma <i>Activity</i> .....                | 20        |
| 2.3.3. O Ciclo de Vida de um <i>Service</i> .....                  | 22        |
| 2.3.4. O Arquivo <i>AndroidManifest</i> (.xml) .....               | 24        |
| <b>2.4. JAVA.....</b>                                              | <b>25</b> |
| <b>2.5. EXTENSIBLE MARKUP LANGUAGE - XML.....</b>                  | <b>26</b> |
| <b>2.6. SEGURANÇA .....</b>                                        | <b>28</b> |
| <b>2.7. SQLITE .....</b>                                           | <b>28</b> |
| 2.7.1. <i>Features</i> que o SQLite possui .....                   | 31        |
| 2.7.2. <i>Features</i> que o SQLite não possui .....               | 32        |
| 2.7.3. Quando o SQLite não é uma boa escolha .....                 | 33        |
| <b>3. OBJETIVOS.....</b>                                           | <b>34</b> |
| <b>3.1. OBJETIVO GERAL .....</b>                                   | <b>34</b> |
| <b>3.2. OBJETIVOS ESPECÍFICOS.....</b>                             | <b>34</b> |
| <b>4. DESENVOLVIMENTO .....</b>                                    | <b>35</b> |
| <b>4.1. APRESENTAÇÃO DAS APLICAÇÕES E CONCEITOS .....</b>          | <b>35</b> |
| <b>4.2. MODELAGEM DAS APLICAÇÕES .....</b>                         | <b>36</b> |
| 4.2.1. Especificação de Requisitos do Saara Desktop .....          | 36        |
| 4.2.2. Diagrama de Classes do Saara Desktop .....                  | 37        |
| 4.2.3. Casos de Uso do Saara Desktop .....                         | 39        |
| 4.2.4. Especificação de Requisitos do Saara Android .....          | 41        |

|                                                                                         |           |
|-----------------------------------------------------------------------------------------|-----------|
| 4.2.5. Diagrama de Classes do Saara Android.....                                        | 41        |
| 4.2.6. Casos de Uso do Saara Android .....                                              | 42        |
| <b>4.3. IMPLEMENTAÇÃO DA APLICAÇÃO SAARA DESKTOP .....</b>                              | <b>43</b> |
| <b>4.4. IMPLEMENTAÇÃO DA APLICAÇÃO SAARA ANDROID.....</b>                               | <b>48</b> |
| 4.4.1. Repositório de Mensagens: Comunicação entre Objetos de uma Aplicação Android ... | 52        |
| 4.4.2. Persistência de Dados em Android .....                                           | 54        |
| <b>5. RESULTADOS .....</b>                                                              | <b>57</b> |
| <b>5.1. EXECUÇÃO E INTERAÇÃO DAS APLICAÇÕES DESKTOP E ANDROID.....</b>                  | <b>57</b> |
| <b>6. CONCLUSÃO .....</b>                                                               | <b>59</b> |
| <b>7. REFERÊNCIAS.....</b>                                                              | <b>61</b> |
| <b>8. GLOSSÁRIO.....</b>                                                                | <b>63</b> |

## 1. INTRODUÇÃO

Atualmente hardware e software vêm adquirindo um caráter de mobilidade sem precedentes, e isso não parece ser uma tendência passageira, o senso comum é de que essas “novidades” vieram para ficar.

Esta afirmação não requer uma fonte especializada, afinal de contas, é de domínio público o que aconteceu recentemente com a revolução na comunicação trazida pelos telefones celulares, antes deles, um telefonema normalmente partia de telefones fixos. Contudo, às vezes, era preciso nos comunicar enquanto estávamos na rua. Tudo bem, alguém teve a brilhante idéia dos telefones públicos – apelidados no Brasil de “orelhões” – entretanto, isso não era o bastante.

Em meados da década de 90, no Brasil, houve a popularização dos aparelhos de telefonia celular, a idéia de comunicação em qualquer lugar, a qualquer hora, que já era uma necessidade latente, foi muito bem recebida e sabe-se o que aconteceu.

Pois bem, vemos o mesmo fenômeno se repetir agora, todos nós estávamos habituados com os bons e velhos computadores, em casa e no trabalho, mas surgiram os notebooks como uma solução para atender a uma crescente demanda por mobilidade, mas, a exemplo dos telefones públicos, ainda não foram uma solução plena e definitiva.

Neste momento, vivemos uma revolução provocada pelos *tablets* e *smartphones* que chegaram ao mercado brasileiro e, assim como em outros mercados, estão sendo muito bem aceitos e muito procurados. Estes aparelhos conjugam o conceito de mobilidade e informática, e permitem aos seus usuários consumir informação em qualquer lugar usando tecnologias de conexão sem fio como WiFi, Bluetooth ou 3G.

Para contribuir ainda mais com o sucesso desta revolução, o governo brasileiro anunciou um incentivo para baratear o custo dos *tablets* para o consumidor final. Conforme vemos em notícia vinculada pela SERPRO: “Medida Provisória que autoriza incentivos fiscais para os *tablets* reduz em até 31% os impostos que incidem sobre o produto fabricado no Brasil”. (2011)

Conclui-se que um cenário em que a difusão destas novas tecnologias de mobilidade é uma realidade próxima, inevitável e irreversível. Este fato é importante para este trabalho em função do impacto que certamente existirá na academia. E estas mudanças já

começaram a ocorrer, como se percebe na ação da Universidade carioca Estácio de Sá, que distribuiu *tablets* para seus alunos no começo de 2011.

“A Estácio é a primeira universidade do Brasil a distribuir *tablets* aos seus alunos. (...) Com a implantação dessa nova ferramenta no ensino estima-se uma economia de 214 milhões de folhas papel por ano”. (ESTÁCIO DE SÁ, 2011)

A tendência é de que estas inovações tecnológicas causem impacto na maneira através da qual professor e aluno se relacionam academicamente, e, é o uso de tecnologias que serão decisivas sobre este impacto que este trabalho se propõe a explorar.

Atualmente, dois tipos básicos de avaliação são muito comuns nas instituições de ensino: (a) avaliações físicas, que são aquelas avaliações impressas em papel em que as repostas são assinaladas pelo aluno ou (b) avaliações presenciais virtuais, em que a prova se constitui em perguntas através das quais o aluno, devidamente credenciado, navega e assinala as suas respostas fazendo uso de um software.

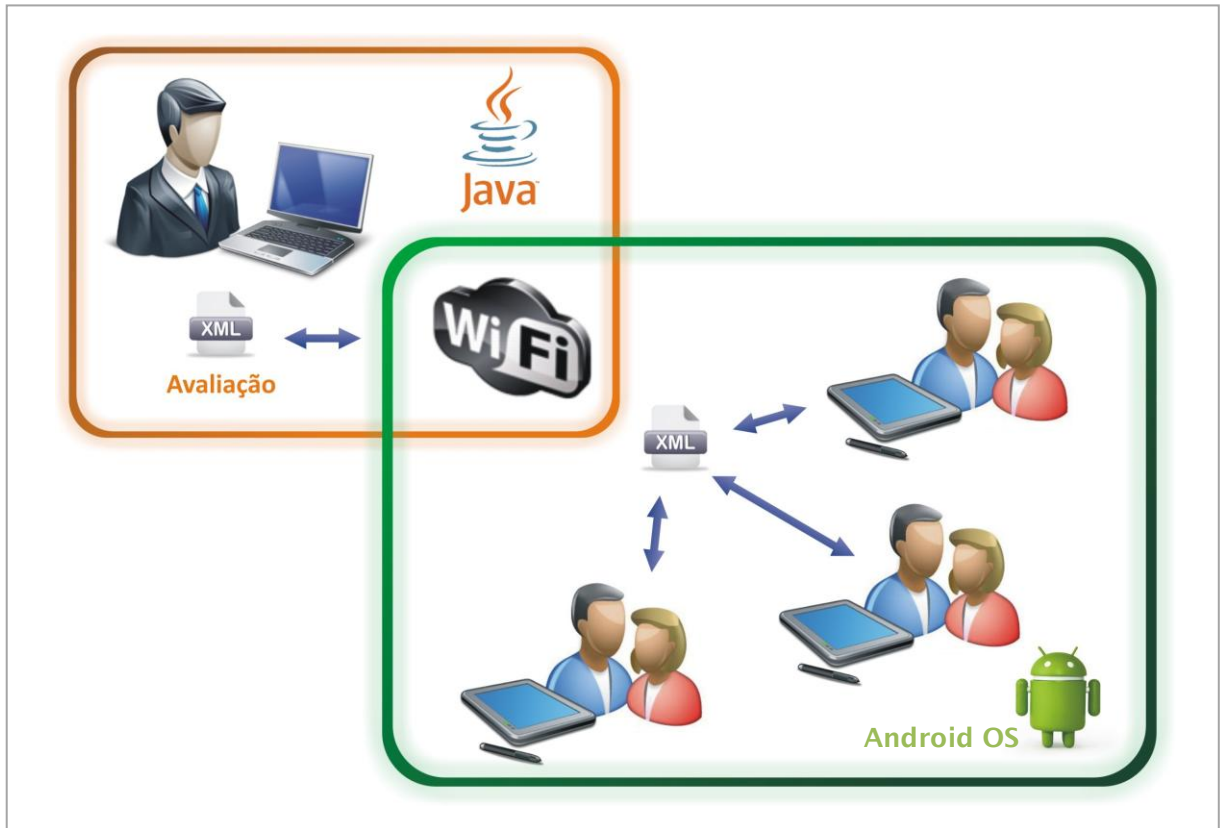
A primeira forma de avaliação gera uma problemática devido à necessidade de correção manual – embora exista a tecnologia para correção de questões objetivas baseada em scanners – já a segunda, oferece a vantagem da correção automática de questões objetivas, o que é uma boa solução à problemática da correção manual de avaliações, no entanto, requer uma dispendiosa infra-estrutura de informática.

Laboratórios de informática não são dispendiosos apenas por causa do custo dos computadores, manter uma estrutura de laboratório de informática, nos moldes atuais, implica em despesas com suporte de rede, software e hardware; além de uma despesa considerável com consumo de energia elétrica.

Portanto, o cerne deste trabalho está em explorar o uso de uma estrutura composta por um modelo diferente para o ambiente de aplicação de avaliações acadêmicas, este ambiente é basicamente composto por um computador e vários *tablets*. O computador funcionaria como uma espécie de servidor, e seu papel seria o de auxiliar o professor no gerenciamento da conexão dos dispositivos e publicação das avaliações, enquanto os *tablets*, fazendo o papel de clientes, são as ferramentas utilizadas pelos alunos para navegar, responder e enviar a avaliação de volta ao professor.

Como vemos na Figura 1, esta proposta foi pensada sobre uma infra-estrutura criada por conexões sem fio entre um computador executando uma aplicação Java Desktop e os *tablets* ou *smartphones* executando uma aplicação Android através de WiFi.

**Figura 1.** Visão geral das aplicações propostas por este trabalho.



Fonte: Própria.

Usar uma aplicação nativa de um hardware portátil, como é o caso de um *tablet* ou *smartphone* com Android, garante muitas vantagens em lugar de um sistema Web otimizado para estes aparelhos, algumas delas são: (a) controle dos integrantes de uma rede em nível de MAC (*Media Access Control*), (b) controle sobre o que o aluno está fazendo no hardware e (c) melhor aproveitamento dos recursos de hardware.

## 2. REFERENCIAL TEÓRICO

### 2.1. ENGENHARIA DE SOFTWARE

#### 2.1.1. Unified Modeling Language – UML

A Linguagem Unificada para Modelagem, conhecida como UML, acrônimo para o inglês *Unified Modeling Language*, foi a linguagem de símbolos e especificações utilizada como referência para a modelagem dos Softwares propostos neste trabalho, uma definição amplamente utilizada para UML é a seguinte:

“A UML é uma linguagem de modelagem visual multi-propósito que é usada para especificar, visualizar, construir e documentar artefatos de um sistema de software.” (RUMBAUGH; JACOBSON; BOOCH; 1999, p. 3).

Diz-se que a UML é uma linguagem de modelagem visual, porque a construção simbólica dos artefatos de software, ou, a modelagem, é representada fundamentalmente por diagramas, e estes divididos em dois grandes grupos: Diagramas Estruturais e Diagramas Comportamentais (RUMBAUGH; JACOBSON; BOOCH; 1999).

Os Diagramas Estruturais tem o papel de descrever atributos, arranjo e modelos de relacionamento dos artefatos estáticos do próprio software.

Os Diagramas Comportamentais descrevem aspectos dinâmicos dos artefatos do software, como interações do usuário, mudança de estados do sistema e etc. De forma que os modelos deste tipo de diagrama sempre representam uma seqüência de eventos e relacionam estes eventos com um comportamento.

É importante citar que embora a UML tenha, entre outros, o propósito de especificar dentro do processo de software, “não é um objetivo de a UML ser um método completo de desenvolvimento” (RUMBAUGH; JACOBSON; BOOCH; 1999, p. 8).

Além de especificar, os diagramas da UML podem ser utilizados para criar uma visualização de artefatos do sistema de software. Como exemplo, é possível citar o Diagrama de Classes e o Diagrama de Objetos, que são uma visualização que representam as entidades, atributos e métodos que podem ser integralmente exportados para a aplicação.

Construir e documentar são os outros propósitos da UML (RUMBAUGH; JACOBSON; BOOCH; 1999), embora construir artefatos de software seja possível através da UML, isso

não é uma realidade para todos os seus diagramas. Um exemplo de construção de artefatos de software utilizando UML é o dos Diagramas de Classes, já que é possível exportar classes criadas no JUDE, uma ferramenta CASE abordada no tópico seguinte, para arquivos de linguagens de programação como Java e C#.

### 2.1.2. Ferramentas CASE (Computer-Aided Software Engineering)

Ferramentas CASE estão presentes em todo o processo de software, desde a modelagem, passando pela implementação, debug e teste, isso porque ferramentas CASE podem atuar em dois níveis na engenharia de software: (a) Alto nível e (b) Baixo nível;

Ferramentas CASE de alto nível são essencialmente utilizadas para modelar, e um exemplo de ferramenta CASE de alto nível é JUDE, cuja sigla significa *Java UML Development Enviroment*, que traduzida significa: Ambiente Java de Desenvolvimento UML.

Já as ferramentas CASE de baixo nível estão mais dedicadas às tarefas de implementação de artefatos do sistema de software em uma linguagem de programação como Java ou C#. Alguns exemplos de ferramentas CASE de baixo nível são: Eclipse, Visual Studio e NetBeans.

Essa realidade multifacetada das ferramentas CASE concorda com o que vemos na definição da sigla CASE,

“O Acrônimo CASE está para [em inglês] Engenharia de Software Suportada por Computador. E cobre uma extensa gama de diferentes tipos de programas que são usados para suportar as atividades de processo de software tais como análise de requerimentos, modelagem de sistema, debug e teste.”  
(SOMMERVILLE; 2007, p.34)

As ferramentas CASE escolhidas como padrão para o desenvolvimento das aplicações propostas por este trabalho foram o (a) JUDE, para a modelagem e como ferramenta de suporte de alto nível e o (b) Eclipse, como ferramenta para o desenvolvimento das aplicações na linguagem Java.

O JUDE será utilizado no processo de modelagem dos softwares propostos por este trabalho, através da criação do Diagrama de Classes e Diagrama de Casos de uso. Uma informação importante é que recentemente esta ferramenta, amplamente conhecida como JUDE, mudou seu nome para Astah, assim como também sua logo.

O Eclipse é uma plataforma de desenvolvimento que permite a construção de aplicativos em Java, PHP, Ruby e outras linguagens.

O Eclipse foi escolhido por suportar tanto a construção do aplicativo Java desktop proposto, como também suporta a construção do aplicativo Android. A versão do Eclipse escolhida foi a mais recente, a versão 3.7 também chamada de Indigo.

Diz-se que o Eclipse é, mais precisamente, uma plataforma ao invés de um IDE, devido à possibilidade de qualquer um desenvolver para o próprio Eclipse. Por causa desta característica existem milhares de complementos, no formato de *plugins*, que tornam o Eclipse uma ferramenta completa e versátil, e, embora a maioria destes *plugins* seja gratuita, existe também um grande número de *plugins* pagos, ideais para usuários corporativos que desejam suporte e garantias.

Por muito tempo o Eclipse não dispunha de uma ferramenta para desenvolvimento visual de aplicações desktop tão estável e robusta quanto a de seu principal concorrente, o NetBeans, e esse foi um motivo pelo qual se cogitou utilizar o NetBeans, atualmente suportado pela Oracle, em lugar do Eclipse para o desenvolvimento da aplicação Java Desktop proposta por este trabalho.

Entretanto, antes de abandonar o Eclipse, recorreu-se ao Google para uma pesquisa sobre o que havia, em termos de ferramenta, para o desenvolvimento de aplicações Java Desktop no Eclipse, essa pesquisa resultou na descoberta de uma ferramenta gratuita e de código aberto, o WindowBuilder, que suporta o desenvolvimento das janelas de uma aplicação Java Desktop de forma visual através de *plugin* para o Eclipse.

Ao utilizar o WindowBuilder, durante a implementação Java Desktop proposta por este trabalho, ficou claro de que se trata de uma ferramenta muito estável. Podemos destacar como pontos positivos a flexibilidade para se trabalhar com diferentes bibliotecas visuais Java, SWT ou SWING, e a possibilidade de editar o código de geração dos componentes visuais do formulário, já que “a maioria dos construtores de GUI [para Java] só podem ler e escrever códigos que eles mesmos escreveram” (GOOGLE, 2011c), o WindowBuilder permite a edição do código visual criado através de uma *feature* chamada de *Bi-directional Code Generation*.

Para se informar mais sobre o WindowBuilder e ver detalhes sobre a instalação, configuração e uso desta ferramenta no ambiente Eclipse, que por enquanto é o único

suportado, visite o site do projeto <http://code.google.com/intl/pt-BR/javadevtools/wbpro/index.html>.

## 2.2. WIFI

As primeiras conexões sem fio remontam à época em que surgiam os rádios, e, embora se tratasse de uma conexão *simplex*, em que existe um emissor para vários receptores, e, cada receptor, não pode ser um emissor, foi com esta tecnologia que se começou a explorar os benefícios da ausência de fios para a realização da comunicação. A primeira transmissão de rádio no mundo ocorreu em 1906, já no Brasil, as primeiras ondas radiofônicas foram emitidas em sete de setembro de 1922 (FOLHA, 2002). Na verdade, esse é um fato importante, mas provavelmente só foi citado porque esta obra era escrita nos dias em que se falou do aniversário do mesmo, importante mesmo para este trabalho é que desde então, muita coisa mudou.

Atualmente, redes de dispositivos podem ser criadas a partir de uma infra-estrutura sem cabos, graças à tecnologia WIFI, esse acrônimo designa o termo em inglês *Wireless Fidelity*, ou seja, fidelidade sem fio. Segundo a própria WIFI Alliance, que é a organização que mantém as especificações do padrão WIFI, “Em 1999 muitos líderes visionários juntaram-se para formar uma organização global sem fins lucrativos e com o objetivo de conduzir a adoção de um único padrão global para redes locais sem fio de alta velocidade.” (WIFI ALLIANCE, 2011)

A tecnologia WIFI adota o padrão especificado pelo IEEE (Instituto de Engenheiros Elétricos e Eletrônicos), dos Estados Unidos, esse padrão foi estabelecido pela norma 802.11 deste instituto. Em 2010, a WIFI Alliance tinha 350 companhias membro.

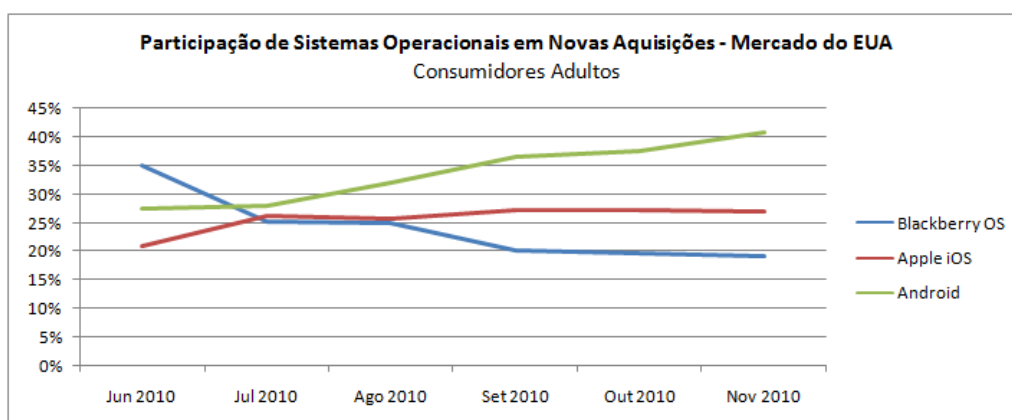
Depois desta breve abordagem panorâmica sobre o histórico da tecnologia WIFI, é preciso que se diga que não é objetivo deste trabalho explorar os pormenores da implementação e especificações da tecnologia WIFI, estas questões são transparentes para as aplicações em foco, já que as mesmas fazem uso do WIFI apenas como meio de conexão entre dispositivos através dos protocolos TCP/IP na camada de transporte e rede.

### 2.3. ANDROID

Segundo o site oficial para desenvolvedores Android, o Android “é uma pilha de software para dispositivos móveis que inclui um sistema operacional, middleware e aplicações chave” (GOOGLE, 2011a).

O Android é uma aposta do Google e se trata, como foi dito, de um sistema operacional para dispositivos móveis. O surgimento do sistema operacional Android tem tornado o mercado de *smartphones* extremamente agitado. Em 2010, o número de aparelhos com o Android embarcado superou o número de iPhones (smartphones da Apple), que embarcam o IOS, entre as novas aquisições feitas por adultos, como é possível visualizar no gráfico da figura 2.

**Figura 2.** Gráfico com a participação dos OS para dispositivos móveis.



Fonte: ANGELIM, 2011.

Outra notícia positiva para o Android OS foi divulgada recentemente pelo instituto de pesquisa IDC (apud. MAX, 2011), e dá conta de que até 2015 o sistema operacional para dispositivos móveis do Google irá ter conquistado 49% do mercado, seguido pela solução desenvolvida pela Microsoft e Nokia, que ocuparia 20,9% do mercado de smartphones, em terceiro ficaria o sistema operacional da Apple, o IOS, com 15,3% de participação.

De acordo com ROGERS *et. al.* (2009, p. 19), “o Android, a plataforma de software [SO] para dispositivos móveis do Google, pode ser a próxima grande oportunidade para os desenvolvedores de aplicação”.

Feitas essas observações, diga-se que demonstram a razão pela qual foi feita a escolha do Android como plataforma para o desenvolvimento da aplicação para dispositivos móveis proposta, em lugar do velho conhecido J2ME, que seria uma escolha mais óbvia, em função

de haver sido abordado no decorrer deste curso de especialização. No momento, o Android parece ser uma tendência bem mais promissora, além disso, embora suas aplicações precisem seguir especificações muito particulares, podem ser escritas em Java.

### 2.3.1. Componentes de um Aplicativo Android

Justificada a escolha, iremos abordar alguns conceitos técnicos fundamentais de uma aplicação Android, segundo ROGERS (*et. al.*, 2009), os seus principais componentes são: (a) *Activities*, (b) *Services*, (c) *Intents* e *Broadcast Receivers* e (d) *Content Providers*.

Vejamos uma definição breve de cada um desses componentes:

- a) *Activities*: Este componente pode se comparar a uma parte do aplicativo executável. São fragmentos de código que podem ir e vir durante a execução de uma aplicação; Cada *activity* normalmente corresponde a uma tela de interação com o usuário;
- b) *Services*: Os serviços são a implementação de uma tarefa que precisa existir sem que haja uma interface de interação com o usuário. São semelhantes aos processos que existem em um sistema operacional desktop, como Windows, por exemplo;
- c) *Intent* e *Broadcast Receivers*: Estes componentes existem para permitir a interação com outra aplicação, por meio de uma resposta a uma requisição vinda de outra aplicação. *Broadcast Receivers* ouvem a um anúncio de evento feito a todo o sistema – *system-wide* –, esses anúncios podem partir do sistema ou de outra aplicação. *Intent Receivers* ou *Intents* são componentes utilizados para interagir com outras aplicações ou serviços invocando-os;
- d) *Content Providers*: São criados para compartilhar dados com outros *activities* ou *services*. Funcionam quando uma aplicação solicita dados de um Contato, ela deve apontar para um *content provider* através da URI “content://contacts/people”, e essa requisição será atendida pela aplicação registrada para isso. Se houver mais de uma, o sistema irá precisar que se escolha uma.

Outros detalhes de sintaxe e outros aspectos do uso dos componentes de um aplicativo Android *Activity* e *Service* poderão ser vistos no tópico Desenvolvimento, presente neste trabalho. Entretanto, neste tópico ainda serão abordados mais detalhes dos componentes *activities* e *services*, através da abordagem de seus respectivos ciclos de vida, cujo entendimento é fundamental para o desenvolvimento de aplicações Android.

### 2.3.2. O Ciclo de Vida de uma *Activity*

Uma forma interessante de se visualizar uma *activity* é pensá-la como um Caso de Uso, aliás, o nome deste componente é bem sugestivo, sua tradução livre é “atividade”. Pois bem,

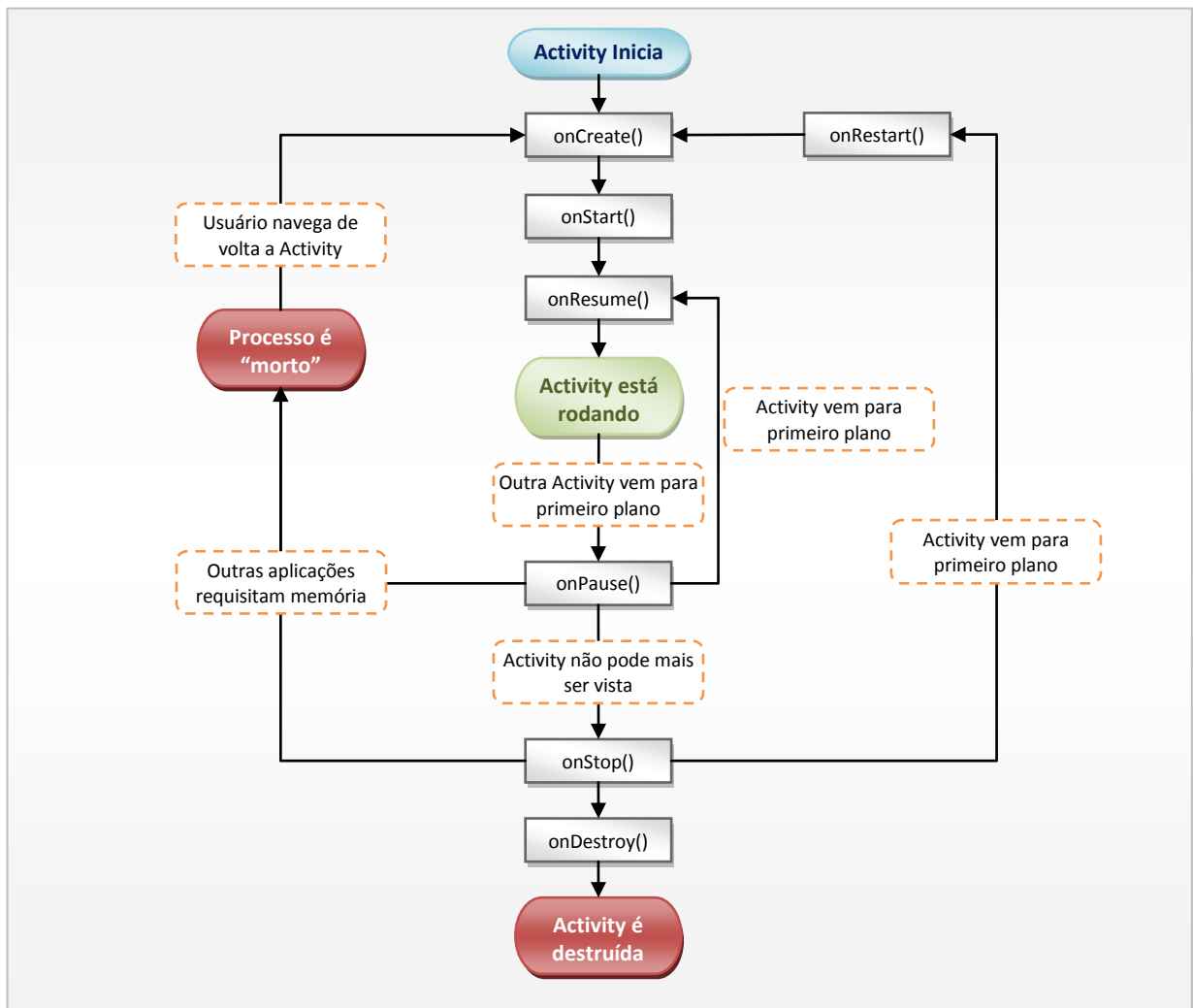
*activities* estão estreitamente relacionadas com uma interface de usuário e uma tarefa bem específica dentro da aplicação Android.

É melhor ter várias *activities*, com cada uma fazendo pouca coisa, do que poucas *activities* fazendo muita coisa. Isso ocorre por causa da limitação de recursos em dispositivos móveis, tanto a memória quanto bateria são decisivos para se pensar dessa forma. É importante notar que é o Android quem decide quando uma *activity* está pronta para ser destruída, e isso ocorrerá enquanto uma *activity* não estiver em *foreground* – primeiro plano – e o sistema requisitar espaço de memória.

A bateria também é um fator importante para se entender esta lógica porque a montagem de um aplicativo na memória, para seja executado, tem um custo de processamento, e processamento significa consumo de bateria, segundo ROGERS et. al. (p. 21, 2009), “A limitação fundamental de um dispositivo móvel é a capacidade da bateria. Cada *clock tick* do processador, cada atualização da memória, cada pixel na tela do usuário consome energia da bateria. O tamanho das baterias é limitado, e usuários não gostam de cargas freqüentes de bateria.”

Sendo assim, o gerenciamento eficiente de *activities* e *services* passa pelo conhecimento de seus ciclos de vida, e isso para que nenhuma *activity* ou *service* permaneça consumindo recursos a menos que isso seja realmente necessário. Na figura 3, podemos ver o ciclo de vida de uma *activity*.

**Figura 3.** Ciclo de vida de uma *Activity*.



Fonte: Google Inc., 2011b

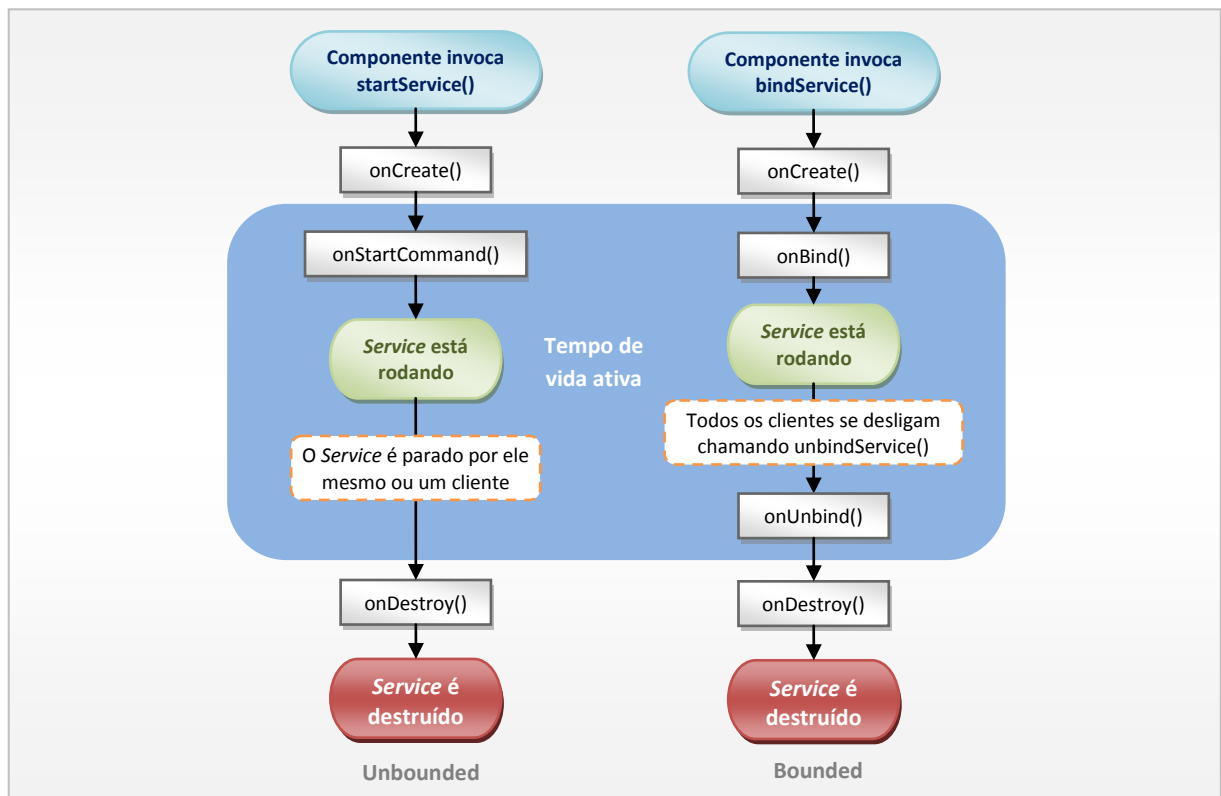
### 2.3.3. O Ciclo de Vida de um *Service*

Em algumas ocasiões é preciso executar tarefas mesmo sem a interação do usuário, para que isso ocorra em segundo plano é preciso fazer uso dos *services*. De forma análoga a que ocorre em um computador Desktop, serviços são executados em segundo plano e geralmente não necessitam de interação direta com o usuário, mas executam tarefas de apoio à aplicação.

De acordo com ROGERS *et. al.* (2009, 23), “[uma] aplicação pode precisar da implementação de *services* para realizar tarefas de segundo plano que persistam sem uma interface de usuário”.

Assim como *activities*, ao implementar *services*, também é preciso conhecer seu ciclo de vida. Na figura 4 está ilustrado o ciclo de vida de *services*, vinculados e não vinculados.

**Figura 4.** Ciclo de vida de *services* vinculados e não vinculados.



Fonte: Google Inc., 2011b.

Serviços vinculados, ou *bound services*, permitem que outros componentes de uma aplicação Android interajam com ele por meio da Comunicação Inter-processo (*Interprocess Communication – IPC*).

Para que essa interação possa ocorrer, o *bound service* só precisa implementar o método `onBind()`, este método deve retornar um objeto 'IBinder' que define a interface de comunicação com o *service* quando este for chamado por um componente de aplicativo Android – uma *activity*, por exemplo – através do método `bindService()`.

Um único *bound service* pode atender múltiplos clientes ao mesmo tempo, quando um cliente conclui sua interação com o *bound service*, é necessário invocar o método `unbindService()`. Quando não houver nenhum cliente vinculado ao *bound service*, o sistema irá destruir o serviço.

Os serviços não vinculados são chamados através do método `startService()`, e não podem interagir com o componente que o chamou, é importante que ao concluir sua tarefa o *service* seja finalizado através do uso do método `stopSelf()` ou `stopService()`. O que, como

já foi abordado, não precisa ocorrer com os serviços vinculados, que são destruídos pelo sistema.

#### 2.3.4. O Arquivo AndroidManifest (.xml)

Um último aspecto a respeito da plataforma Android, a ser abordado nesta visão panorâmica, é um arquivo em XML que é chave no desenvolvimento de aplicações Android e se chama: AndroidManifest.

O AndroidManifest é um arquivo de configuração sem o qual sua aplicação não será compilada. Entre seus propósitos está o de declarar e configurar *activities*, *services*, *providers* e *receivers*. Na figura 5 vemos um exemplo do AndroidManifest (arquivo xml) padrão criado pelo Eclipse para um novo projeto Android.

**Figura 5.** Exemplo do arquivo AndroidManifest (XML)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.saara.aplicacao"
    android:versionCode="1"
    android:versionName="1.0">

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

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

        <activity android:name=".inicio"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Fonte: Própria.

Além desse propósito de declarar componentes da aplicação, outro aspecto importante é o de requisitar permissão para utilizar recursos do dispositivo.

Sabemos que a aplicação Android proposta neste trabalho precisa realizar uma conexão WIFI para obter os dados de outra aplicação via rede, para isso é preciso declarar no arquivo AndroidManifest a necessidade de usar esse recurso no dispositivo.

Observe os nós "*uses-permission*" na figura 5, neles podemos perceber declaradas duas permissões: CHANGE\_WIFI\_STATE e INTERNET.

A primeira permissão, `CHANGE_WIFI_STATE`, existe para informar ao usuário do dispositivo a intenção da aplicação de alterar o estado do WIFI, ligar ou desligar, conforme a conveniência.

A segunda permissão, `INTERNET`, informa que a aplicação precisa criar *sockets* de rede, e, obviamente, de que precisará enviá-los e recebê-los por uma conexão com a internet (GOOGLE, 2011b).

O usuário da aplicação Android que tenha declarado essas permissões irá ser notificado de que elas são requeridas no momento em que estiver realizando a instalação do software em seu dispositivo.

## 2.4. JAVA

Ao falar de Java, de acordo com Flanagan (1999, p. 20), “precisamos entender e separar bem três conceitos fundamentais: (a) a linguagem Java, (b) a Máquina Virtual Java e a (c) Plataforma Java”.

A linguagem Java é linguagem de programação, orientada a objetos, com a qual é possível codificar aplicações Java, ao compilarmos o código a aplicação é criada em Byte Codes.

A Máquina Virtual Java (ou *Java Virtual Machine* – JVM) é responsável pela interpretação dos Bytes Codes Java e da tradução em linguagem de máquina compreensível pelo Sistema Operacional Host.

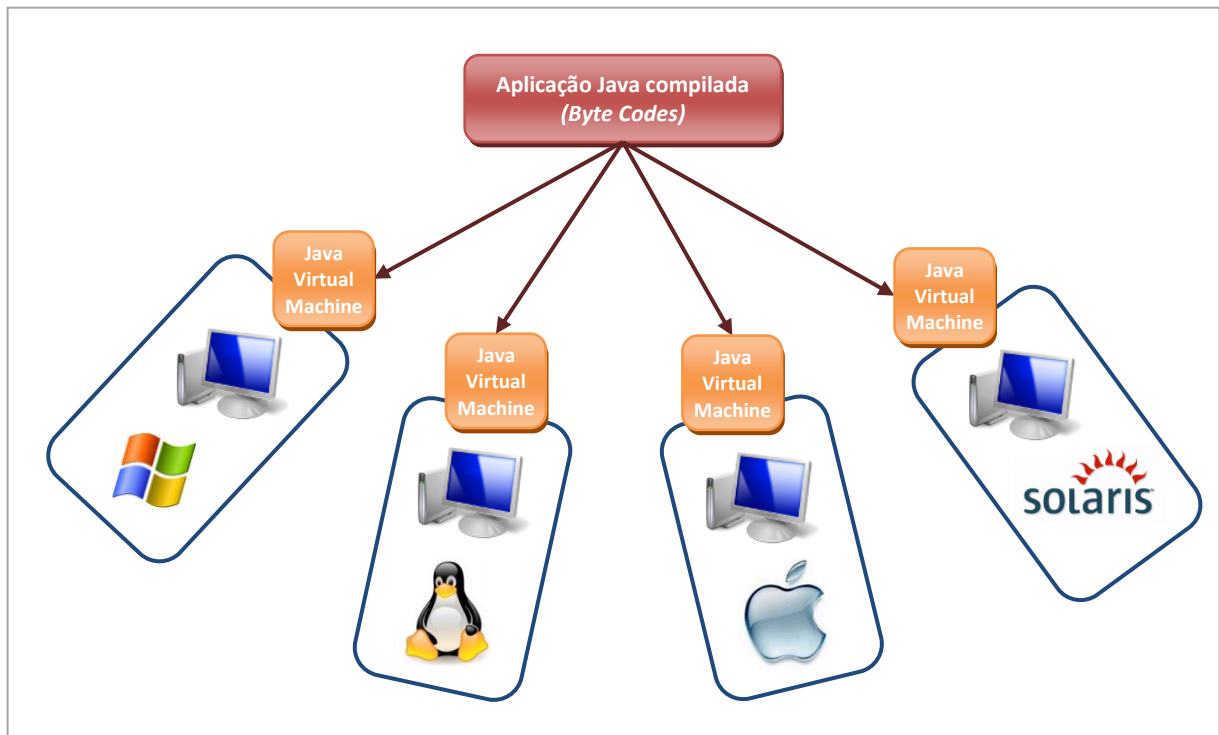
Por último, a plataforma Java é o conjunto das classes oferecidas ao desenvolvedor Java, que também são conhecidas como *Core Java APIs*. Além das classes nativas, a plataforma Java é extensível e novas classes podem ser incorporadas à plataforma.

A idéia por trás da concepção do Java é a portabilidade, ou seja, criar uma maneira de fazer com que aplicações pudessem rodar em diferentes sistemas operacionais, mesmo com o mesmo código e uma única compilação, um slogan comumente ouvido é “compile uma vez, execute em qualquer lugar” (FLANAGAN, 1999).

Na prática, essa idéia está restrita aos sistemas operacionais para os quais existe uma JVM disponível, porque o segredo da portabilidade do Java está nas JVMs, hoje, a Oracle mantém oficialmente JVMs para o Windows, Solaris, Linux, e, no caso do OSX, a máquina

virtual é fornecida pela própria Apple. Como exemplo do que fez a Apple, terceiros podem desenvolver sua própria máquina virtual para interpretar *byte codes* de aplicações Java para a linguagem de máquina nativa do sistema operacional, conforme é possível observar na figura 6.

**Figura 6.** Esquema da infra-estrutura que proporciona a portabilidade do Java.



Fonte: Baseado em FLANAGAN, 1999.

## 2.5. EXTENSIBLE MARKUP LANGUAGE - XML

Segundo Daum e Merten (2003), a linguagem XML, um padrão aberto, surgiu para ser um padrão que permitisse a troca de dados de forma estruturada e independente de plataforma ou linguagem de programação.

O XML pode se comparar a padrões abertos como HTML, SMTP, HTTP, POP3 e etc. Estes protocolos tornaram a internet possível, e são comparáveis ao XML porque todos se tratam de 'textos de processamento' baseados em um padrão aceito universalmente.

Caso protocolos não fossem aceitos por todas as companhias, e cada organização tivesse seu protocolo particular, a internet seria inviabilizada. Redes privadas Windows não se conectariam com redes desenvolvidas pela IBM ou Apple. Ao invés de uma rede universal teríamos milhares de bolhas.

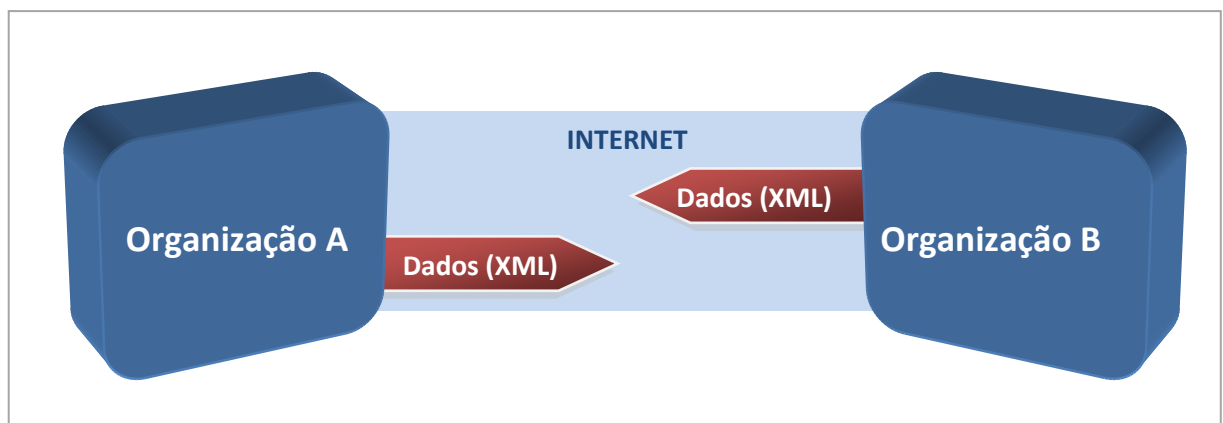
O XML só existe graças à internet, e, ainda segundo Daum e Merten,

“As raízes de sua linguagem estão baseadas em tecnologias de processamento de documentos consolidadas e sua tecnologia move-se em direção à área de banco de dados e TI organizacional, sua principal aplicação é estabelecer comunicação entre partes cooperantes na internet, extranet ou intranet”.  
(2003, p. 18)

Um fato importante é que o surgimento do XML foi tardio, se comparado com outros protocolos já citados, porque a necessidade de troca de dados, em forma similar a como são guardados em bancos de dados convencionais, só surgiu alguns anos depois do florescimento da rede mundial, no final dos anos 90 (DAUM & MERTEN, 2003).

É preciso lembrar que nos primórdios da internet apenas as páginas ‘estáticas’ em HTML existiam, as organizações ainda não confiavam na internet o bastante para usá-la como ferramenta para compartilhar dados. Quando a demanda por troca de dados entre empresas através da internet surgiu, a linguagem XML se estabeleceu, de acordo com Daum e Merten (2003), como a forma padrão adotada para fazer isso, podemos ver na figura 7, uma ilustração de como a linguagem XML conjuga o uso da internet na relação das organizações e os dados. Com o uso de XML, tanto faz a plataforma ou linguagem de programação que as empresas A ou B utilizam, elas poderão trocar dados sem problemas.

**Figura 7.** A integração promovida pela linguagem XML.



Fonte: Baseado em DAUM & MERTEN, 2003.

A linguagem XML consiste em *tags* que compõe uma árvore, com nós pais e filhos, de um modo geral, estes nós delimitam uma representação de ‘tabelas’, ‘campos’ e ‘registros’ criando um documento com aspecto similar ao que é mostrado na figura 8.

**Figura 8.** Exemplo de um arquivo em linguagem XML.

```
<?xml version="1.0"?>
<boletim data="10/07/2011">
<aluno>
  <nome>DIEGO MORAIS DE MEDEIROS</nome>
  <notas>
    <portugues>7</portugues>
    <matematica>8</matematica>
    <ciencias>6</ciencias>
  </notas>
</aluno>
<aluno>
  <nome>THOMAS JEFFERSON</nome>
  <notas>
    <portugues>5</portugues>
    <matematica>8</matematica>
    <ciencias>7</ciencias>
  </notas>
</aluno>
</boletim>
```

Fonte: Própria.

## 2.6. SEGURANÇA

A segurança no contexto da informática pode ter um sentido muito amplo e abrangente, segundo OAKS (2001), segurança é um termo que pode traduzir (a) integridade na comunicação de dados, (b) confiabilidade de hardware, (c) inviolabilidade de informações, (d) integridade no armazenamento de dados e etc.

No contexto da aplicação proposta, a segurança desejada é a de inviolabilidade de informações, ou seja, evitar que dados não sejam interceptados, e mesmo se forem, que eles não possam ser compreendidos.

Uma maneira de se alcançar este objetivo é a criptografia, ainda segundo OAKS (2001), a criptografia consiste na técnica decompor, cifrar e desordenar caracteres com base em um padrão algorítmico estabelecido por uma chave, de forma que, através desta chave, um receptor genuíno esteja apto a reordenar, recompor e decifrar os caracteres enviados de maneira adequada.

Tanto uma aplicação Android como Java Desktop podem realizar a criptografia de dados, bem como a descriptografia, fazendo uso do pacote '*javax.crypto*', e isso de forma síncrona ou assíncrona.

## 2.7. SQLITE

Em termos simples, SQLite é um Sistema Gerenciador de Banco de Dados Relacional, SGBD, gratuito e de domínio público, cuja arquitetura está baseada em um modelo de

simplicidade, baixo-consumo e ‘leveza’, no entanto, sem abrir mão de capacidade, confiabilidade e integridade (KREIBICH, 2010).

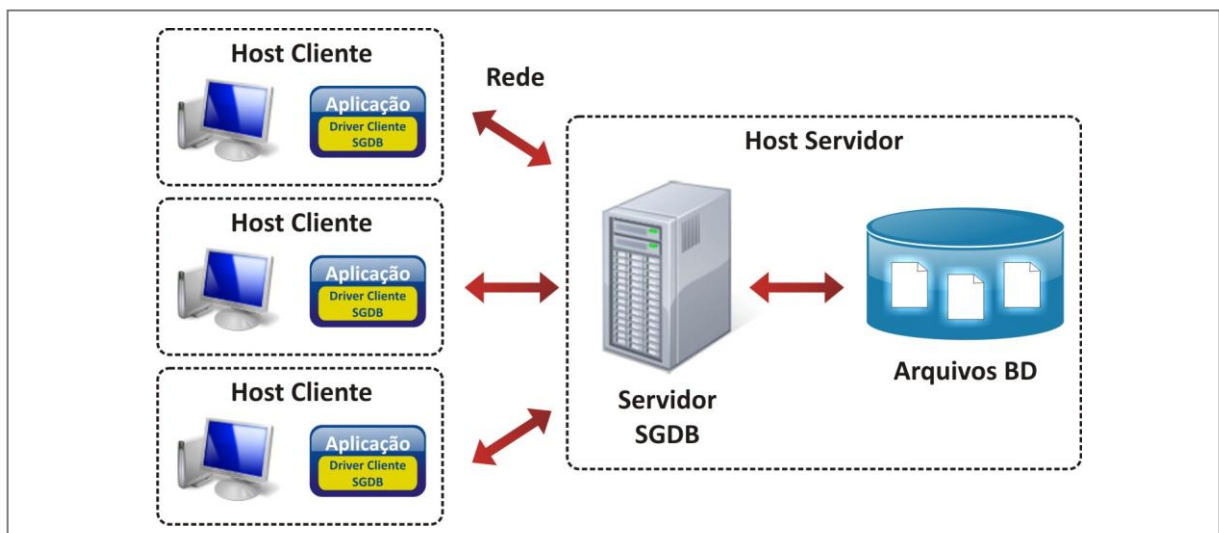
No que diz respeito ao SQLite, e, ainda segundo Kreibich (2010, p. 1), “O Lite [variação do inglês para light: leve, frágil] não faz referência às suas capacidades. Ao invés disso, SQLite é leve quando se trata de complexidade de setup, *overhead* administrativo e uso de recursos.”

Diferente de outros produtos de banco de dados relacionais, tais como Oracle, DB2 e SQL Server, o SQLite não possui uma arquitetura cliente/servidor. O SQLite não possui um servidor separado para gerenciar a conexão de múltiplos clientes sobre uma rede, arquitetura na qual aplicações que desejam acessar um banco precisam de um *driver* específico para isso.

Na verdade, apesar de suportar acesso concorrente ao Banco de Dados, o SQLite não foi pensado sobre uma arquitetura cliente/servidor, as diferenças começam pelo modo como as aplicações se comunicam com os dados, para que isso ocorra, o *engine* do SQLite deverá estar embutido em qualquer que seja a aplicação que queira fazê-lo. Já que não existe servidor, para gerenciar o acesso com integridade aos dados em diferentes arquivos, no SQLite os dados de cada Banco ficam em um único arquivo.

Nas figuras 9 e 10, vemos diagramas com a arquitetura de um SGBD tradicional, voltado para o atendimento de requisição de vários computadores através de um servidor separado, e o SQLite, como um SGBD em uma arquitetura sem servidor, respectivamente.

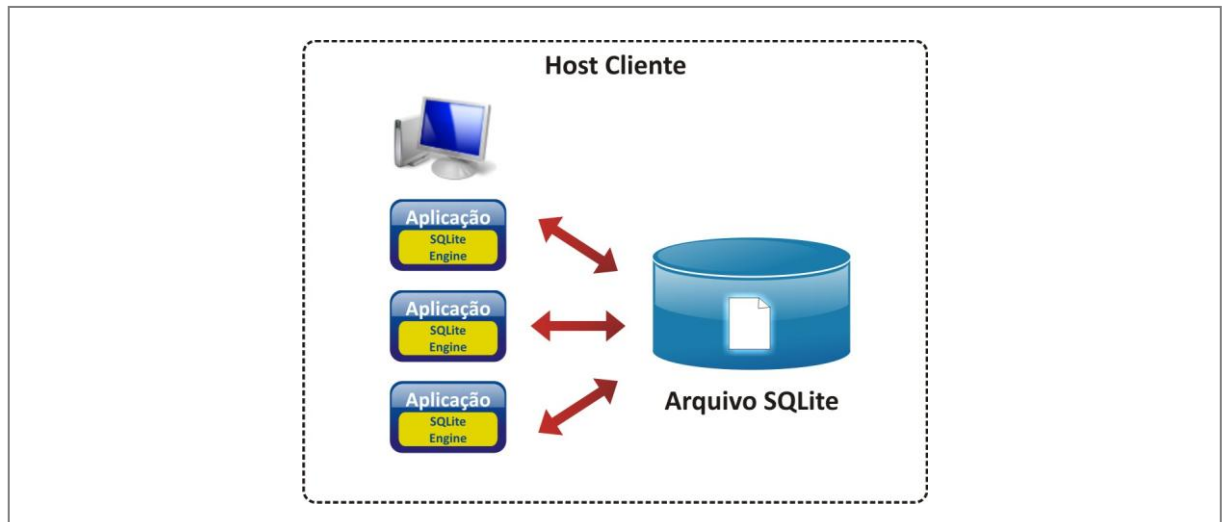
**Figura 9.** Arquitetura cliente/servidor de um SGBD tradicional.



Fonte: Baseado em KREIBICH, 2010.

Na arquitetura cliente/servidor, fica evidente o quão intrusivo um SGBD costuma ser, e como o consumo recursos pode ser elevado. Nesse modelo, é muito comum, até como hábito de boa prática, ter um computador *host* específico para basear o ‘Servidor de Banco de Dados’ para aplicações de médio e grande volume de acessos.

**Figura 10.** Arquitetura SQLite de conexão a dados.



Fonte: Baseado em KREIBICH, 2010.

Para as aplicações propostas por este trabalho, uma Java Desktop e outra Android, o ambiente de execução será diferente. Como já foi discutido, o ambiente Android é desenvolvido para dispositivos móveis, nos quais a limitação de recursos é um aspecto fundamental para se considerar no desenvolvimento de aplicações.

Tanto a aplicação Desktop como a aplicação Android, propostas, terão como requisito funcional a persistência de dados. Escolher um banco de dados relacional para uma aplicação Java Desktop é uma tarefa relativamente fácil, já que existe um driver JDBC específico para SGBDs como MySQL, SQL Server, Oracle e etc.

Entretanto, para aplicações Android, esta questão é muito mais restritiva, de modo que a única opção, suportada nativamente pelo ambiente, é o SQLite.

É fundamental citar que não é correto comparar o SQLite com um SGBD tradicional, já que o consumo de recursos são absurdamente diferentes, enquanto a execução do *engine* do SQLite consome, em média, 240KB de memória, a soma do consumo de memória de um SGBD tradicional pode facilmente chegar a algumas dezenas ou centenas de megabytes

(KREIBICH, 2010). Além disso, a arquitetura do SQLite deixa claro que sua intenção não é competir com um SGBD tradicional, mas preencher um nicho específico, especialmente de dispositivos móveis e aplicações *standalone*.

#### 2.7.1. *Features* que o SQLite possui

Embora existam algumas limitações, o SQLite tem, além de uma arquitetura leve, um conjunto de *features* que justificam sua escolha por grandes companhias para ser o SGBD de uma grande variedade de SOs para dispositivos móveis, além do Android, objeto de exploração deste trabalho, o IOS da Apple e o Symbian da Nokia, também embarcam esta tecnologia.

Vejamos as principais *features*, que diferenciam o SQLite dos principais nomes para SGBD de arquitetura tradicional, segundo ALLEN e OWENS (2010):

**Configuração Zero:** Fazer a conexão com os dados de uma base SQLite não requer a configuração avançada de um servidor, o *engine* do SQLite só precisa saber onde o arquivo com os dados está, e ter acesso a ele.

**Portabilidade:** O SQLite foi concebido sobre a idéia de portabilidade. Um banco de dados do SQLite pode ter sido criado no ambiente Windows, no entanto, ele poderá ser carregado e lido no Linux, Mac OSX, iPhone, Android e etc. Assim como, o SQLite pode ser compilado em qualquer desses ambientes indiferentemente.

**Compacto:** Tudo está contido em um único arquivo – tabelas, gatilhos, schemas, índices e visualizações – além disso, dados só ocuparão o espaço mínimo requerido, por exemplo, um dado num campo varchar(100) que utilize 2 bytes ocupará 3 bytes, e não os 100 bytes reservados, o byte extra armazena o tipo do dado.

**Gratuito:** O código do SQLite está sob domínio público e seu uso é livre e totalmente gratuito, ou seja, não existe nenhuma restrição legal ao uso, incorporação, alteração, distribuição, venda ou qualquer outra operação, comercial ou não, feita com o uso do SQLite.

**Confiabilidade:** Embora se trate de um software gratuito, o SQLite é severamente testado, segundo KREIBICH (2010, p. 6), “o teste padrão do SQLite consiste em mais de

10 milhões de testes de unidade e queries. O teste de carga, feito antes de cada nova versão, consiste em mais de 2,5 bilhões de testes”.

**Transacional:** O SQLite é um SGBD transacional, isso porque atende aos princípios de Atomicidade, Consistência, Isolamento e Durabilidade – ACID. Contudo, o nível de bloqueio do SQLite é de arquivo, e não de registro, como em um SGBD tradicional. Isso significa que uma alteração no arquivo de dados, precisa aguardar o fim de uma alteração iniciada por outra requisição, mesmo que as requisições afetem tabelas diferentes.

#### 2.7.2. *Features* que o SQLite não possui

Serão abordadas agora algumas características que não estão presentes no SQLite, *features* que foram deixadas de lado, essas características ausentes representam uma escolha deliberada em função da arquitetura escolhida e dos objetivos do SQLite (KREIBICH, 2010).

Allen e Owens (2010) listam também as características comuns aos SGBDs tradicionais que estão ausentes no SQLite, são elas:

**Concorrência:** É o gerenciamento das múltiplas requisições de acesso aos dados de uma base, seja para ler ou para escrever. O SQLite é transacional, isso significa que uma requisição de acesso aos dados não pode afetar uma que esteja em andamento, para isso o *engine* sabe esperar o fim de uma transação anterior antes de acessar o arquivo dos dados, evitando que seja corrompido. A grande limitação do SQLite é que ao acessar os dados, em qualquer operação que não seja um *SELECT*, todo o arquivo fica bloqueado, e, enquanto outros SGBDs são capazes de isolar um único registro para uma transação, o SQLite precisa isolar o banco dados inteiro. Essa limitação torna o SQLite capaz de processar somente algumas centenas de transações por segundo quando se tratam de alterações nos dados.

**Redes:** Embora os arquivos com os dados de uma base SQLite possam ser compartilhados, os métodos de *lock* de arquivo compartilhados em uma rede não atendem aos requisitos do SQLite, de forma que a equipe de desenvolvimento do SQLite não certifica ou recomenda seu uso dessa maneira.

**Controle de Acesso:** Já sabemos que os dados de um banco SQLite ficam em um arquivo, essa característica garante acesso aos dados por qualquer pessoa que possa ter acesso ao arquivo, ou seja, não é preciso ter usuário e senha do SGBD para acessar os dados.

**Engine do Banco:** Entre as operações que não são suportadas pelo SQLite estão: (a) nem todas as instruções de ALTER TABLE são suportadas, (b) Views são somente leitura, (c) algumas instruções de gatilho não são suportadas, (c) as instruções RIGHT e FULL OUTER JOIN não são suportadas e (d) NULL pode ser problemático em cláusulas de um WHERE, por experiência própria, esqueça que este ‘tipo de dados’ existe em qualquer rotina SQL contra um banco de dados SQLite. Nos casos em que especificamente precise verificar se um campo é nulo, carregue os dados para tipos primitivos da plataforma com a qual estiver trabalhando então percorra-os para verificar.

### 2.7.3. Quando o SQLite não é uma boa escolha

Por último, é importante citar motivos que indicam quando o SQLite pode não ser a melhor escolha, segundo KREIBICH (2010), é preciso avaliar com base nos benefícios e limitações para decidir se o SQLite é a melhor escolha para um determinado cenário, perceba que esta abordagem é diferente de uma mera comparação com outros Bancos de Dados, isso porque, de fato, eles normalmente irão oferecer mais recursos (e também consumir mais recursos do ambiente), a pergunta que deve-se fazer é: todos esses recursos de um SGBD tradicional são realmente necessários?

O SQLite não será uma boa escolha quando: (a) um grande volume de transações for requerido, “o SQLite fará apenas algumas poucas dúzias de transações por segundo” (SQLITE ORG., 2011), (b) um grande volume de dados for requerido, mais de 10 gigabytes por exemplo, (os dados estarão em um único arquivo, e alguns sistemas de arquivos não lidam bem com arquivos tão grandes) (c) quando as informações do banco de dados forem sensíveis, já que não existe controle de acesso, e, (d) quando a redundância dos dados for um requisito.

### **3. OBJETIVOS**

#### **3.1. OBJETIVO GERAL**

Implementar um suíte de softwares em múltipla plataforma, Java Desktop e Android, capazes de comunicar-se entre si, de modo bi-direcional, sobre uma arquitetura baseada em uma rede computacional sem fio, de forma que sejam capazes de suportar a realização de avaliações acadêmicas.

#### **3.2. OBJETIVOS ESPECÍFICOS**

- Modelar e implementar um aplicativo Java Desktop para criar, editar e enviar avaliações acadêmicas em formato XML para um dispositivo Android sobre uma conexão WIFI com persistência dos dados em SQLite;
- Modelar e implementar um aplicativo Android para receber, navegar, responder e enviar avaliações acadêmicas em formato XML para uma aplicação Java Desktop sobre uma conexão WIFI com persistência dos dados em SQLite;
- Fazer uso de técnicas de comunicação de dados com segurança através de criptografia em aplicações Java Desktop e Android;

#### 4. DESENVOLVIMENTO

Este tópico aborda as etapas da implementação das aplicações propostas por este trabalho, como citado anteriormente, uma aplicação Java Desktop e outra para a plataforma Android de dispositivos móveis.

##### 4.1. APRESENTAÇÃO DAS APLICAÇÕES E CONCEITOS

Antes de entrarmos nos aspectos mais técnicos da modelagem e a conseqüente implementação das aplicações propostas, é bom que se fale um pouco sobre elas, a partir desta breve apresentação com nome, ícone e conceitos fundamentais, os principais artefatos deste trabalho ganham uma identidade.

Os aplicativos Desktop e Android terão um nome comum: SAARA. Esse nome é um acrônimo das palavras Sistema de Avaliação Acadêmica por Rádio. O ícone das aplicações também é comum, e é ilustrado pela imagem de um sol. Esse ícone representa a idéia de irradiação, presente nas aplicações por meio da rede WiFi da qual necessitam para que possam ser executadas. A figura 11 mostra a logo com o ícone criado para as aplicações.

Figura 11. Logo e ícone das aplicações desenvolvidas.



Fonte: Própria.

Como já foi apresentado, e está também ilustrado na Introdução deste trabalho, o conceito fundamental das aplicações desenvolvidas e do foco desta obra estão baseados no uso de uma rede WiFi e protocolos próprios que promovam a comunicação entre um

aplicativo Desktop, no papel de servidor, e aplicativos Android, no papel de clientes, de forma que seja possível a realização de avaliações acadêmicas.

Outro conceito importante é o de criação de um ambiente de rede *Ad Hoc*, ou seja, do latim, “para este fim”, já que não se trata da disponibilidade passiva dos dados para acesso dos clientes da rede, como observa-se, por exemplo, em um *web service*, mas da distribuição seletiva desses dados para os dispositivos escolhidos.

## 4.2. MODELAGEM DAS APLICAÇÕES

Nos sub-tópicos seguintes serão abordadas as etapas de modelagem das aplicações propostas. Estas etapas não seguiram especificações de nenhum padrão de projeto em particular, mas obedecem a princípios gerais da normalização do desenvolvimento de software ao passar pelas etapas de (a) análise de requisitos, (b) modelagem e (c) implementação.

É bom que se esclareça a ausência de um tópico abordando qualquer metodologia de teste – fundamental na construção de software – como devida ao fato de se tratar de aplicações de cunho acadêmico em que o propósito é muito mais ilustrativo exploratório do que propriamente funcional. Os softwares foram testados de maneira unitária até o ponto de cobrirem minimamente os requisitos funcionais propostos.

### 4.2.1. Especificação de Requisitos do Saara Desktop

Os Requisitos Funcionais do software Desktop que devem estar presentes para o propósito de tornar viável a realização de avaliações acadêmicas sobre conexões WiFi com dispositivos móveis embarcando o Sistema Operacional Android, são os seguintes:

**RF1 Criação e exclusão de Registros de Avaliações:** Deve ser possível criar, editar e excluir de forma persistente, em banco de dados, registros de avaliações acadêmicas, compostas por questões e alternativas;

**RF2 Criação, edição e exclusão de Registros de Alunos:** Deve ser possível criar, editar e excluir de forma persistente, em banco de dados, registros de alunos, assim como o de dispositivos relacionados;

**RF3 Gerenciamento da conexão de dispositivos:** Deve ser possível gerenciar que dispositivos estão conectados e que dispositivos estão aptos a receber as avaliações;

**RF4 Envio de avaliações por WiFi a dispositivos Android:** Deve ser possível enviar avaliações de maneira seletiva com suas respectivas questões e alternativas a dispositivos móveis, com Android como sistema operacional, e por meio de conexão de rede WiFi. Os dados devem ser enviados atendendo às especificações do padrão XML, e devem ser criptografados.

**RF5 Recebimento das respostas por WiFi de dispositivos Android:** Deve ser possível receber, tratar e persistir, em banco de dados, as respostas enviadas pelos dispositivos móveis através de conexão de rede WiFi.

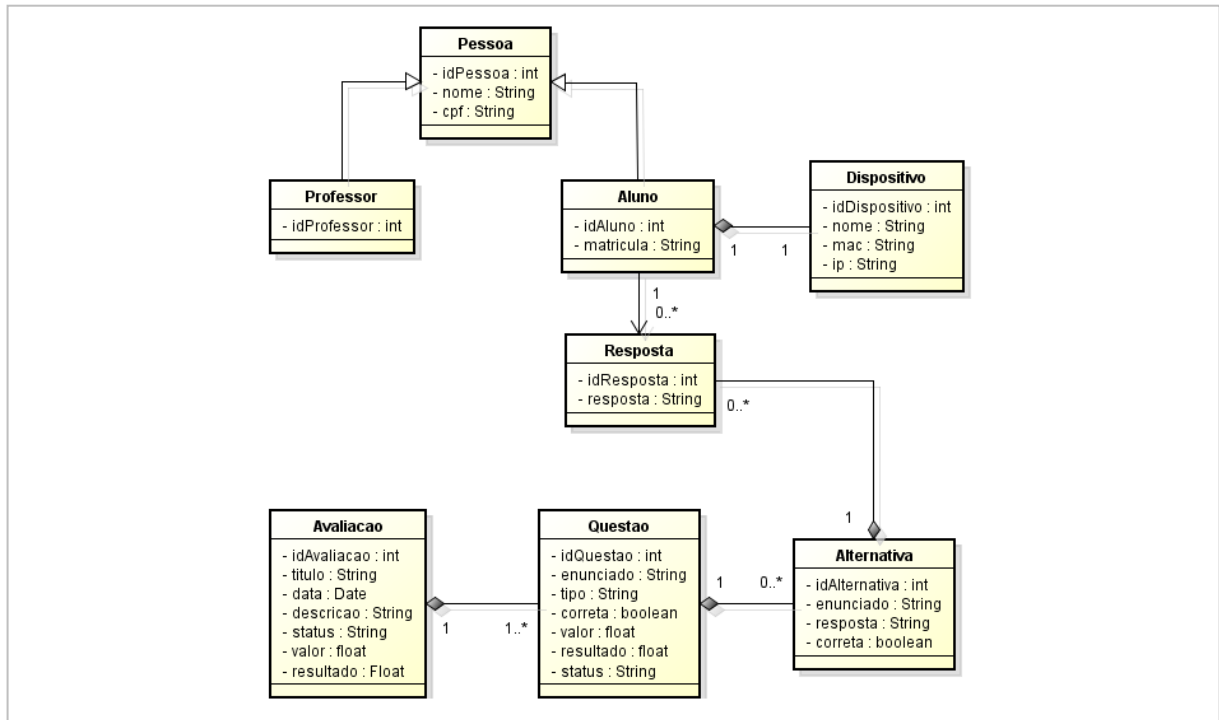
É válido frisar que esta troca de dados através do WiFi, proposta pelos requisitos funcionais citados, deve ocorrer através de uma conexão criada a partir da comunicação deste aplicativo Desktop com o aplicativo Android, instalado nos dispositivos cliente e apto a interpretar o protocolo que irá criar esta conexão.

Devido ao fato dos requisitos não funcionais estarem fortemente ligados às expectativas de desempenho, estrutura e etc., típicas em um relacionamento em que há desenvolvedor e um cliente, não faz sentido elencá-los em um aplicativo puramente acadêmico.

#### 4.2.2. Diagrama de Classes do Saara Desktop

Para a implementação da aplicação Desktop, e baseado em seus requisitos funcionais, chegou-se a estrutura das classes de domínio necessárias para tal que se vê na figura 12.

**Figura 12.** Diagrama de Classes de Domínio do aplicativo Saara Desktop.



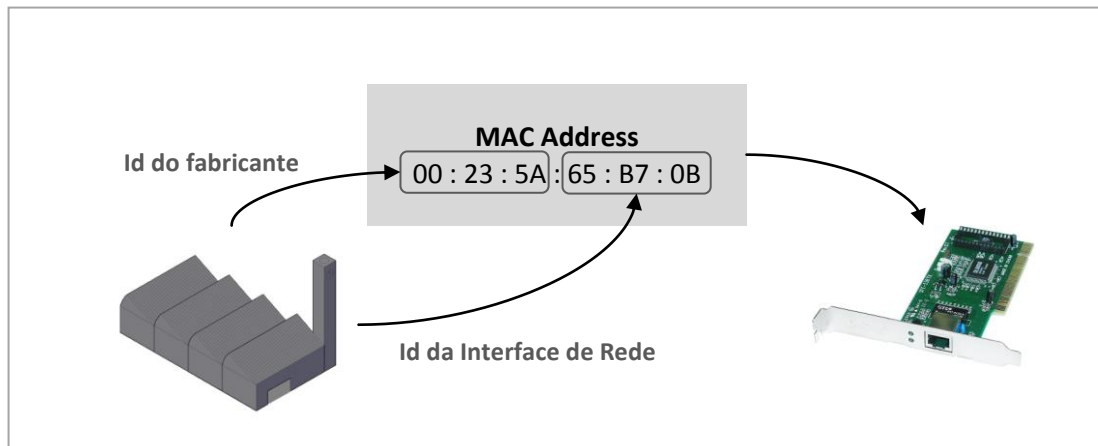
Fonte: Própria.

Para o desenvolvimento da aplicação Saara Desktop, foi suficiente a criação das classes descritas na figura 12, com um destaque para o relacionamento entre as classes: Dispositivo e Aluno, além das classes que compõe os registros de Avaliação para a aplicação.

Dentro da classe Dispositivo existe o atributo 'mac', que armazena o endereço físico dos dispositivos (MAC, do inglês: *Media Access Control*) pela representação única de sua interface de rede. A proposta do *MAC Address*, como também é conhecido, é de ser único para cada e qualquer dispositivo que esteja em uma rede, para que, dessa forma, se possa garantir que não haverá conflitos em uma rede ou sub-rede (CONDEY & DARCEY, 2010). Outra característica importante desse atributo é que ele é imutável, ou melhor, para ser mais preciso, não é possível modificar um *MAC Address* a menos que se faça uma alteração física no hardware de interface de rede.

Dessa forma, este atributo é uma chave para garantir o endereçamento correto de pacotes a um dispositivo físico dentro de uma rede, já que o endereço IP – *Internet Protocol Address* – de um dispositivo conectado a rede pode ser dinâmico, ou seja, dependendo das configurações do dispositivo e da rede, o endereço IP que também é um endereço de rede, pode mudar a cada entrada do dispositivo na rede.

**Figura 13.** Esquema de atribuição de um MAC a uma Interface de Rede.



Fonte: Própria.

É importante citar, no entanto, que recentemente houve uma implantação de mudanças no endereçamento de rede dos dispositivos, e os padrões do IPv4, baseados em 32 bits, estão sendo substituídos, ou ampliados, pelos padrões IPv6, baseados em 128 bits. Embora o esquema representado na figura 13 possibilite teoricamente quatro bilhões de combinações diferentes, este número já está se tornando escasso (ROGERS et al., 2009).

Atualmente já não se trata unicamente do endereçamento de computadores, roteadores e etc., mas também de celulares, veículos, *tablets* e até equipamentos eletrodomésticos, e é dessa maneira que o número de dispositivos aumenta exponencialmente. Só para constar, o IPv6 é um padrão de endereçamento que permite alguns milhares de bilhões de combinações.

Por fim, outra classe importante é a classe Avaliação, e as classes que mantêm um relacionamento de composição: Questão e Alternativa. O relacionamento delas é bastante óbvio; Uma avaliação possui várias questões e cada questão, pode ter nenhuma ou várias alternativas. Nos casos em que não há nenhuma alternativa, espera-se que a questão seja do tipo discursiva.

#### 4.2.3. Casos de Uso do Saara Desktop

Abaixo vemos listados os principais Casos de Uso mais importantes da aplicação, foram ignorados os Casos de Uso referentes a cadastro de registro.

**UC1 Gerenciar Conexão dos Dispositivos:** O professor inicia o Servidor Saara e aguarda a conexão dos dispositivos, visualmente, ele será informado que dispositivos foram

reconhecidos como válidos (previamente cadastrados) e que dispositivos não foram reconhecidos;

**UC2 Selecionar e Enviar as Avaliações:** Uma vez listados, os dispositivos válidos estão aptos ao recebimento das avaliações cuja entrega é disparado depois do professor haver selecionado a avaliação e confirmado o envio;

**UC3 Receber as Avaliações:** O Servidor Saara pode ser configurado para somente aceitar as respostas dos dispositivos cliente a partir de um determinado prazo, no entanto, por padrão ele está apto ao recebimento de resposta a qualquer momento, além disso, o recebimento precisa ser confirmado pelo professor;

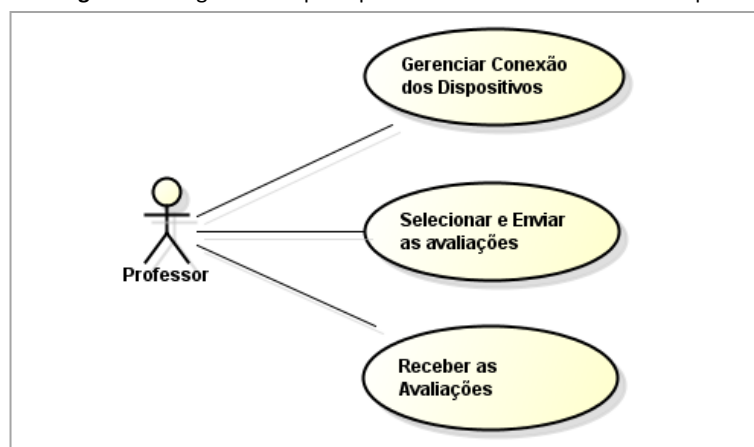
O UC1 é o momento inicial, em que o professor ativa o Servidor Saara, que obviamente irá fazer uso da conexão WiFi para se comunicar os dispositivos cliente. Sua primeira ação é ficar esperando mensagens com um conteúdo específico das aplicações cliente que ele reconhece como sendo requisições de conexão.

Uma vez que o professor tenha tratado os dispositivos que fazem parte ou não do grupo que deve receber a avaliação, no UC2 ele deve escolher que avaliação o Saara Desktop deve distribuir para os alunos.

Feita a distribuição da avaliação, o professor deve ficar atento às notificações de entrega de cada avaliação, isso porque ele deve ativamente informar ao sistema que está ciente do recebimento, e isso, através do UC3, se dá numa forma de protocolar o procedimento.

Estes que são os principais Casos de Uso do aplicativo Saara Desktop estão ilustrados do Diagrama de Casos de Uso da figura 14.

**Figura 14.** Diagrama dos principais Casos de Uso do Saara Desktop.



Fonte: Própria.

#### 4.2.4. Especificação de Requisitos do Saara Android

Para permitir que dispositivos com Android como sistema operacional, versão 2.2 ou posterior, se conectem com um dispositivo Desktop, por meio de WiFi, a fim de receber, navegar e responder questões de uma avaliação acadêmica, o software precisa atender ao seguintes requisito mínimos:

**RF1 Conexão com o Servidor Saara:** Deve ser possível se conectar ao Servidor Saara por meio de uma rede WiFi;

**RF2 Recebimento de Avaliação:** Deve ser possível receber, tratar e persistir em um banco de dados, os dados que compõe a avaliação acadêmica recebida. O tratamento recupera os dados do documento XML criptografado recebido;

**RF3 Navegação e Resposta da Avaliação:** Deve ser possível navegar, responder e persistir em banco de dados as respostas da avaliação recebida;

**RF4 Envio de Respostas:** Deve ser possível recuperar do banco de dados e converter os registros, contendo as respostas da avaliação, em formato XML criptografado e enviá-los ao Servidor Saara.

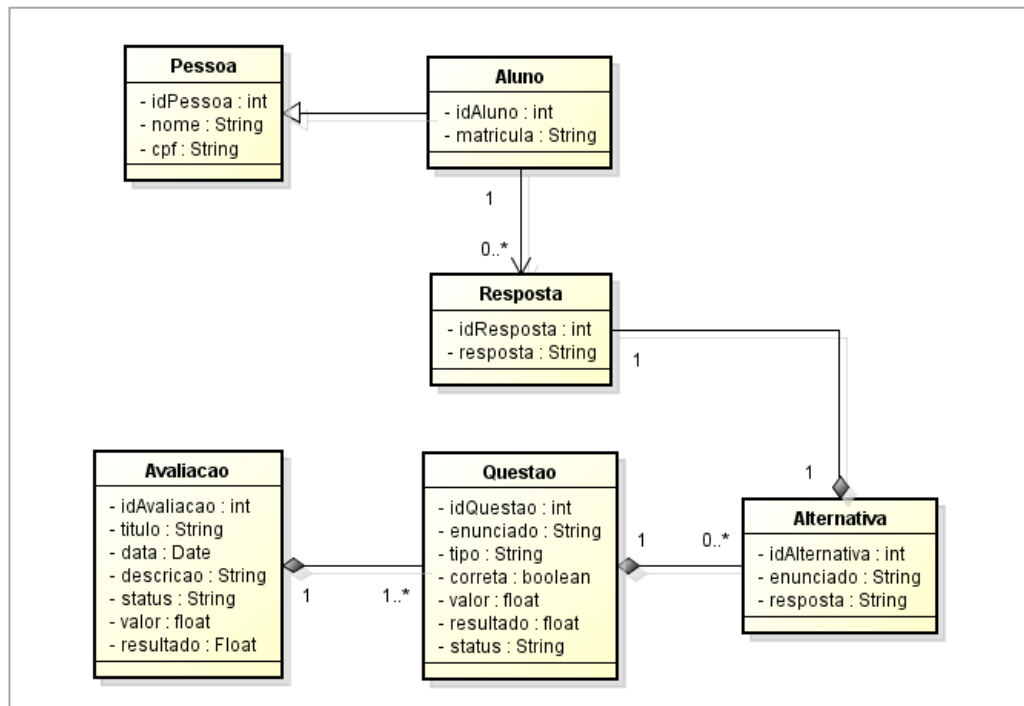
A persistência dos dados se faz necessária devido às características do ambiente de um hardware, que é um dispositivo móvel, mais sujeito a uma parada brusca em função do fim da carga da bateria, por exemplo. Em dispositivos móveis a memória de Cache e RAM não são lugares seguros para guardar dados.

Outra razão é ser menos complicado o processo de leitura e gravação de dados em um banco de dados, no caso do Android, o SQLite, do que em um arquivo do sistema de diretório.

#### 4.2.5. Diagrama de Classes do Saara Android

As classes presentes na aplicação Saara Android são praticamente as mesmas classes presentes na aplicação Saara Desktop, salvo duas exceções. São elas as classes Professor e Dispositivo. Como é possível ver na figura 15. Além disso, por razões óbvias, não faz nenhum sentido manter o atributo booleano 'correta' na aplicação cliente, já que ele que determina se esta, dentre as alternativas, é a correta.

Figura 15. Diagrama de Classes de Domínio da Aplicação Saara Android.



Fonte: Própria.

Uma vez exposto que se tratam das mesmas classes, mais detalhes sobre estas classes já foram dados no tópico Diagrama de Classes do Saara Desktop.

É importante citar que a vinculação do aluno a um MAC se dá por meio do protocolo de conexão enviado pelo aplicativo Saara Android ao Servidor do aplicativo Saara Desktop, este protocolo é composto por uma mensagem e o MAC do dispositivo que a enviou. O IP do dispositivo é recuperado pelo Socket recebido através do método `getRemoteSocketAddress()`.

#### 4.2.6. Casos de Uso do Saara Android

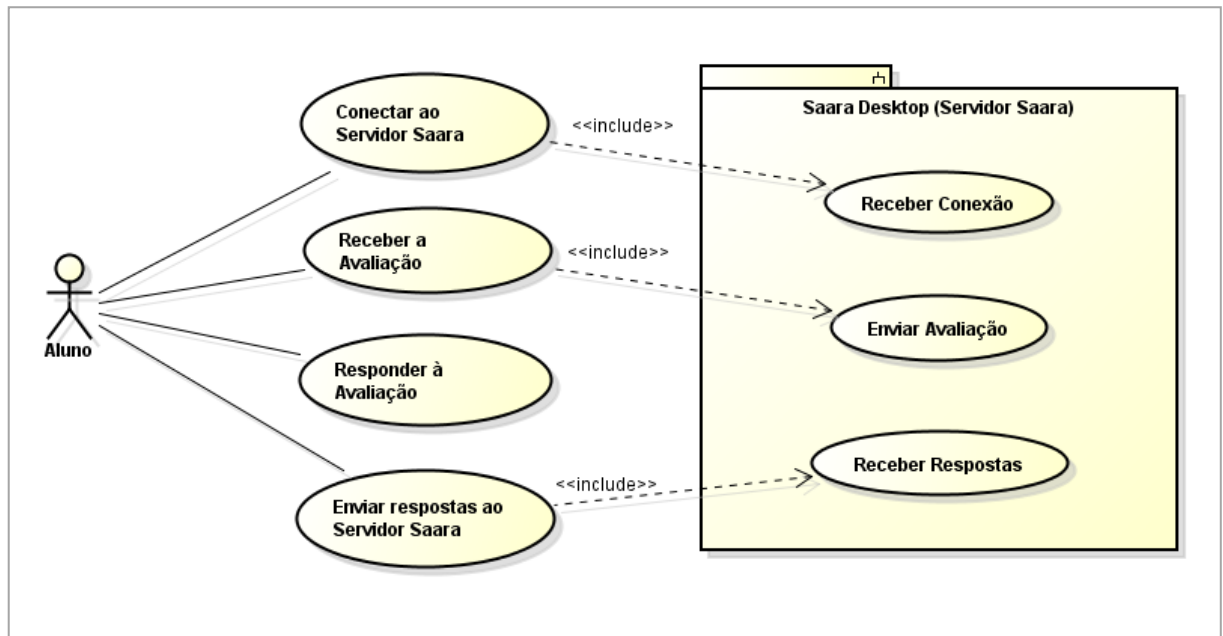
**UC1 Conectar ao Servidor Saara:** O aluno deve informar o IP do Servidor Saara, anunciado pelo professor, e solicitar a conexão via WiFi;

**UC2 Receber a Avaliação:** O aluno aguardar a liberação do professor e então proceder o recebimento da avaliação;

**UC3 Responder à Avaliação:** Uma vez recebida a avaliação, o aluno pode navegar através das questões e respondê-las, selecionando uma alternativa ou respondendo discursivamente;

**UC4 Enviar Respostas ao Servidor Saara:** Ao finalizar todas as questões, o aluno deve aguardar a autorização do professor, se for o caso, para enviar suas respostas ao Servidor Saara, informado no UC1.

**Figura 16.** Diagrama dos principais Casos de Uso do Saara Android.



. Fonte: Própria.

Um detalhe importante, que ficou claro na especificação de seus requisitos, é que a aplicação Saara Android está fortemente acoplada ao Servidor Saara, consequentemente, praticamente todos os seus Casos de Uso possuem uma dependência com a aplicação Saara Desktop. Percebe-se essa dependência ilustrada na figura 16, que contém o Diagrama de Casos de Uso do Saara Android.

#### 4.3. IMPLEMENTAÇÃO DA APLICAÇÃO SAARA DESKTOP

A implementação do Saara Desktop passou por duas etapas distintas uma (a) que envolveu a implementação da lógica e gerenciamento da persistência dos dados e outra (b) que envolveu a implementação do servidor e uma classe que gerenciasse o status da conexão dos dispositivos móveis.

A primeira etapa, menos complexa, envolveu a implementação dos cadastros para a persistência dos dados no banco SQLite, e, como já foi abordado, o SQLite é um banco de dados leve, com nenhuma necessidade de configuração e permite que a aplicação leve consigo a *engine* do banco, tornando-a *standalone*.

Para o acesso aos dados, adotou-se o padrão de desenvolvimento DAO – *Data Access Object*, um *pattern* Java *Enterprise Edition* – J2EE, sua filosofia é simples e está fundamentada na idéia de centralização da responsabilidade de acesso aos dados em uma única classe, ao invés de espalhar métodos que acessam o banco por toda a aplicação, criando um alto acoplamento e um grande problema em caso de manutenção do código. Na figura 17, podemos ver um exemplo da implementação do padrão de desenvolvimento DAO. Todos os métodos necessários para manipular registros no banco de dados, referentes a objetos da classe Aluno, por exemplo, ficam nesta classe que precisa unicamente ser instanciada para que seus métodos sejam acessados.

**Figura 17.** Exemplo da estrutura de uma classe DAO.

```
package br.com.saara.dao;

import java.sql.ResultSet;

public class AlunoDAO {

    private ConexaoDAO conexao;
    private Statement statement;

    public AlunoDAO() {
        conexao = ConexaoDAO.getInstance();
    }

    public void SalvarAluno(Aluno aluno){}

    public boolean ApagarAluno(Aluno aluno){}

    public List<Aluno> RecuperaAlunos(){[]}

    public Aluno recuperarAlunoPorMAC(String mac){[]}

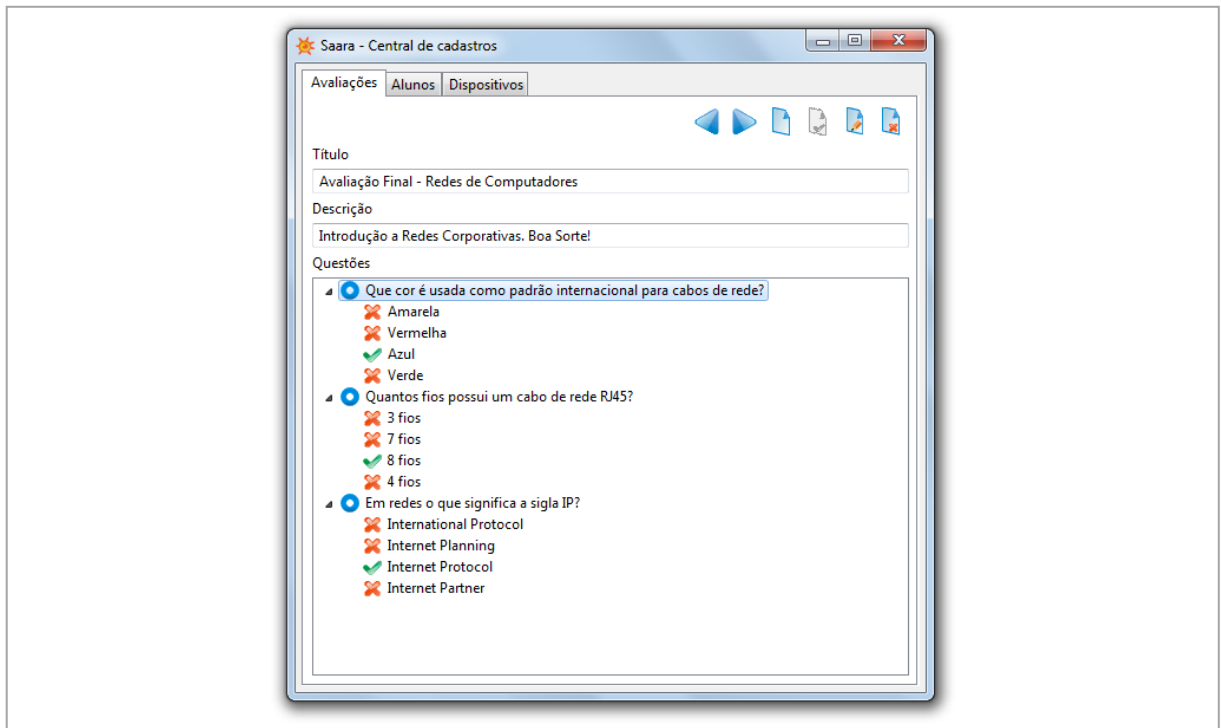
    public boolean DesvincularDispositivo(int idAluno){[]}

}
```

Fonte: Própria.

Na figura 18 vemos o exemplo da tela de cadastro de uma Avaliação, que é uma instância da classe CentralDeCadastros.java (que estende org.eclipse.swt.widgets.shell), os botões de manipulação dos dados: anterior, próximo, novo, salvar, editar e excluir, fazem apenas uma referência aos métodos da classe AvaliacaoDAO.java.

**Figura 18.** Janela de Cadastro de Avaliações.



Fonte: Própria.

Concluída a etapa da persistência e manipulação dos registros, iniciou-se uma segunda etapa em que foram implementadas as classes e os métodos que tornam possível a comunicação, via WiFi, com a aplicação Saara Android.

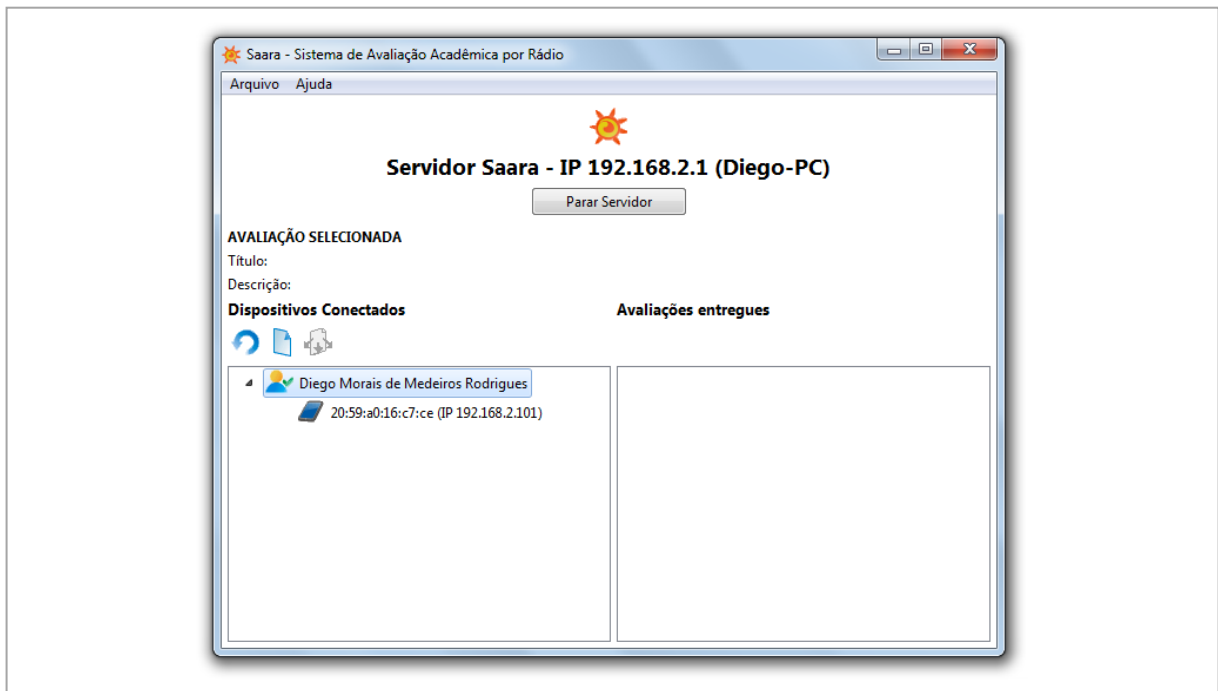
Pode-se dizer que esta etapa foi também a mais complexa da implementação do Saara Desktop, isso porque envolveu a criação de uma classe que se comportasse como servidor TCP/IP e de uma classe que gerenciasse os dispositivos conectados. Antes de abordarmos a implementação destas classes em si, vejamos a figura 19, que mostra a tela principal do programa, em uma visão panorâmica desta tela temos como componentes principais:

**Um painel superior central:** Neste container fica o IP do Servidor Saara, enquanto este estiver ativado, além de um botão para ativar e desativar o Servidor; A informação do IP é propositalmente destacada, já que é através deste endereço que os dispositivos móveis irão se conectar ao Servidor;

**Um painel à esquerda:** Aqui o professor pode visualizar que avaliação foi selecionada para a distribuição, os botões para, da esquerda para direita, atualizar o registro dos dispositivos conectados, selecionar uma avaliação e distribuir as avaliações. A lista dos dispositivos, na verdade, exibe prioritariamente os alunos conectados, para visualizar um dispositivo é preciso expandir o item com o nome de um aluno associado a ele;

**Um painel à direita:** Depois de seleccionar um aluno do lado direito, é possível visualizar neste painel a avaliação entregue, se for o caso, inclusive com as questões corretas assinaladas por um '*check mark*' em verde, enquanto as incorretas são representadas por um 'X' em vermelho.

**Figura 19.** Tela principal do Saara Desktop.



Fonte: Própria.

A partir deste ponto do tópico, as duas classes chave vão ser abordadas, são elas: `ServidorSaara.java` e `GerenciadorDeConexoes.java`. São essas classes as responsáveis pelo envio e recebimento de pacotes, com o tratamento apropriado dependendo do tipo de conteúdo e o gerenciamento dos dispositivos conectados, ou seja, aptos a receber avaliações e enviar as respectivas respostas.

A classe `ServidorSaara.java` estende a classe `Thread` do pacote `java.lang.thread`, para estendê-la, é preciso implementar o método `run()` com as instruções que serão executadas pelo `thread`. Essa classe foi estendida porque ela permite a criação de um novo processo para a aplicação de forma transparente para o desenvolvedor. Esse recurso permite a execução de tarefas que seriam inviáveis sem o uso de um `thread` porque congelariam a aplicação.

Uma tradução livre para *thread* seria fio, filamento ou linha. No contexto da informática, *thread* significa um processo ou tarefa para o processador.

Uma aplicação Java é executada com base em um único *thread*, e esse fato explica porque operações longas, como um loop demorado, por exemplo, retornam ao usuário o famigerado: ‘Aplicação não está respondendo’. Isso ocorre porque o thread principal da aplicação está ocupado processando uma tarefa longa e o sistema operacional entende que, por isso, a aplicação está inapta a ‘responder’ qualquer interação do usuário.

No caso da classe `ServidorSaara.java`, o uso de um *thread* se faz necessário porque o método `accept()`, da classe `ServerSocket` do pacote `java.net.ServerSocket`, paralisaria a aplicação enquanto espera por novas conexões, seu uso pode ser visto no método `run()` na estrutura da classe `ServidorSaara`, mostrada na figura 20.

Figura 20. Estrutura da classe `ServidorSaara.java`.

```
package br.com.saara.wifi.conexoes;

import java.io.File;

public class ServidorSaara extends Thread {

    private boolean ativo;
    private String nome;
    private SaaraMain saaraMain;

    public ServidorSaara() { }

    public void apertarMao(Socket socket){}

    public void run() {
        try {
            this.ativo = true;
            ServerSocket serverSocket = new ServerSocket(9096);
            System.out.println("Servidor Saara iniciado...");

            while (this.ativo) {
                apertarMao(serverSocket.accept());
            }

            System.out.println("Servidor Saara encerrado.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public boolean distribuirAvaliacaoes() {}
    public ArrayList<Resposta> retornaRespostas(String arquivoXML) {}
    private static String getTagValue(String sTag, Element eElement) {}
    public boolean isAtivo() {}
    public void setAtivo(boolean ativo) {}
    public String getIp() {}
    public String getNome() {}
    public void setNome(String nome) {}
    public SaaraMain getSaaraMain() {}
    public void setSaaraMain(SaaraMain saaraMain) {}
}
```

Fonte: Própria.

Ditas estas coisas, o mais importante aspecto agora é entender como esse servidor, que está em um *thread* específico, se comunica com o *thread* principal da aplicação. A solução encontrada foi um o padrão *singleton* de desenvolvimento. A idéia por trás deste padrão é simples: uma instância única de um objeto durante todo o ciclo de vida da aplicação.

Estamos falando da classe `GerenciadorDeConexoes.java`, um *singleton* com a responsabilidade de armazenar, enquanto a aplicação estiver em execução, os dispositivos que realizaram conexão. A estrutura dessa classe pode ser observada na figura 21.

**Figura 21.** Estrutura da classe `GerenciadorDeConexoes.java`.

```
package br.com.saara.wifi.conexoes;

import java.util.ArrayList;

public class GerenciadorDeConexoes {

    private static GerenciadorDeConexoes gerenciadorDeConexoes;
    private List<Dispositivo> dispositivosConectados = new ArrayList<Dispositivo>();
    private SaaraMain saaraMain;

    private GerenciadorDeConexoes(){

    }

    public static GerenciadorDeConexoes getInstance(){
        if(gerenciadorDeConexoes == null){
            synchronized (GerenciadorDeConexoes.class){
                if(gerenciadorDeConexoes == null){
                    gerenciadorDeConexoes = new GerenciadorDeConexoes();
                }
            }
        }
        return gerenciadorDeConexoes;
    }

    public void adicionarDispositivo(Dispositivo dispositivo){

    }

    public List<Dispositivo> getDispositivosConectados() {

    }

    public SaaraMain getSaaraMain() {

    }

    public void setSaaraMain(SaaraMain saaraMain) {

    }

}
```

Fonte: Própria.

Ao receber um *socket* cujo conteúdo seja o protocolo aguardado e o MAC de um dispositivo, o Servidor Saara cria um objeto da classe `Dispositivo` e o adiciona na lista de dispositivos chamada `dispositivosConectados`, que pode ser acessada por qualquer classe da aplicação para visualizar os dispositivos que estão conectados.

O fato de ser *singleton* garante que a lista de dispositivos conectados seja única, dessa forma, várias classes da aplicação podem acessá-la sem inconsistências.

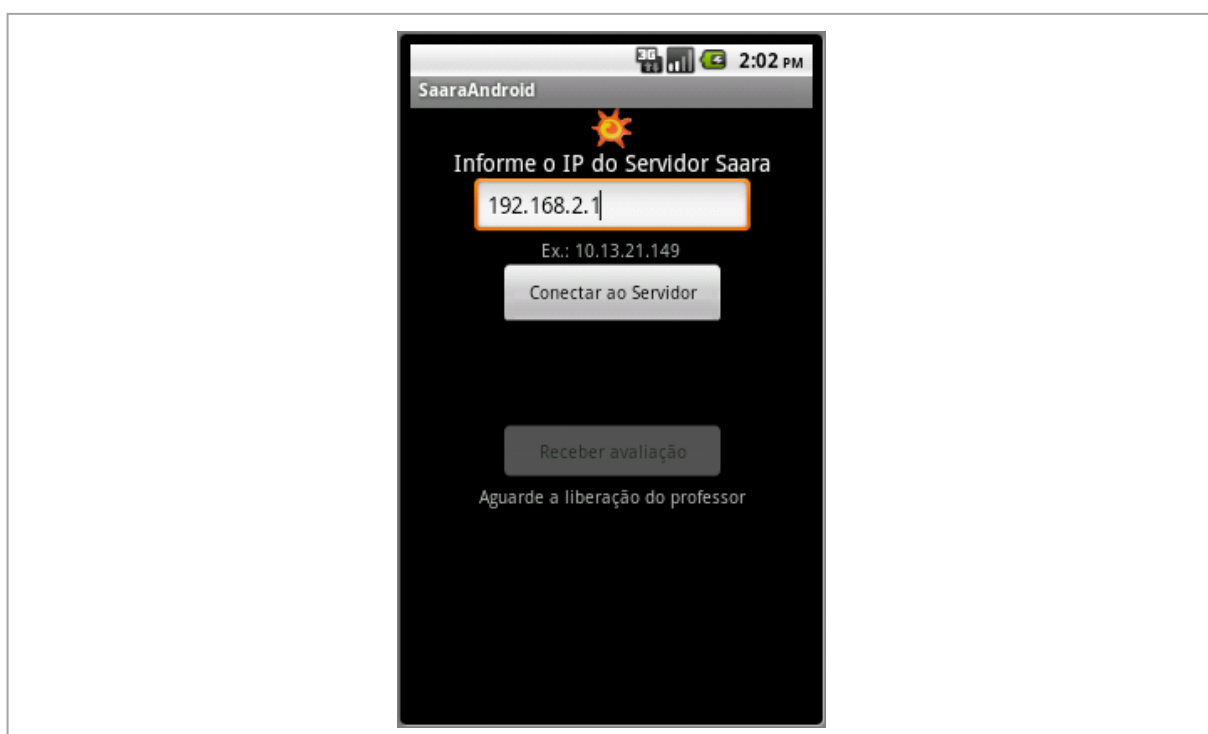
#### 4.4. IMPLEMENTAÇÃO DA APLICAÇÃO SAARA ANDROID

A aplicação Saara Android foi desenvolvida com base em um princípio: deveria ser simples e intuitiva, de forma que ao primeiro contato o aluno usuário não tivesse dificuldades para interagir com sua interface.

Simplicidade e intuitividade na composição da interface desta aplicação são aspectos especialmente delicados porque o seu funcionamento envolve, inevitavelmente, conceitos de rede como endereço IP e portas lógicas, acerca dos quais os alunos não deveriam precisar ter conhecimento prévio para interagir com a aplicação.

Na figura 22 pode-se observar a primeira tela do Saara Android, nela a primeira interação é para informar o IP do Servidor Saara, que deve ter sido divulgado pelo professor. É importante citar que importantes operações como conectar ao servidor, receber a avaliação e enviar as respostas, terão perto dos botões que as disparam dicas de como proceder.

**Figura 22.** Tela inicial da aplicação Saara Android.



Fonte: Própria.

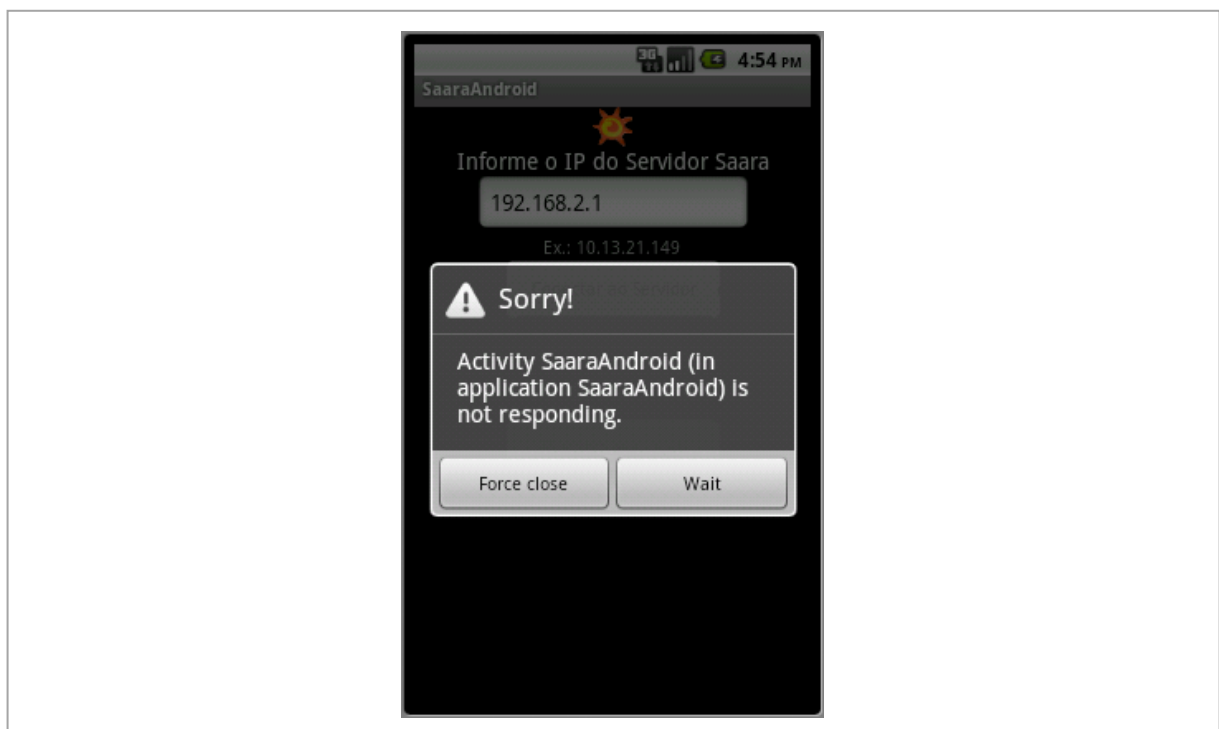
As portas lógicas utilizadas pelo dispositivo para receber uma avaliação ou enviar as respostas são pré-definidas e não podem ser alteradas, esta estratégia permitiu que o conceito de porta lógica fosse algo transparente para o aluno usuário. Tanto a aplicação Desktop quanto o Saara Android irão se comunicar sem que este parâmetro precise ser configurado.

Para realizar a conexão com o Servidor Saara, a aplicação envia uma mensagem contendo um protocolo específico, composto pela cadeia de caracteres “SaaraAndroid” seguida por um espaço e o endereço MAC do dispositivo.

Assim como em uma aplicação Java Desktop, uma aplicação Android também é executada com base em um *thread* principal, de forma que se esta interação com o Servidor Saara fosse executada normalmente, a interface da aplicação ficaria suspensa, já que a conclusão desta execução requer uma resposta do Servidor Saara, que pode não ser imediata.

A abordagem dada pelo sistema operacional Android é muito semelhante a do Windows para casos em que a aplicação deixe de responder. Como é possível observar na figura 23, o sistema operacional Android irá perguntar ao usuário se ele deseja fechar aplicação ou esperar por uma resposta, caso uma execução exceda algo em torno de cinco ou dez segundos.

**Figura 23.** Mensagem quando uma aplicação Android deixa de responder.



Fonte: Própria.

Para evitar esse problema, mais uma vez a solução é criar um novo *thread* que permitirá ao aplicativo que a conexão seja realizada sem que a interação da interface com o usuário

seja interrompida. No trecho de código exibido na figura 24, vemos como isso pode ser feito. Esse código foi incluído no evento onCreate() da activity ReceberAvaliações.

**Figura 24.** Trecho de código com a criação de um thread em Android.

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {

            ServerSocket serverSocket = new ServerSocket(9099);
            serverSocket.setSoTimeout(0);
            RepositorioMensagens.getInstance().setSocket(serverSocket.accept());
            RepositorioMensagens.getInstance().setRecebimentoAvaliacao(true);

        } catch (IOException e) {
            Toast.makeText(ReceberAvaliacaoActivity.this, "Erro: " + e.getMessage(), 7000).show();
        }
    }
}).start();
```

Fonte: Própria.

Uma nota muito importante sobre *threads* em Android, é que criar uma classe que estenda a classe *thread*, do pacote `java.lang.thread`, pode não produzir o resultado esperado, ou seja, se nessa classe houver uma execução longa, o indesejável efeito de ‘congelamento’ da tela irá acontecer caso haja interação desse thread com a interface do usuário, além disso significar uma violação modelo *single-threaded* do Android UI toolkit.

A forma apresentada na figura 24 é válida neste contexto porque não atualiza o estado de nenhum componente da interface do usuário, para um *thread* que precise interagir com a interface da aplicação é preciso fazer uso da classe `AsyncTask`, do pacote `android.os.AsyncTask`, que é a forma recomendada pelas especificações de desenvolvimento Android (GOOGLE, 2011d).

De volta ao código da figura 24, que, como já foi dito, está inserido no evento onCreate() da atividade responsável pelo recebimento do arquivo ‘saara\_avaliacao.xml’, e trata-se da criação de um objeto da classe `ServerSocket`, do pacote `java.net.ServerSocket`, que estará esperando um socket do Servidor Saara na porta 9099 com o referido arquivo.

O método `serverSocket.accept()` irá bloquear o *thread* enquanto aguarda a chegada do socket contendo o arquivo XML que representa a avaliação, cujo exemplo é mostrado na figura 25. Este método retorna o socket recebido para um *setter* da classe `RepositorioMensagens`, uma solução interessante para o problema de comunicação entre activities no Android, que será abordada no tópico seguinte.

**Figura 25.** Exemplo da estrutura XML de uma avaliação do Saara.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<avaliacao id="7">
  <titulo>INTRODUÇÃO A REDES DE COMPUTADORES</titulo>
  <descricao>Esta avaliação abrange E1 e E2. Boa Sorte!</descricao>
  <questao id="4">
    <enunciado>Quantos fios tem um cabo de rede RJ45?</enunciado>
    <alternativa id="15">7 fios;</alternativa>
    <alternativa id="16">5 fios;</alternativa>
    <alternativa id="18">8 fios;</alternativa>
    <alternativa id="19">2 fios entrelaçados;</alternativa>
  </questao>
  <questao id="5">
    <enunciado>Qual é a abrangência do padrão RJ45?</enunciado>
    <alternativa id="5">Mundial</alternativa>
    <alternativa id="6">Somente na América Latina</alternativa>
    <alternativa id="14">Brasileira</alternativa>
    <alternativa id="23">EUA e Europa</alternativa>
  </questao>
  <questao id="11">
    <enunciado>Que cor é usada como padrão internacional para cabos de
      rede?</enunciado>
    <alternativa id="9">Amarela</alternativa>
    <alternativa id="10">Vermelha</alternativa>
    <alternativa id="11">Azul</alternativa>
    <alternativa id="13">Verde</alternativa>
  </questao>
</avaliacao>
```

Fonte: Própria.

#### 4.4.1. Repositório de Mensagens: Comunicação entre Objetos de uma Aplicação Android

O objetivo deste tópico é abordar a técnica utilizada na implementação do Saara Android que substituiu, nesta aplicação, a maneira tradicional com a qual *activities* trocam dados entre si.

Normalmente, o processo de troca de dados entre *activities* ocorre por meio do método `Intent.putExtra()`, que recebe dois parâmetros: o primeiro do tipo `string`, com o nome do ‘extra’ e o segundo com o valor, que pode ser do tipo `string`, `int`, `boolean` etc.

Para recuperar este valor, na *activity* criada pelo *intent* que ‘colocou’ os valores, basta invocar o método `getIntent().getIntExtra()` para recuperar um inteiro, `getIntent().getStringExtra()` para recuperar um valor do tipo `string`, e assim por diante. De maneira geral este método espera dois argumentos, o primeiro é do tipo `string` e espera o nome do ‘extra’, já o segundo deve ser compatível com o tipo de retorno e representa o valor padrão a ser retornado caso o ‘extra’ esteja vazio.

Para troca de dados simples e para uma *activity* apenas, é uma solução viável e descomplicada, mas quando se trata de trocar dados entre mais de duas *activities*, e mesmo

enviar um dado a uma *activity* que só será invocada por uma terceira *activity*, a troca de extras pode não ser uma boa escolha.

Como opção para melhorar a troca de dados entre *activities* é possível criar um *Service* que receba esta responsabilidade de receber e distribuir dados, entretanto, como já foi abordado no Referencial Teórico deste trabalho, *Services* requerem registro no arquivo *AndroidManifest* e para a implementação dessa comunicação precisam ser do tipo *Bounded Services*, invocados pelo método *bindService()* de uma *activity*.

Além de todos estes aspectos de configuração, um *Bounded Service* ainda gera a preocupação com gerenciamento de seu ciclo de vida, pelo fato de poder ser destruído pelo sistema enquanto não houver nenhuma *activity* em *foreground* conectado a ele.

A proposta de solução para esta questão veio do Saara Desktop e uma de suas classes: *GerenciadorDeConexoes.java*. Uma classe no padrão *singleton* que compartilha a lista dos dispositivos conectados pelo *thread* do Servidor Saara com várias outras classes da aplicação.

A classe *RepositorioMensagens* também é um *singleton*, porém com uma única regra de ouro – que tomei a liberdade de criar para este projeto – para evitar o caos na troca de mensagens, que é a seguinte: “*Write once. Read Everywhere.*”

Ou seja, uma mensagem, que é um atributo da classe *RepositorioMensagens*, que, obviamente, pode ser de qualquer tipo, inclusive um objeto, só deve ser atribuído uma única vez, mas pode ser consumido por qualquer outro, ou vários outros componentes de uma aplicação Android: *Service* ou *Activity*.

Se um atributo da classe *RepositorioMensagens.java* precisar ser modificado mais de uma vez, é melhor considerar a criação de um novo atributo específico para o fim de atender a esta segunda modificação.

Uma maneira de facilitar o entendimento do estado e o uso de uma mensagem no repositório é com o uso do Java Doc, que no Eclipse se dá na forma como vista no método *setIdAluno()*, onde o comentário ‘*/\*\**’ pode ser seguido por uma variável antecedita pelo ‘*@*’ que pode ser *param*, *author*, *return* e etc. Na figura 26, é possível ver uma parte da estrutura da classe *RepositorioMensagens.java* da aplicação Saara Android.

Figura 26. Estrutura da Classe Repositório de Mensagens.

```

+ import java.net.Socket;
+
+ * @author Diego Morais
+ public class RepositorioMensagens {
+
+     private static RepositorioMensagens repositorioMensagens;
+
+     private String IpServidor;
+     private int idAluno;
+     private int idAvaliacao;
+     private boolean RecebimentoAvaliacao;
+     private String MAC;
+     private String abertoDeMao;
+
+     private Avaliacao avaliacao;
+
+     private Socket socket;
+
+     public String getAbertoDeMao() {}
+
+     public void setAbertoDeMao(String abertoDeMao) {}
+
+     public String getMAC() {}
+
+     public void setMAC(String mAC) {}
+
+     public Avaliacao getAvaliacao() {}
+
+     public void setAvaliacao(Avaliacao avaliacao) {}
+
+     public Socket getSocket() {}
+
+     public void setSocket(Socket socket) {}
+
+     public int getIdAluno() {}
+     * @param idAluno Setado uma única vez pelo método ReceberAvaliacaoActivity.retornaAvaliacao()
+     public void setIdAluno(int idAluno) {}
+
+     public int getIdAvaliacao() {}
+     * @param idAvaliacao Setado uma única vez pelo método ReceberAvaliacaoActivity.retornaAvaliacao()
+     public void setIdAvaliacao(int idAvaliacao) {}
+
+ }

```

Fonte: Própria.

#### 4.4.2. Persistência de Dados em Android

Criar um banco de dados para uma aplicação Android é uma tarefa relativamente simples, entretanto, se trata de um conjunto de técnicas muito específicas da plataforma.

O primeiro passo é criar uma classe que representará (ROGERS *et. al.*; 2009) o banco de dados. Essa classe irá estender a classe abstrata `android.database.sqlite.SQLiteOpenHelper` e implementar, basicamente, dois métodos: `onCreate()` e `onUpgrade()`.

O método `onCreate()` será invocado sempre que a aplicação tentar executar qualquer instrução SQL, esteja o banco de dados criado ou não, por este motivo, nesse método devem estar contidas as instruções de criação do banco de dados da aplicação. Além disso, é preciso definir pelo menos duas constantes dessa classe: `DATABASE_NAME` do tipo `String` e `DATABASE_VERSION` do tipo `int`.

O método `onUpgrade()` será executado se existir a tentativa de criar um novo banco de dados com o mesmo nome e uma versão, descrita na constante `DATABASE_VERSION`, diferente. A figura 27 contém uma parte da classe `SaaraDatabase`.

**Figura 27.** Fragmento da classe `SaaraDatabase`.

```
public class SaaraDatabase extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "Saaradb";
    private static final int DATABASE_VERSION = 1;
    private final Context dbContext;

    public SaaraDatabase(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        this.dbContext = context;
    }

    public void onCreate(SQLiteDatabase db) {}

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}

    public void salvarResposta(Resposta resposta){
        try{
            String sql = "INSERT INTO Respostas (idAvaliacao, idQuestao, idAluno, resposta) VALUES ('"+
                resposta.getAvaliacao().getIdAvaliacao()+"', '"+resposta.getQuestao().getIdQuestao()+"', '"+
                resposta.getAluno().getIdAluno()+"', '"+resposta.getResposta()+"')";

            getWritableDatabase().execSQL(sql);

        }catch (Exception e) {
            Toast.makeText(dbContext, "Erro: "+e.getMessage(), 7000).show();
        }
    }
}
```

Fonte: Própria.

A figura 27 contém um fragmento com o exemplo de uma atualização no banco de dados, nesse caso, nenhum registro é retornado. No entanto, a aplicação `Saara Android` precisa realizar consultas em seu banco de dados para recuperar registros de respostas, por exemplo.

Para que isso seja possível, é preciso criar uma classe que irá implementar um 'Cursor' para navegar em um conjunto de registros que tenha retornado do banco de dados através de uma instrução `SELECT`. É possível observar no exemplo de um método que faz uma consulta ao banco de dados na figura 28 a criação de uma instância da classe `RespostaCursor`, cujo objeto permite a navegação nos registros retornados pela instrução `SELECT` em questão.

**Figura 28.** Exemplo de uma consulta ao Banco de Dados em uma aplicação Android.

```

public Questao proximaQuestao(int idAvaliacao){
    ArrayList<Questao> questoes = new ArrayList<Questao>();

    try{
        String sql = "SELECT idQuestao FROM Questoes WHERE idAvaliacao = "+idAvaliacao+
            " AND idQuestao NOT IN (SELECT idQuestao FROM Respostas WHERE idAvaliacao = "+idAvaliacao+")";

        SQLiteDatabase db = getReadableDatabase();
        RespostaCursor respostaCursor = (RespostaCursor) db.rawQueryWithFactory(new RespostaCursor.Factory(), sql, null, null);

        for (int row = 0; row < respostaCursor.getCount(); row++){
            respostaCursor.moveToPosition(row);
            questoes.add(new Questao(respostaCursor.getColIdQuestao()));
        }

    }catch (Exception e) {
        Toast.makeText(dbContext, "Erro: "+e.getMessage(), 7000).show();
    }
}

```

Fonte: Própria.

Uma parte da estrutura da classe RespostaCursor pode ser observada na figura 29. Observe que uma classe como essa, que estenda a classe android.database.sqlite.SQLiteDatabase precisa ser criada para recuperar as colunas e os registros de uma consulta. Para acessar as colunas de um JOIN, por exemplo, seria preciso criar uma classe como essa contendo um *getter* como os do exemplo para cada coluna que a consulta retornasse.

**Figura 29.** Estrutura da Classe RespostaCursor.

```

import android.database.Cursor;

public class RespostaCursor extends SQLiteCursor {

    public RespostaCursor(SQLiteDatabase db, SQLiteCursorDriver driver,
        String editTable, SQLiteQuery query) {
        super(db, driver, editTable, query);
    }

    public static class Factory implements CursorFactory{
        @Override
        public Cursor newCursor(SQLiteDatabase db,
            SQLiteCursorDriver driver, String editTable,
            SQLiteQuery query) {
            return new RespostaCursor(db, driver, editTable, query);
        }
    }

    public int getColIdResposta(){
        return getInt(getColumnIndex("idResposta"));
    }
    public int getColIdAvaliacao(){
    }
    public int getColIdQuestao(){
    }
    public int getColIdAluno(){
    }
    public String getColResposta(){
    }
}

```

Fonte: Própria.

## 5. RESULTADOS

Neste tópico serão abordados os resultados obtidos com a implementação das aplicações propostas. A forma como se dá a comunicação entre elas na prática e sob uma perspectiva de usuário das aplicações Saara Desktop e Saara Android.

### 5.1. EXECUÇÃO E INTERAÇÃO DAS APLICAÇÕES DESKTOP E ANDROID

O objetivo final das aplicações Saara Desktop e Saara Android, propostas neste trabalho, é a realização de avaliações acadêmicas fazendo o uso de dispositivos móveis e uma estrutura de rede WiFi. A figura 30 ilustra uma espécie de diagrama de atividades com o papel e interações entre as aplicações para que uma avaliação seja aplicada de forma bem-sucedida.

**Figura 30.** Diagrama das atividades de interação entre as aplicações Desktop e Android.



Fonte: Própria.

Tal qual em uma prova tradicional, todo o processo é disparado pelo professor, no momento em que ativa o Servidor Saara e solicita aos alunos que realizem a conexão, cuja configuração depende apenas de dois fatores fundamentais: (a) que o Saara Android esteja na mesma rede que estiver o computador que executa o Saara Desktop e (b) que o aluno informe o IP do Servidor Saara.

À medida que os alunos realizam as conexões, através da interface do Saara Android, os dispositivos são exibidos para o professor, é importante citar que alunos cujo dispositivo não esteja previamente cadastrado, serão exibidos como ‘desconhecidos’ e não receberão a avaliação, embora o Saara Desktop reconheça seu endereço MAC e IP. A figura 30 exibe um caso em que o dispositivo não foi encontrado na base de dados.

Uma vez que tenha gerenciado os dispositivos e ordenado questões diversas, o professor deve autorizar o recebimento da avaliação e posteriormente enviá-la, depois, obviamente, de haver selecionado a avaliação escolhida para o momento.

A partir deste momento, os alunos recebem uma notificação do recebimento da avaliação e de que ‘pressionar o botão home do dispositivo não é permitido, sob pena de envio automático da avaliação’. Essa medida se faz necessária porque, uma vez que o sistema operacional Android é pensado predominantemente para telefones, nenhuma aplicação consegue controle absoluto do dispositivo, de maneira que o usuário ou o sistema operacional pode abandoná-la a qualquer momento (e.g. uma chamada), um dos recursos de que a aplicação dispõe para tratar uma parada em sua execução é sobrescrevendo o evento `onPause()` de uma *activity*.

Uma vez que inicie a avaliação, para respondê-la o aluno navega por uma interface de usuário simples em que é possível encontrar duas configurações possíveis: (a) Um enunciado com várias alternativas representadas por *Radio Buttons* para questões objetivas, ou (b) um enunciado e um *Text Box* para questões discursivas; além de um botão comum para ‘responder’ a questão e avançar para uma próxima que será escolhida randomicamente.

No momento em que se esgotarem as questões, automaticamente o aluno será encaminhado para uma tela com um único botão para enviar suas respostas ao Servidor Saara, realizando a entrega de sua avaliação.

## 6. CONCLUSÃO

No início deste trabalho foi mencionado como os dispositivos móveis estão entrando no dia-a-dia das pessoas, e, como pouco a pouco *gadgets* como *tablets* e *smartphones* vão ganhando espaço entre as coisas que passamos a considerar indispensáveis.

ROGERS parece estar ainda mais certo hoje que à época da citada publicação quando afirma que “Android pode ser a grande próxima oportunidade para desenvolvedores de software” (2009, p. 3).

Além dos desenvolvedores, é importante que as organizações dispensem mais atenção às novas possibilidades que a portabilidade da informática continuará criando à medida que os avanços tecnológicos ampliam as capacidades de processamento e armazenamento de dados de dispositivos móveis.

No particular de que este trabalho tratou, é importante frisar que seu foco esteve na realização de avaliações acadêmicas, no entanto, não é difícil vislumbrar como um suíte de aplicações nos moldes do que foi desenvolvido, um servidor e vários dispositivos móveis clientes, poderia também suportar atividades didáticas, por exemplo.

Acerca dos objetivos deste trabalho, pode-se afirmar que foram satisfatoriamente alcançados, ao passo que as aplicações Java Desktop e Android foram implementadas e puderam criar de forma bem sucedida a comunicação de dados entre si por meio do uso de uma rede WiFi através de XML e persistindo dados em SQLite.

Entretanto, é importante citar que a criptografia dos dados, que era um objetivo específico, e indispensável para ajudar a garantir a inviolabilidade no tráfego de dados sobre uma rede de computadores, não foi implementada.

Uma percepção obtida durante a produção deste trabalho foi a de que muitas organizações já começaram a considerar os benefícios dos dispositivos móveis. Recentemente, o Banco do Brasil (2011) lançou um aplicativo para Android. É óbvio que as transações realizadas pelos dispositivos móveis, a exemplo do que ocorre pela internet, irão cooperar para desafogar suas agências bancárias.

Os maiores benefícios aparecem quando as organizações se ‘apropriam’ do poder desses dispositivos para ‘trabalharem’ em seu favor, quando pensam nesses dispositivos como

extensão de seus ativos. Essa foi a proposta deste trabalho, tentar apontar para os benefícios desta possibilidade.

Pode ser que ainda leve alguns anos até que dispositivos móveis ganhem mais espaço na interação que as organizações realizam com as pessoas, mas não deixa de ser empolgante imaginar que, muito provavelmente, logo será possível interagir mais, através de seu dispositivo móvel, com as organizações com as quais lida, não apenas conectando-o à rede WiFi pública para navegar na internet, mas para escolher sua pizza em uma pizzeria, por exemplo. É obvio que os ilustres garçons não precisam desaparecer, mas, com a devida licença para concluir de forma amistosa, será mais difícil vir algo que não foi pedido.

## REFERÊNCIAS

ALLEN, Grant; OWENS, Mike. **The Definitive Guide to SQLite**. 2ed. New York: Apress, 2010.

ANGELIM, Rogério. **Pesquisa Aponta Android OS na liderança de venda recentes**. Disponível em: <<http://rogerioangelim.wordpress.com/2011/01/03/pesquisa-aponta-android-os-na-lideranca-de-vendas-recentes/>>. Acesso em 28 de junho de 2011.

BANCO DO BRASIL. **BB Lança Aplicativo para Android**. Disponível em: <<http://www.bb.com.br/portalbb/page251,138,5504,0,0,1,0.bb?codigoMenu=0&codigoNoticia=27341>>. Acesso em 05 de setembro de 2011.

CONDER, Shane; DARCEY, Lauren. **Android Wireless Application Development**. Boston: Pearson Education, 2010.

DAUM, Berthold; MERTEN, Udo. **System Architecture with XML**. San Francisco: Elsevier, 2003.

FLANAGAN, David. **Java In a Nutshell**. 3 ed. Sebastopol: O'Reilly, 1999.

FOLHA de São Paulo. **Primeira Transmissão de Rádio no Brasil faz 80 anos**. Disponível em: <<http://www1.folha.uol.com.br/folha/informatica/ult124u11128.shtml>>. Acesso em 01 de julho de 2011.

GOOGLE INCORPORATION. **Bi-directional Code Generation**. Disponível em: <<http://code.google.com/intl/pt-BR/javadevtools/wbpro/features/bidirectional.html>>. Acesso em 04 de julho de 2011c.

GOOGLE INCORPORATION. **Framework Topics: Activities**. Disponível em: <<http://developer.android.com/guide/topics/fundamentals/activities.html>>. Acesso em 29 de junho de 2011b.

GOOGLE INCORPORATION. **Painless Threading**. Disponível em: <<http://developer.android.com/resources/articles/painless-threading.html>>. Acesso em 25 de agosto de 2011d.

GOOGLE INCORPORATION. **What is Android?** Disponível em: <<http://developer.android.com/guide/basics/what-is-android.html>>. Acesso em 14 de junho de 2011a.

KREIBICH, Jay A. **Using SQLite**. Sebastopol: O'Reilly, 2010.

MAX, Richard. **IDC mostra suas previsões para o mercado smartphone de 2015**. Disponível em: <<http://www.ngabr.com.br/richard/1691-idc-mostra-suas-previsoes-para-o-mercado-smartphone-de-2015>>. Acesso em 28 de junho de 2011.

OAKS, Scott. **Java Security**. 2ed. Sebastopol: O'Reilly, 2001.

ROGERS, Rick *et al.* **Android Application Development**. Sebastopol: O'Reilly, 2009.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. **The Unified Modeling Language Reference Manual**. Massachusetts: Addison, 1999.

SERPRO – Serviço Federal de Processamento de Dados. **Governo Incentiva a Produção Nacional de Tablets**. Disponível em: <<http://www.serpro.gov.br/noticiasSERPRO/2011/maio/governo-incentiva-a-producao-nacional-de-tablets>>. Acesso em 15 de junho de 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 8a. ed. Pearson Education, 2007.

SQLITE ORG. **Frequently Asked Questions**. Disponível em: <<http://www.sqlite.org/faq.html#q19>>. Acesso em 11 de julho de 2011.

Universidade Estácio de Sá. **Universidade Estácio de Sá Distribui Tablets aos Alunos**. Disponível em: <<http://www.portaluniversidade.com.br/noticias-ler/universidade-estacio-de-sa-distribui-tablets-aos-alunos/1360>>. Acesso em 15 de junho de 2011.

WIFI Alliance. **Organization**. Disponível em: <<http://www.wi-fi.org/organization.php>>. Acesso em: 01 de julho de 2011.

## GLOSSÁRIO

**Access Point:** Dispositivo capaz de permitir e gerenciar a conexão WiFi com outros dispositivos para o compartilhamento de arquivos ou recursos;

**Driver:** Software que cria uma interface de acesso aos recursos de um dispositivo físico ou outro software (e.g. banco de dados).

**Engine:** Refere-se ao conjunto dos processos que constroem o serviço de um Banco de Dados, como por exemplo, selecionar registros, atualizar ou apagar registros, gerenciar as conexões e etc.

**Features:** Qualidades funcionais de um software ou hardware, funções.

**Host:** No contexto da Máquina Virtual Java, é o computador no qual a mesma está sendo executada.

**OS:** Vem do inglês *Operating System*, e significa Sistema Operacional.

**Overhead:** Alto custo de processamento.

**Plugin:** Complemento instalável em um aplicativo que estende ou acrescenta uma funcionalidade.

**WiFi:** Padrão internacional de comunicação de dados sem fio.