UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE INFORMÁTICA

ANDRÉ LUIS CORDEIRO DE FARIA

CLÉCIUS JOSÉ MARTINKOSKI

IMPLEMENTAÇÃO DE WEB SERVICE PARA DISTRIBUIÇÃO DE E-BOOKS E SEU CLIENTE ANDROID

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2012

ANDRÉ LUIS CORDEIRO DE FARIA

CLÉCIUS JOSÉ MARTINKOSKI

IMPLEMENTAÇÃO DE WEB SERVICE PARA DISTRIBUIÇÃO DE E-BOOKS E SEU CLIENTE ANDROID

Trabalho de Conclusão de Curso de graduação da Universidade Tecnológica Federal do Paraná campus Ponta Grossa, apresentado à disciplina de Trabalho de Conclusão de Curso como requisito parcial a obtenção do título de Tecnólogo em Analise e Desenvolvimento de Sistemas.

Orientador: Prof. Cristian Cosmoski Rangel de Abreu

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DE WEB SERVICE PARA DISTRIBUIÇÃO DE E-BOOKS E SEU CLIENTE ANDROID

por

ANDRÉ LUIS CORDEIRO DE FARIA
CLÉCIUS JOSÉ MARTINKOSKI

Este Trabalho de Conclusão de Curso (TCC) ou esta Monografia ou esta Dissertação foi apresentado(a) em 05 de junho de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. Os(as) candidatos(as) foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Cristian Cosmoski Rangel de Abreu
Prof.(a) Orientador(a)

Wellton Costa de Oliveira
Membro titular

Alison Roger Hajo Weber
Membro titular

Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

Simone de Almeida
Coordenador(a) do Curso
UTFPR - Campus Ponta Grossa

AGRADECIMENTOS

Eu André Luis Cordeiro de Faria gostaria de agradecer aos meus pais Dayton e Tereza que me orientaram durante minha vida e que proporcionaram a oportunidade estudar e consequentemente chegar aonde estou. Gostaria de agradecer também aos meus irmãos Marcia Luciane e Marcio Fernando que sempre me incentivaram nos momentos de dificuldade e me aconselharam. Aos meus amigos que sempre foram sinceros e com quem pude dividir os bons momentos desta jornada. À Bruna pelo companheirismo e que foi importante em um momento de grande dificuldade.

Eu Clécius José Martinkoski agradeço aos meus pais por estarem presentes me dando força, incentivo e sabedoria durante todo o tempo. Agradeço aos familiares que foram presentes e que me deram conselhos e apoio. Aos meus amigos com quem divido mais esta vitória.

RESUMO

FARIA, André Luis Cordeiro; MARTINKOSKI, Clécius José. **Implementação de Web Service para Distribuição de E-Books e seu Cliente Android.** 2012. 52. Trabalho de Conclusão de Curso, Tecnologia em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

O objetivo deste trabalho é o levantamento de tecnologias que podem auxiliar o desenvolvimento de um software para Android, e implementar um cliente em Android que acesse um Web Service de distribuição de E-Books para realizar a leitura, este cliente utiliza base de dados orientada a objeto, para auxilio na injeção de contextos e dependências é utilizado o framework RoboGuice, trechos de código são exibidos como exemplo para melhor entendimento, propriedades importantes da linguagem e projeto também são exploradas. A implementação do Web Service também é feita, utilizando a linguagem Java na versão J2EE com a base de dados db4o. Ao final deste são apresentados resultados do desenvolvimento como de *performance* e usabilidade no cliente e servidor, com justificativa e comparação das escolhas das tecnologias utilizadas.

Palavras-chave: Android. *Web Service*. *E-Book*. Base de Dados Orientada a Objeto.

ABSTRACT

FARIA, André Luis Cordeiro; MARTINKOSKI, Clécius José. **Implementing of Web Service for Distribution of E-Books and its Client for Android**. 2012. 52. Course Conclusion Work, Analysis and Systems Development Technology – Federal Technological University of Paraná, Ponta Grossa, 2012.

The objective of this study is a technology mapping that can assist the development of a software for Android and implement a client in Android that accesses a Web Service of E-Books distribution to make the reading, this client uses data base object oriented, to aid in the injection of contexts and dependencies is used RoboGuice framework, excerpts of code are displayed as an example for better understanding, important properties of language and project are also explored. The implementation of the Web Service is also made, using the Java language version J2EE with db4o data base. At the end of this study the development results are presented like performance and usability in client and web service, with justification and comparison of the choices used.

Keywords: Android. Web Service. E-Book, Object Oriented Database.

LISTA DE ILUSTRAÇÕES

Figura 1 – Plataformas Android e respectivas APIs Level.....14
Figura 2 – Emulador do Android plataforma 2.2.....16
Figura 3 – Ciclo de vida de uma Activity 21
Figura 4 – SQLite Personal Expert 26
Figura 5 – Tabela comparativa de base de dados orientada a objetos.....28
Figura 6 – Arquitetura de um Web Service 30
Figura 7 – Exemplo de texto em xml.....31
Figura 8 – Pagina inicial da ZBRA Solutions.....34
Figura 9 – Estrutura de pesquisa e desenvolvimento.....37
Figura 10 – Tela de entrada do aplicativo 40
Figura 11 – Tela de login e cadastro.....41
Figura 12 – Lista de livros no celular.....42
Figura 13 – Lista dos E-Books presentes no Web Service. 43
Figura 14 – Exemplo de busca de E-Book. 44
Figura 15 – Erro ao tentar conectar o Web Service 45
Figura 16 – Carregamento dos livros do Web Service 46
Figura 17 – Comparação de desempenho SGBDs 50
Figura 18 – Exemplo de diferença entre os modelos de persistência 52

LISTA DE QUADROS

Quadro 1 – Exemplo parâmetro para Intent 20
Quadro 2 – Exemplo de declaração de Activity 20
Quadro 3 – Passagem de parâmetro para o construtor da classe Carro 23
Quadro 4 – Construtor da classe Carro que recebe um objeto do tipo Context 23
Quadro 5 – Declaração de atributo com RoboGuice 24
Quadro 6 – Declaração da classe que recebe o contexto 24
Quadro 7 – Declaração de importação do layout xml 24
Quadro 8 – Exemplo função delete da classe SQLiteDatabase 25
Quadro 9 – Caminho da base de dados no emulador 25
Quadro 10 – Exemplo de abertura ou criação da base de dados 26
Quadro 11 – Exemplo de insert com SQLite 51
Quadro 12 – Exemplo de insert com db4o 52
Quadro 13 – Instanciando objeto Fonte 53
Quadro 14 – Delimitação política de instanciações Fonte 53

LISTA DE ABREVIATURAS E SIGLAS

ADT	Android Development Tools (Ferramentas de Desenvolvimento Android)
API	Application Programming Interface (Interface de Programação de Aplicações)
AVD	Android Virtual Device (Dispositivo Virtual Android)
CDI	Context injection componentes (Injeção de Componentes de Contexto)
CSS	Cascading Style Sheets (Folhas de Estilo em Cascata)
DAO	Data Access Object
DB4O	Database for objects (Base de Dados para Objetos)
E-Book	Electronic-BOOK (Livro Eletronico)
EPUB	Eletronic Publication (Publicação Eletronica)
GPS	Global Positioning System (Geo-posicionamento por Satélite)
HTML	HyperText Markup Language (Linguagem de Marcação de Hiper Texto)
IDE	Integrated Development Environment (Ambiente Integrado de Desenvolvimento)
JAX-WS	Java API for XML Web Services (API Java para Serviços Web XML)
JSE	Java Standart Edition (Java Edição Padrão)
MVC	Model, View e Controller (Modelo, Visão, Controle)
RPC	Remote Procedure Call (Chamada de Procedimento Remoto)
SDK	Software Development Kit (Kit de Desenvolvimento de Software)
SGBDOO	Sistemas de Gerenciamento de Banco de Dados Orientados a Objetos
SOAP	Simple Object Access Protocol (Protocolo de Acesso Simples ao Objeto)
SQL	Structured Query Language (Linguagem Estruturada de Consultas)
SWT	Standard Widget Toolkit (Kit de Ferramentas Widget Padrão)
WSDL	Web Services Description Language (Linguagem de Descrição de Serviços Web)
XHTML	eXtensible Hypertext Markup Language (Linguagem de Marcação de Hiper Texto Extensível)
XML	eXtensible Markup Language (Linguagem de Marcação Extensível)
XTEA	eXtend Tiny Encryption Algorithm (Algoritmo extendido de criptografia minúscula)

SUMARIO

1 INTRODUÇÃO 10
1.1 JUSTIFICATIVA 10
1.2 PROBLEMA 11
1.3 OBJETIVOS 11
1.3.1 Objetivo geral 11
1.3.2 Objetivos específicos 12
2 REFERENCIAL TEÓRICO 13
2.1 CONCEITOS 13
2.1.1 ANDROID SDK 13
2.1.2 Eclipse 17
2.1.2.1 Adt Pug-In 18
2.1.3 Android 19
2.1.3.1 RoboGuice 22
2.1.4 SQLite 24
2.1.5 Banco de dados orientado a objetos 26
2.1.6 Web Services 29
2.1.6.1 XML 30
2.1.6.2 SOAP 31
2.1.7 EPub 32
2.2 TRABALHOS CORRELATOS 33
2.2.1 Zbra 34
2.2.2 Google Play 35
3 MATERIAIS E MÉTODOS 37
3.1 ESTRUTURA DE PESQUISA E DESENVOLVIMENTO 37
4 CASO DE USO 39
4.1 FUNCIONAMENTO 39
5 RESULTADOS 47
5.1 SERVIDOR 47
5.2 CLIENTE 48
5.2.1 Plataforma Android 48
5.2.2 Persistência 49
5.2.3 RoboGuice 52
6 CONCLUSÃO 54
6.1 TRABALHOS FUTUROS 55

1 INTRODUÇÃO

Considerando o contexto social e ecológico atual, temos que relevar a importância de rever alguns costumes cotidianos. A leitura de um livro impresso é um exemplo clássico de um processo que existe há muito tempo sem sofrer alterações. Considerando o âmbito ecológico atual e o nível de desenvolvimento tecnológico exemplificado pela computação móvel, podemos revisar esse processo. O conceito de E-Books permite uma consciência ecológica e que condiz com a agilidade dos tempos atuais.

Dentro da nossa realidade atual em que pessoas têm o tempo como um bem escasso, tanto no trabalho quanto em sua vida pessoal, processos como a leitura de um livro que se subdivide em processos como a busca do título desejado em bibliotecas ou livrarias, exige tempo para o deslocamento e investigação.

Visto que a internet é uma importante ferramenta para democratização do conhecimento e redução de fronteiras e com tendências a computação nas nuvens, além de atentar à crescente popularização dos *tablets* e *smartphones* que são o ápice da mobilidade e interatividade e sendo estes cada vez mais presentes na vida das pessoas, devemos considerar que o papel como agregador de informações tem possibilidade de ser descontinuado. Portanto a nova tendência é que livros em papel se tornem uma lembrança e suas informações sejam disponibilizadas na “nuvem” (internet), sendo acessados por qualquer pessoa em qualquer lugar do mundo, ocupando apenas o espaço do dispositivo usado. Também se evidencia o fato supracitado pela crescente consciência ecológica e preocupação com o meio ambiente.

1.1 JUSTIFICATIVA

O objetivo de se desenvolver um sistema cliente-servidor de distribuição e leitura de livros digitais é a crescente tendência deste mercado.

Os usuários comuns buscam cada vez mais dispositivos que proporcionem mais recursos e o mercado corporativo cresce também buscando incorporar aplicações móveis para agilizar os negócios. Segundo Lecheta (2010), estudos apontam que cerca de 3 bilhões de pessoas possuem um celular, o que corresponde

a mais ou menos metade da população mundial.

A utilização de uma aplicação cliente desenvolvido em plataforma móvel se justifica pela crescente tendência de utilização de dispositivos móveis como *smartphones* e *tablets*, sendo estes importantes canais de distribuição de dados e plataforma propícia para a leitura de um livro, devido ao alto grau de mobilidade e interatividade.

Devido à capacidade inferior dos dispositivos móveis com relação ao processamento dos dados por possuir um tamanho físico reduzido, uma aplicação cliente que apresente apenas uma interface agradável ao usuário sem que esse exija do dispositivo um grande uso do processador torna se uma solução viável, para isso pode ser feita a comunicação do cliente ao *web service*.

Segundo McllRaith (2001) a rápida evolução da *Web* e a criação dos *web services* estão transformando a *Web* de um simples repositório de texto e imagens para um ambiente provedor de serviços.

De forma que a utilização de *web service* no lado servidor se justifica principalmente por ser uma plataforma consolidada no encapsulamento e distribuição de dados, além do alto nível de portabilidade possibilitando a diversas plataformas acessarem os dados.

1.2 PROBLEMA

Como desenvolver uma aplicação cliente em Android que se conecta a um *web service* para pesquisa, *download* e leitura de *E-Books*, contribuindo com o meio ambiente e agregando tecnologias para auxilio no desenvolvimento.

1.3 OBJETIVOS

1.3.1 Objetivo geral

Desenvolver um leitor de livros digitais, sendo este um cliente em Android que acessa um *web service* de distribuição de *E-Books* em J2EE.

1.3.2 Objetivos específicos

a) Levantamento de informações para desenvolvimento de dispositivos móveis utilizando o Android SDK.

b) Levantamento de informações para utilizar sistema gerenciador de banco de dados orientado a objetos db4o.

c) Levantamento dos padrões e utilização de *web services*.

d) Implementação de um cliente em Android com base de dados orientada a objeto.

e) Implementação de um *web service* em J2EE com base de dados orientada a objeto.

f) Utilizar *framework* de injeção de contextos e dependências no desenvolvimento.

2 REFERENCIAL TEÓRICO

Desenvolver um *software* não é uma tarefa fácil e pode tomar muito tempo das equipes de programação. Outro fator de dificuldade durante o processo de desenvolvimento é a linguagem a ser usada e em quais plataformas a linguagem que o *software* foi desenvolvido irá suportar.

Para evitar que o código de um *software* tenha que ser desenvolvido em varias linguagens seja ela para *web*, *desktop* ou dispositivos móveis, o uso de um *web service* se torna viável, possibilitando juntar essa tecnologia a um cliente desenvolvido para Android com uma base de dados orientada a objetos contribuindo ainda mais para a produtividade.

Nesta seção serão abordados os conceitos que foram utilizados para pesquisa e desenvolvimento do trabalho. Subseção 2.1.1 Android SDK, subseção 2.1.2 ambiente de desenvolvimento Eclipse IDE e o plug-in ADT, subseção 2.1.3 plataforma Android, subseção 2.1.4 o banco de dados SQLite, subseção 2.1.5 banco de dados orientado a objetos, subseção 2.1.6 web services.

2.1 CONCEITOS

2.1.1 ANDROID SDK

Para iniciar o desenvolvimento de um *software* para o sistema operacional Android é necessária a instalação de um emulador onde é possível testar as aplicações que forem desenvolvidas. O Android SDK é a ferramenta que possibilita os testes, ele possui um emulador que simula um celular com todas as versões do sistema Android existentes, possui também ferramentas e classes prontas que são necessárias para o desenvolvimento, ele deve ser baixado, instalado e configurado.

O endereço para o download do arquivo é http://developer.android.com/sdk/, após o termino de *download* do Android SDK é necessária a instalação do mesmo desde que atenda aos requisitos de sistema operacional. O SDK do Android depois de instalado contém várias aplicações prontas de podem ser encontradas dentro de suas pastas no diretório do computador instalado, são aplicações para uso didático que tem como utilidade demonstrar recursos do sistema Android, demonstrar uso de

tecnologias como *bluetooth*, o uso de classes do Android, uso do banco de dados SQLite, uso de interfaces gráficas com tratamento de resolução de tela entre outros exemplos.

No mercado existem muitos celulares e eles podem conter versões diferentes do sistema operacional, no Android cada uma dessas versões existentes são chamadas de plataforma, então é possível dizer que o Android contém muitas plataformas diferentes.

Essas plataformas estão ligadas ao termo *API Level*, este termo identifica qual plataforma do Android é utilizada, pois quando se deseja desenvolver aplicativos para Android também é necessário saber qual será o celular ou *smartphone* alvo. Por exemplo: Supondo que será desenvolvido um aplicativo para a plataforma 2.0 então a *API Level* mínima será 5, conforme novas versões de plataforma surgem, o numero da *API Level* também aumenta (LECHETA, 2010).

Abaixo a Figura 1 mostra as *API Levels* que existem atualmente e as seguintes plataformas do Android correspondentes, sendo a primeira coluna a versão da plataforma e a segunda coluna a *API Level*.

Versão da Plataforma	Level da API
Android 4.0.3	15
Android 4.0, 4.0.1, 4.0.2	14
Android 3.2	13
Android 3.1.x	12
Android 3.0.x	11
Android 2.3.4 Android 2.3.3	10
Android 2.3.2 Android 2.3.1 Android 2.3	9
Android 2.2.x	8
Android 2.1.x	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Figura 1 – Plataformas Android e respectivas APIs Level
fonte: Adaptado de Android Developers (2012)

Para emular o aplicativo desenvolvido, é necessária a criação de uma AVD (*Android Virtual Device*) que é um aplicativo que cria uma máquina virtual exatamente como é o celular alvo. Essa AVD tem como objetivo simular um celular real, tendo as mesmas configurações e resolução de tela, a criação da *AVD* pode ser feita por *prompt* de comando ou pela ferramenta visual, que é mais produtiva (LECHETA, 2010).

Este emulador criado consegue simular o *software* e o *hardware* de um celular, tem a capacidade de executar músicas, vídeos, armazenamento de dados através do banco de dados SQLite que será citado posteriormente, fazer a chamada de outros aplicativos dentro dele, acesso a rede, sendo assim consegue-se navegar na internet através do navegador que já vem junto com o emulador, e é possível também simular uma chamada telefônica via o *prompt* de comando do sistema operacional (SILVA, 2009).

Abaixo a Figura 2 exibe como é um emulador na plataforma 2.2 do Android com *API Level* 8.

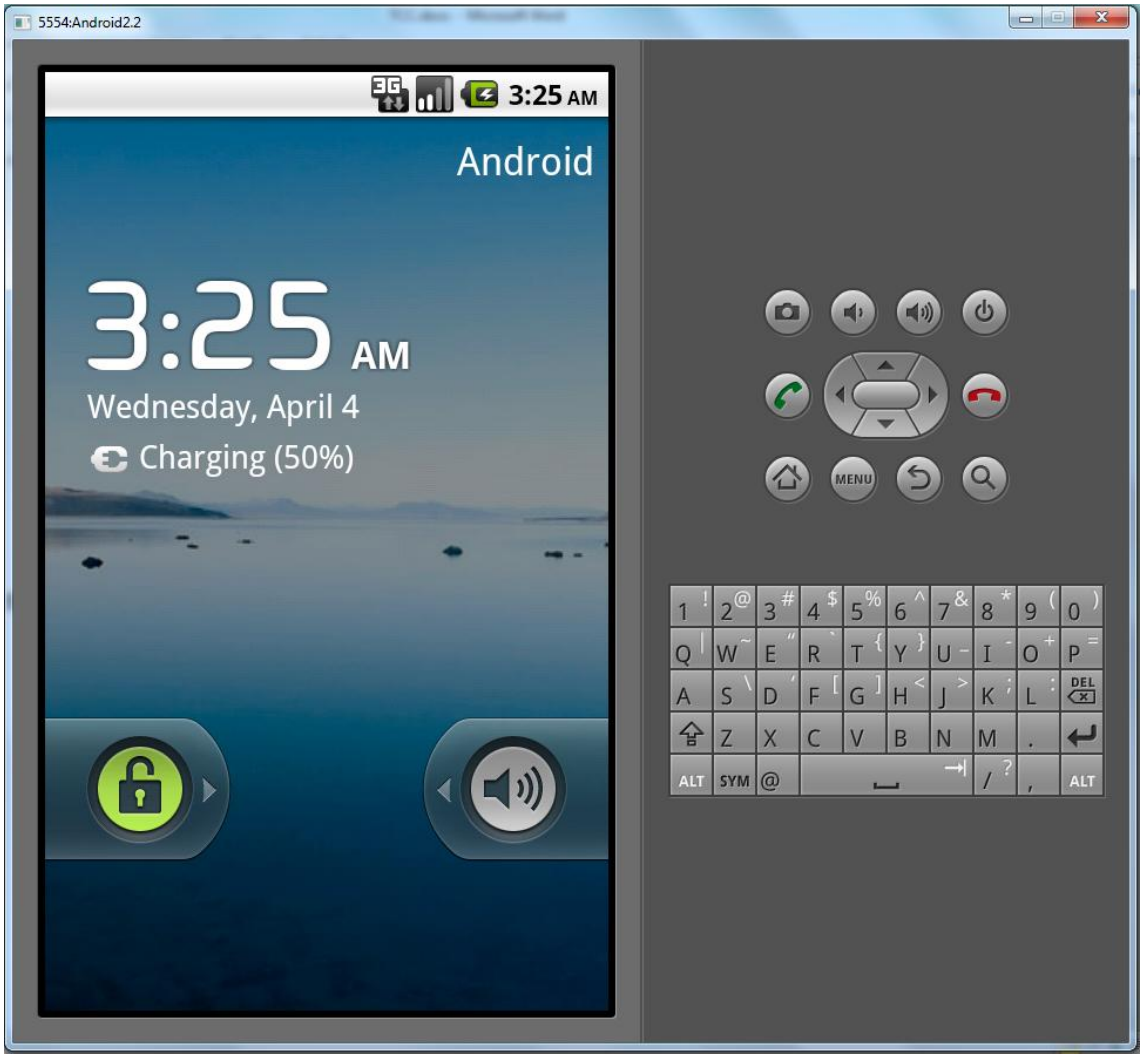

Figura 2 – Emulador do Android plataforma 2.2
Fonte: (Autoria Própria)

Para Lecheta (2010) é muito importante ficar atento ao fato de que se for utilizada uma *API* existente em plataformas recentes, a aplicação pode não funcionar em sua totalidade em celulares que não contenham tal plataforma instalada, ela pode ser executada parcialmente pois os desenvolvedores procuram criar o máximo de compatibilidade entre as plataformas, o que pode acontecer é caso o desenvolvedor utilize um recurso recente que não exista na plataforma mais antiga, então este recurso não irá funcionar.

2.1.2 Eclipse

O Eclipse IDE é uma ferramenta de desenvolvimento de *software* sendo ela uma das mais populares entre os programadores, ele permite o desenvolvimento de *software* na linguagem Java.

O Eclipse foi desenvolvido por uma empresa americana chamada IBM, ele é um dos *IDE's* mais utilizados no mundo em especial por utilizar a biblioteca gráfica *SWT* que será citada mais adiante (SUN, 2005 apud LIMA *et al.,* 2005).

Dentre as varias características do Eclipse pode ser citadas geração automática de código, ferramentas que gerenciam o trabalho coletivo, projetos compilados em tempo real, fácil visualização e acesso aos arquivos do projeto. O fato do Eclipse apresentar como característica o uso de plug-ins permite que o ambiente de desenvolvimento seja personalizado de acordo com o objetivo do desenvolvedor, os plug-ins também podem ser responsáveis pela criação de outros plug-ins (LIMA *et al.,*2005).

O Eclipse é portátil portanto sua aplicação é multi-plataforma e funciona em vários ambientes e não está restrito apenas a linguagem de programação Java, linguagens como C/C++ também podem ser codificadas, a única restrição é a instalação *plug-ins*, é um projeto livre de patentes.

A interface de um sistema operacional é composta por vários componentes, entre eles incluem menus, botões, janelas etc., o eclipse utiliza uma plataforma chamada *SWT* permitindo ao desenvolvedor criar aplicações gráficas, essa plataforma agiliza o trabalho do programador fazendo com que tal tenha acesso direto aos componentes que desenham a tela, assim ele pode arrasta-los e posiciona-los da maneira desejada de modo ágil (LOZANO, edição 31 apud FARIA, 2010).

Quando é criado o primeiro projeto Android no Eclipse, após instalação do *plug-in ADT* nos deparamos com a estrutura do projeto que foi criado, essa estrutura apresenta várias pastas com arquivos, e que devem ser entendidos para o desenvolvimento de aplicações e que são em sequencia:

- *src* é a pasta onde estão as classes java;

- *gen* pasta onde está o arquivo R.java, este arquivo nunca deve ser alterado manualmente, é uma classe gerada automaticamente pelo ambiente de

desenvolvimento, essa classe permite acesso a qualquer recurso do projeto como *layouts* de telas, imagens pois são geradas constantes;

- *assets* estão os arquivos opcionais ao projeto.

- res essa pasta é dividida em três subpastas (*drawable, layout, values*), é onde ficam arquivos com a aparência da tela e imagens usadas no projeto;

- *drawable* é a pasta onde estão contidas as imagens do projeto, porém como são diversos os celulares com o Android, é permitida a personalização das imagens para que cada uma tenha o tamanho de cada resolução de celular diferente. Existem três subpastas dentro desta pasta que dividem as imagens (*drawable-ldpi, drawable-mdpi, drawable-hdpi*);

- *layout* é onde estão arquivos de extensão .xml, são arquivos responsáveis por construir as telas do aplicativo;

- *values* estão arquivos de internacionalização da aplicação e algumas outras configurações;

Os arquivos que estão contidos na pasta *res* são referenciados na classe R da pasta gen citada acima, que são arquivos como os de imagens e *layout* de telas, quando um *layout* é alterado, o nome que foi dado a um determinado componente (botões, campos de texto, imagens, etc.) ou se um componente for excluído do *layout*, as alterações são geradas automaticamente pelo *plug-in* ADT, desta forma não é preciso alterar manualmente os arquivos da classe R (LECHETA, 2010).

2.1.2.1ADT PLUG-IN

O nome *Plug-in* vem da ideia de plugar algo, ou seja, plugar, anexar ou instalar um software existente a um que já exista, isso permitirá estender as funções do ambiente que receber esse plug-in, aumentar a quantidade de recursos para uma atividade especifica (COMPOR apud LIMA *et al.*, 2005).

O ADT é um *plug-in* que deve ser instalado no ambiente de desenvolvimento Eclipse, esse *plug-in* é fornecido pelo Google. O *plug-in* ADT deve ser baixado, ele será responsável por permitir a criação de projetos Android no Eclipse (LECHETA, 2010).

2.1.3 Android

O Android é uma plataforma para desenvolvimento de aplicações que utiliza a linguagem de programação Java e permite que todos os recursos da linguagem sejam usufruídos, o Android possui um ambiente muito poderoso e flexível. Quando se diz que o Android possui um ambiente flexível, atribuímos o termo ao fato de que o menu, agenda de contatos entre outros recursos do celular ou *smartphone* podem ser customizados pelo programador, isso acontece porque o Android é uma plataforma gratuita e consegue integrar aplicações nativas do sistema como também aplicações desenvolvidas por um programador qualquer. Quanto ao termo “integrar”, esta se referindo as aplicações se juntarem, uma funcionalidade de um aplicativo pode precisar talvez em algum momento de uma funcionalidade que está em outro aplicativo do celular ou smartphone, a plataforma Android foi desenvolvida em cima dessa ideia (LECHETA, 2010).

.

Existe uma classe no Android que é a responsável por essa integração, a classe é chamada *Intent*. O nome *Intent* representa a intenção de que algo seja realizado. Algumas das funcionalidades da *Intent* são:

- Abrir uma nova tela no aplicativo;
- Fazer uma chamada telefônica utilizando um numero gravado na memoria do celular ou *smartphone*;
- Acessar o recurso de outro aplicativo;
- Enviar uma mensagem que pode ser captada por qualquer aplicativo.

A *Intent* recebe em seu construtor uma classe *Activity*, que representa uma tela que a aplicação tem a intenção de chamar, mas existe também um outro método construtor e nesse devemos colocar uma String que pode ser de escolha do desenvolvedor, ele irá adaptar essa String conforme o contexto do desenvolvimento, essa será a String que qualquer aplicativo do sistema Android poderá interpretar e chamar uma funcionalidade presente na mesma (LECHETA, 2010).

É possível exemplificar o que foi dito com o seguinte trecho de código, nele irá ser colocado uma String qualquer como parâmetro:

```
Intent it  = new Intent(“estudar”);
startActivity(it);
```

Quadro 1 – Exemplo parâmetro para Intent
Fonte: (Autoria Própria)

Foi definido o termo estudar na *Intent*, esse termo deve ser uma ação implementada pelo programador, mas já existem várias ações definidas pelo Android que podem ser usadas, basta conhece-las.

Varias vezes o termo *Activity* foi citado, ela é uma outra classe muito importante do Android, a *Activity* representa uma tela do aplicativo e é responsável também pelos eventos dessa tela, eventos são ações que o usuário pode tomar ao clicar na tela *touchscreen* ou se algum botão do teclado for pressionado, alguns eventos como selecionar um item em uma lista, pressionar um botão, marcar opções de escolha entre outras. Uma *Activity* deve ficar na pasta que contem os arquivos .java, quando uma classe é cria ela estende da classe *Activity*, porem para que projeto criado saiba que a classe criada será uma tela, ela deve ser declarada em um arquivo chamado AndroidManifest.xml, basta criar uma tag <activity>, o exemplo abaixo demonstra a forma correta que a tag deve ser inserida no código xml do arquivo AndroidManifest.xml (LECHETA, 2010).

```
<activity android:name=”ClasseActivity” />
```

Quadro 2 – Exemplo de declaração de Activity
Fonte: (Autoria Própria)

Segundo Aquino (2007) quando os componentes do projeto estão definidos, necessita-se declara-los no AndroidManifest.xml, pois ele contem os componentes da aplicação além de capacidade e requerimentos para esses componente.

Os aplicativos no momento da instalação no sistema Android pedem a permissão do usuário e podem informar quais recursos serão utilizados, ficando por

conta do usuário aceitar ou recusar a instalação do mesmo, o arquivo que contém as informações dos recursos do celular ou *smartphone* que serão utilizados é o AndroidManifest.xml (SIMÃO *et al., 2011*).

Uma *Activity* possui um ciclo de vida que deve ser intendido, o sistema operacional cuida deste ciclo, mas se o desenvolvedor deseja que sua aplicação seja mais robusta então elas tem de ser levadas em consideração. A Figura 3 exibe para uma melhor compreensão as chamadas dos métodos e os estados em que se encontrão:

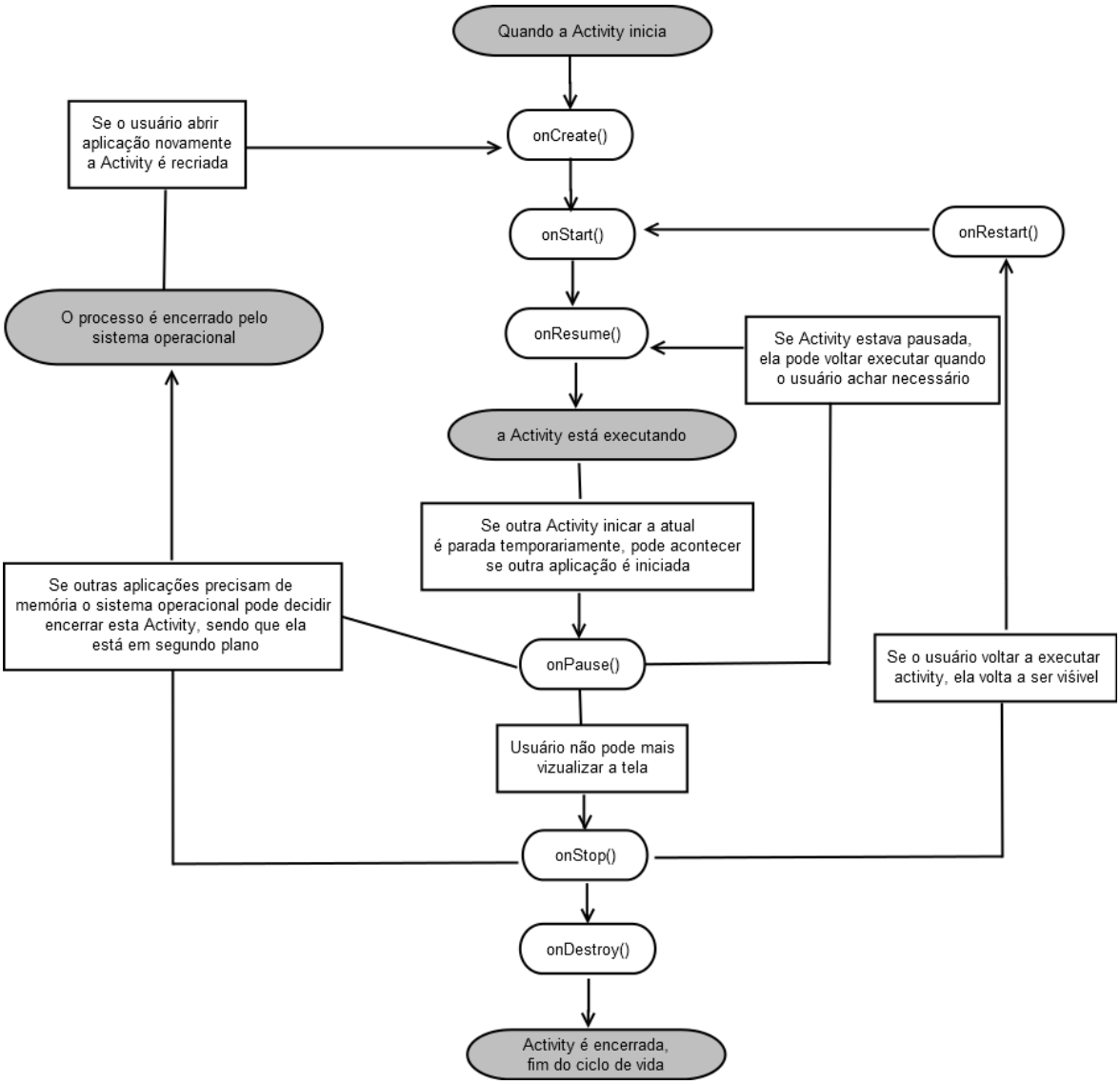

Figura 3 – Ciclo de vida de uma Activity
Fonte: Adaptado de Lecheta (2010)

Os métodos contidos no diagrama acima serão citados e descritos abaixo para uma compreensão mais detalhadas do controle do estado de uma aplicação:

a) *onCreate*(): É um método obrigatório em uma *Activity* que faz a chamada a um outro método chamado "*setContentView*()" para exibir na tela;

b) *onStart*(): Quando a *Activity* já contém uma *view* e está quase visível ao usuário, ele é chamado em duas ocasiões, após os métodos *onCreate*() ou *onRestart*(), esses métodos são invocados conforme o contexto do aplicativo;

c) *onRestart*(): Quando uma *Activity* for pausada por um tempo, este método faz a chamada do método *onStart* caso a *Activity* esteja sendo iniciada outra vez;

d) *onResume*(): É um método que sempre é invocado depois do método *onStart*(), este é o momento em que a aplicação está em execução;

e) *onPause*(): Caso ocorra alguma ação de um evento, e o aplicativo que está sendo executado no momento necessite ser interrompido, esse método é chamado, sendo assim ele salva o estado em que a aplicação está, para que quando esta tela for chamada novamente os dados que estavam nela sejam recuperados e exibido;

f) *onStop*(): Método chamado quando uma outra tela é chamada e a anterior não é mais vista pelo usuário do aplicativo, esta tela continua em memória caso precise ser recuperada, mas se o usuário não acessar a tela novamente durante um tempo longo o sistema pode encerrar essa a *Activity* fazendo a chamada do método *onDestroy*() para que os recursos sejam liberados, sendo assim a memória é liberada;

g) *onDestroy*(): É um método que pode ser invocado com o método *finish*() da classe *Activity*, ele exclui da memória a *Activity* em questão, caso a mesma *Activity* fique muito tempo sem que o usuário a chame o sistema operacional faz a chamada deste método para liberar recursos de memória (LECHETA, 2010).

2.1.3.1RoboGuice

Quando um aplicativo em Android está sendo desenvolvido, as questões a serem trabalhadas são diversas, uma delas é a classe *Context* em que os softwares são fortemente baseados.

A classe *Context* é muito utilizada para criar os componentes das *Views* (telas), quando uma *Activity* é chamada, as *Intents* devem receber como um dos parâmetros o contexto em questão, é utilizada também nos momentos que a base de dados deve ser acessada. Para que os programadores trabalhem com isso, é criada uma variável global que recebe o contexto e que muitas vezes é passada como parâmetro para outras classes, geralmente em seus construtores (CAELUM, 2012).

Abaixo é possível visualizar um exemplo de uma passagem de parâmetro do *Context* para uma classe, e a seguinte recebendo o parâmetro em seu construtor, para o exemplo será utilizado o nome Carro para a classe:

```
Carro carro = new Carro(this);
```

Quadro 3 – Passagem de parâmetro para o construtor da classe Carro
Fonte: (Autoria Própria)

Como parâmetro é passado "*this*", o que significa que está passando o contexto daquela *Activity*, ou senão é possível utilizar o comando *getApplicationContext*(), abaixo o construtor da classe Carro:

```
public Carro(Context ctx)
{
     this.ctx = ctx;
}
```

Quadro 4 – Construtor da classe Carro que recebe um objeto do tipo Context
Fonte: (Autoria Própria)

Essas *Activities* podem se tornar muito complexas e contendo muito código, para auxiliar nesta função que acaba sendo usada quase que sempre, é possível utilizar um framework chamado RoboGuice que auxilia na injeção de contextos e dependências, através do comando "@*injectResource*" a classe que iria receber em seu construtor o contexto como parâmetro apenas declara a variável como no exemplo abaixo já recebendo a *view*:

```
@InjectResource(R.string.cor) private String cor;
```

Quadro 5 – Declaração de atributo com RoboGuice
Fonte: (Autoria Própria)

A *Activity* que irá passar o *Context* para a classe deve extender da classe *RoboActivity* e utilizar o comando abaixo, assim tornando a *Activity* mais enxuta:

```
@Inject private Carro carro;
```

Quadro 6 – Declaração da classe que recebe o contexto
Fonte: (Autoria Própria)

É necessário também importar o *layout* da tela que está contido na pasta *res*, e para isso utiliza o comando:

```
@ContentView(R.layout.tela_carro);
```

Quadro 7 – Declaração de importação do layout xml
Fonte: (Autoria Própria)

Dividir as responsabilidades do programa é recomendado e possível, mesmo que o desenvolvedor não esteja utilizando o *framework RoboGuice*, porém com o uso do *framework* o desenvolvedor pode focar mais nas regras de negócios e na orientação a objetos, pois não precisará mais se preocupar com a injeção de dependências, já que o *RoboGuice* fica responsável por guardar essa instancia (CALEUM, 2012).

2.1.4 SQLite

Quando uma aplicação está sendo desenvolvida, é comum que um banco de dados seja usado para armazenar informações que são inseridas no sistema e controlar os dados. O Android possui integração com um banco de dados leve mas com grande performance, sendo assim é possível incluir um banco de dados para suas aplicações (LECHETA, 2010).

Utilizar um banco de dados é a forma mais comum para o armazenamento de informações, no Android isto se torna muito simples pois ele trabalha com um

sistema de chave e valor, se a deleção de um registro na base de dados é necessária, basta que se utilize o comando “*delete*” da classe SQLite Database como no exemplo abaixo:

```
int id = 5;
db.delete(“fruta”, “código = ” + id, null);
```

Quadro 8 – Exemplo função delete da classe SQLiteDatabase
Fonte: (Autoria Própria)

O código citado é equivalente a “delete from fruta where código = 5, a variável db deve receber a conexão com o banco antes que o método delete seja chamado, o aplicativo Android pode conter mais de um banco de dados, basta que eles estejam na pasta *databases* do pacote do projeto, que fica no caminho:

```
/data/data/pacote_do_projeto/databases/
```

Quadro 9 – Caminho da base de dados no emulador
Fonte: (Autoria Própria)

Esse acesso é feito pelo *file explorer* do ambiente de desenvolvimento Eclipse, o emulador deve estar rodando para que as pastas apareçam na aba *do file explorer*, a base de dados criada deve ser importada para o emulador.

É possível que tabelas do banco de dados sejam criadas e informações sejam preenchidas nas tabelas por código, um *script* pode ser executado porem também é possível que a base de dados e suas respectivas tabelas sejam criadas através de um aplicativo que facilita tal processo, o *SQLite Expert Personal* é uma aplicação gráfica para que o banco de dados seja construído.

Para abrir o banco de dados criado o código abaixo pode ser citado como exemplo, a variável db recebe a conexão como banco:

```
SQLiteDatabase bd =
ctx.openOrCreateDatabase("nome_do_banco",Context.MODE
_PRIVATE, null);
```

Quadro 10 – Exemplo de abertura ou criação da base de dados
Fonte: (Autoria Própria)

O primeiro parâmetro passado foi o nome do banco de dados que foi importado para a pasta *databases* do emulador, o segundo parâmetro indica o modo de abertura deste banco, o *Context.MODE_PRIVATE* indica que apenas este aplicativo pode abri-lo, e o terceiro parâmetro passamos um valor nulo e dificilmente esse valor será alterado, ele recebe um objeto do tipo *Cursor*, tendo a base de dados aberta é possível deleção, inserção e atualização dos registros. (LECHETA, 2010)

A Figura 4 abaixo mostra a aplicação gráfica para criação de bancos de dados *SQLite Personal Expert* em sua versão 3.4.7.2216:

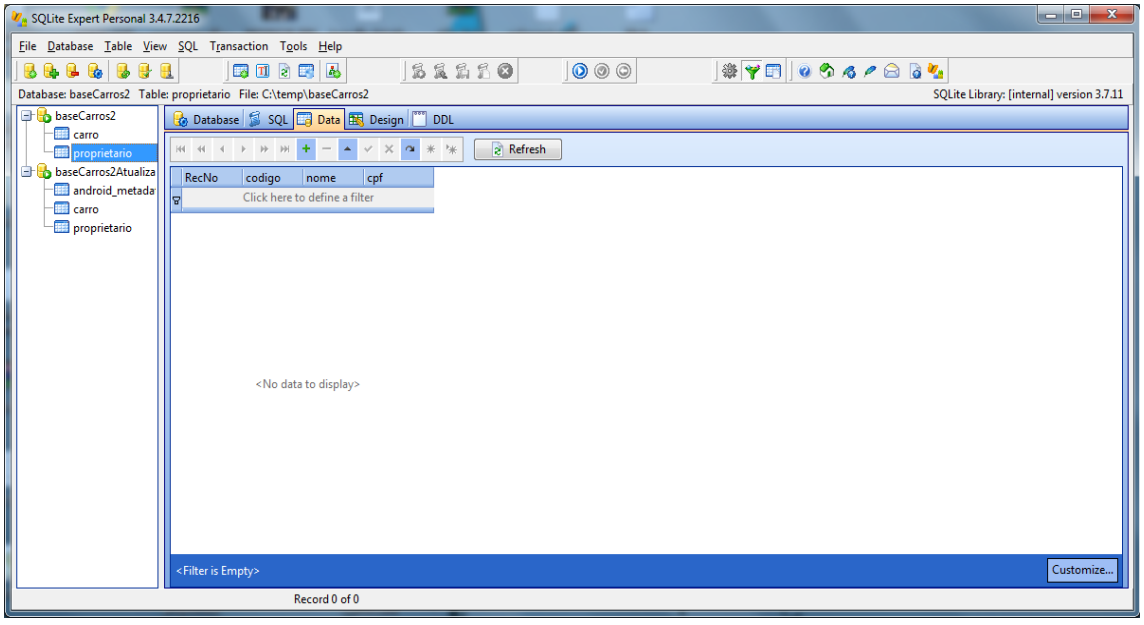

Figura 4 – SQLite Personal Expert
Fonte: (Autoria Própria)

2.1.5 Banco de dados orientado a objetos

Um banco de dados relacional trata os seus registros como colunas e linhas, diferente de um banco de dados orientado a objetos que trata os seus registros

como objetos, pois é desta forma que linguagens como C++, C# e Java veem os dados, em um banco de dados relacional junto a essas linguagens ocorre uma “tradução” das linhas e colunas em objetos, o que não acontece com um banco de dados orientado a objetos.

Existem três fatores que contribuem para a escolha de um banco de dados orientado a objetos: alta performance, necessidade de negócios e dados complexos (BARRY, 2009 apud Monteiro *et al.*, 2009).

Poupar e investir dinheiro estão relacionados a necessidade de negócios, quanto aos dados complexos podem existir uma quantidade muito grande de tabelas com muitos relacionamentos, ou seja relacionamentos do tipo muitos para muitos (m:n), um uso frequente dos atributos.

O db4o é um banco de dados orientado a objetos, ele tem como principal conceito o fato de sua persistência ser trabalhada via uma linguagem de programação.

Ele possui implementação em linguagem Java, assim como em outras linguagens também, o programador deve se prender somente a linguagem que ele conhece, no caso de se trabalhar com Java, basta conhecer apenas a linguagem Java, não sendo necessário ter o conhecimento da linguagem que o banco de dados possui.

Segundo Ledesma (2007), o db4o tem aceitação em muitos países do mundo, em cerca de 170 países é possível encontrar clientes que o utilizam, há também empresas lideres de mercado que o utilizam, o db4o de fato é marcado pela fácil utilização, possui uma programação simples da camada de persistência e tem como foco principal a produtividade de desenlvovimento.

Um banco de dados que é orientado a objetos tem como funcionalidade integrar a linguagem de programação de preferencia do desenvolvedor e integrar suas funcionalidades, o trabalho de persistência é toda baseada em objetos e são manipulados apenas pelos métodos das classes que eles pertencem (SANTANA et al., 2008).

Algumas particularidades a serem citadas a respeito do db4o é que ele pode ser instalado em cliente/ servidor quando *standalone*, oferece suporte a criptografia, utiliza a linguagem de programação Java e plataforma *.NET*, pode ser usado em um sistema embarcado, o que não o prende somente a aplicações *desktop* e pode ser

usado em qualquer tamanho de sistema desenvolvido, outra característica muito importante do db4o é o seu ótimo desempenho, a Figura 5 mostra a comparação entre alguns sistemas gerenciadores de banco de dados, incluindo o db4o.

Barcelona Benchmarks	Leitura	Escrita	Consulta	Exclusão
Native/db4o 6.4	1.0	1.0	1.0	1.0
Hibernate/hsqldb	15.8	3.7	2583.1	4.3
Hibernate/mysql	48.0	26.1	14.4	26.9
JDBC/MySQL	40.8	19.5	9.3	15.8
JDBC/JavaDB	27843.7	20.5	47993.1	17.7
JDBC/SQLite	8.8	519.1	1.1	2554.4

Figura 5 – Tabela comparativa de base de dados orientada a objetos
Fonte: (MONTEIRO, 2009)

O modelo de dados como objetos permite:

- Encapsulamento de dados e códigos

- Reutilização de código

- Polimorfismo

- Identificação única dos objetos

Algumas características que o banco de dados orientado a objetos db4o apresenta são:

- Acesso concorrente de vários usuários

- Gerenciar transações

- Conexão remota da base de dados

- Interfaces em várias linguagens

- Uso de varias *threads* no servidor da base de dados

A linguagem de consulta do db4o em sua versão 7.0 suporta algumas operações como as matemáticas por exemplo, suporta o operador *"IS EMPTY*" e suporta também os operadores "*EXISTS" e "FOR ALL*", permitindo ao desenvolvedor maiores possibilidades durante o seu trabalho com o SGBDOO (VERSANT, 2012).

Com relação a segurança para um banco de dados é um fator importante que deve ser relevado e o db4o neste ponto apresenta argumentos respeitáveis, ele

trata os seus arquivos em forma binária, nesse sistema gerenciador de banco de dados há um algoritmo que trata dessa segurança e é chamado XTEA, com arquivos que podem chegar a 254Gb de tamanho sem que ocorram problemas.

O XTEA (*eXtend Tiny Encryption Algorithm*) possui um sistema de criptografia que pode chegar aos 128bits, uma boa criptografia, e quando é dito que o db4o trata o armazenamento de arquivos em forma binária, significa que há a possibilidade de criptografar os registros da base de dados e para isso é feita a inserção de uma senha, mas não necessariamente isso é obrigatório, fica a respeito do programador, porém a questão segurança na maioria das vezes é indispensável (MONTEIRO *et al.*, 2009).

2.1.6 Web Services

Os *web wervices* surgiram como consequência do uso da internet e essa rede de computadores possibilitou que algo fosse criado para que aplicações fossem espalhadas de forma massiva. Os sistemas distribuídos vieram a funcionar de maneira que usuários comuns conseguiam através dos navegadores inserir, alterar e excluir informações.

Um fator de maior relevância no crescimento e uso de *web services* é o uso da linguagem XML para comunicação entre as aplicações, a linguagem XML permite uma maior integração, o protocolo SOAP é o responsável pelas trocas de mensagens e será visto mais detalhadamente adiante (BOOCH, 2005).

Os *web services* permitem a comunicação com outras aplicações, onde os seus método são invocados por essas aplicações, um dos benefícios do uso de *web services* é que o aplicativo cliente, que é o aplicativo que vai acessar os métodos, não precisa estar na mesma linguagem do aplicativo servidor, não precisa conhecer a plataforma ou a maneira de como o servidor foi desenvolvido.

Um *web service* possui como principais elementos as aplicações de cliente e servidor. A Figura 6 exibe a sua arquitetura detalhando a troca de mensagens e o protocolo citado de transporte: (HANSEN *et al.*, 2003)

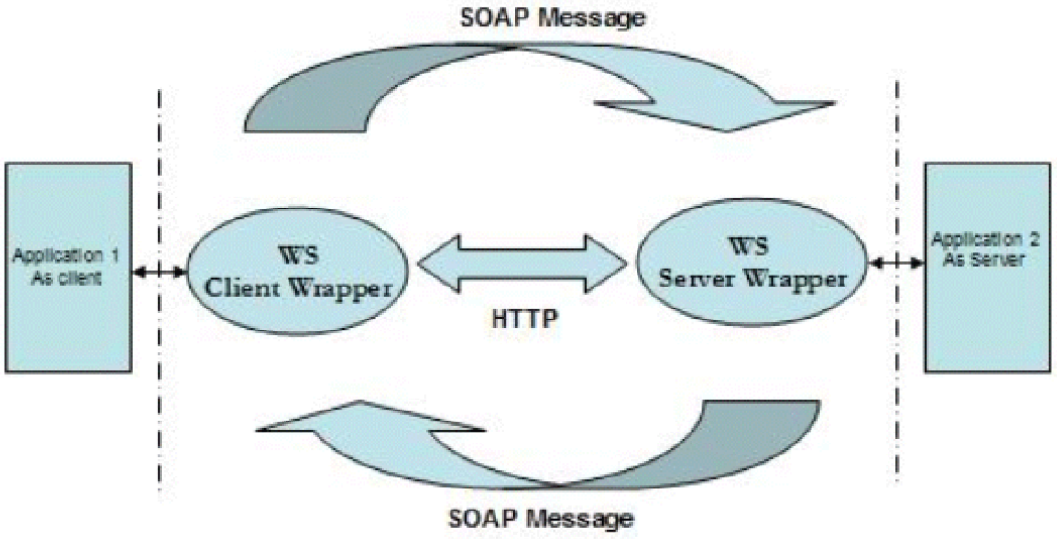

Figura 6 – Arquitetura de um Web Service
Fonte: (AGUIAR, 2008)

Um *web service* apresenta padrões que são considerados neutros, como o uso de XML como protocolo de transporte e protocolos de internet como mostrado na Figura 6, por isso torna-se tão fácil aplicar tal conceito em aplicações de dispositivos moveis, aplicações *desktop* ou aplicações *web* para exibirem dados (AGUIAR *et al.*, 2008).

2.1.6.1XML

O arquivo XML é reponsável por conter dados com as propriedades do *web service*, é um arquivo que possui o acesso publico e seus dados são escritos de acordo com a WSDL, sendo assim os clientes podem ter acesso e saber quais são os métodos oferecidos pelo web service (AGUIAR *et al.*, 2008).

Um XML é um arquivo que pode ser lido pelos desenvolvedores e manipulado pelos computadores, esses arquivos são manipulados por meio de tags, como o arquivo não apresenta uma formatação os computadores conseguem analisa-lo de forma sintática. O poder do XML o torna muito viável como middleware em aplicações cliente/servidor, o XML torna o trafego em uma rede muito mais rápido já que funções delimitadas ao servidores podem ser executadas por clientes com o uso deste arquivo.

Um arquivo XML é composto apenas por texto e tags, sendo assim editores de texto mais comuns como o bloco de notas do *Windows* por exemplo podem ser usados pelos desenvolvedores para cria-los. O XML tem como principais características:

- Independência de seus dados;

- O conteúdo do arquivo todo separado;

- A apresentação do conteúdo do arquivo; (DEITEL, 2001).

A Figura 7 apresenta um fragmento de uma pagina em XML como exemplo, mostrando a forma de hierarquia que é criada, o uso de tags e a separação do conteúdo, mostra também que não há formatação e apresenta apenas texto puro:

```
<mestrado>
        <descrição> Pessoas que estudam na UFMG </descrição>
        <turma>
                <pessoa>
                        <nome> João </nome>
                        <idade> 30 </idade>
                        <email> joao@ufmg.br </email>
                </pessoa>
                <pessoa>
                        <nome> Claudia </nome>
                        <idade> 25 </idade>
                        <email> claudia@ufmg.br </email>
                </pessoa>
                <pessoa>
                        <nome> Jose </nome>
                        <idade> 25 </idade>
                        <email> jose@ufmg.br </email>
                </pessoa>
        <turma>
<mestrado>
```

Figura 7 – Exemplo de texto em xml
Fonte: (ALMEIDA, 2002)

2.1.6.2SOAP

Quando um aplicativo cliente se comunica com um *web service* ele utiliza um protocolo chamado SOAP, este protocolo surge como meio de fazer a chamada de

métodos remotos que estão contidos nas aplicações dos servidores, chamados XML *Web Services* (AGUIAR *et al.*, 2008).

O nome do mecanismo que o SOAP é baseado é o RPC, para que sejam feitas chamadas remotas, as respostas dessas chamadas são no formato XML (FREIRE, 2002).

Segundo Freire (2002) o SOAP é o coração da arquitetura de *web services* apesar de não ser o único protocolo, esse protocolo permite por exemplo que buscas sejam feitas a partir de um determinado assunto.

Em uma aplicação para o sistema Android é necessário o uso de alguma biblioteca para criação de um *web service* pois o Android não possui uma API nativa. Como uma aplicação desenvolvida para o sistema Android irá executar em um dispositivo móvel, e este não possui a mesma capacidade de processamento de um computador comum, a biblioteca tem que ser leve e compacta, o *framework* Ksoap2 possui bibliotecas que são leves, é um *framework* projetado para dispositivos móveis com menos poder de processamento, o Ksoap2 deve ser baixado e incluído no projeto do ambiente de desenvolvimento Eclipse. É possível que o desenvolvedor através de código de programação faça a conexão com o *web service*, basta apenas que saiba a sintaxe do SOAP, porém o Ksoap2 é um *framework* consolidado e seu uso permite ao desenvolvedor ganho de tempo (LECHETA, 2010).

2.1.7 EPub

Para Honorato (2010) uma publicação digital que possua textos e figuras, que possam ser lidas por dispositivos móveis ou computadores, e que muitas vezes pode ser comparado a um titulo impresso pode ser denominado um *E-Book*, um *E-Book* pode se apresentar em vários formatos, ou seja arquivos de diferentes extensões.

O ePub é um dos tantos formatos existentes de livros digitais, ele apresenta como uma de suas principais características o fato de poder ser otimizado conforme o dispositivo que irá efetuar a leitura do mesmo, como o EPUB possui o código *open source* ou seja, código aberto, ele possui também como característica de grande importância poder ser lido em muitos dos sistemas operacionais existentes, inclusive sistemas para dispositivos móveis.

Um arquivo ePub possui como extensão de seus arquivos o .epub, este se apresenta como formato oficial para livros digitais tanto na sua distribuição quanto na venda, além de ser um formato mais compacto do que os outros existentes, essas características tornam o formato ePub mais compartilhável.

O formato do ePub e suas características apresentadas contribuem para que ele possa ser lido em tantas plataformas, sendo que tais possuem uma quantidade grande de aplicativos que efetuem a leitura do livro digital (EPUBR, 2012).

Uma página simples da internet possui um código html, para o *design* é criado um arquivo css que é uma folha de estilos e que apresenta como extensão .css, em um ePub o aquivo é feito em xhtml, basicamente os mesmos códigos usados em paginas da internet e que também é acompanhado de um arquivo css.

Segundo Melo (2010) a medida que o mercado evolui há um aperfeiçoamento dos *E-Books*, o mesmo deve ocorrer com o padrão ePub, este tornou-se o formato internacional de livros digitais por ser código aberto, e uma vantagem é a não necessidade de pagar *royalties* a uma empresa detentora dos direitos do formato, sendo este um dos fatores predominantes para que ele tenha se tornado um formato internacional.

O outro fator que o ePub possui é a quantidade de dispositivos, sistemas operacionais e a gama de aplicativos em que ele pode ser lido.

2.2 TRABALHOS CORRELATOS

Neste tópico serão apresentados alguns exemplos existentes a respeito dos conceitos que foram apresentados neste documento, quem os utilizam, e também artigo que apresenta estudos e explicações na mesma área de pesquisa e desenvolvimento.

Nesta área é possível encontrar empresas que trabalham com *web services* e programação Android utilizando-a para o desenvolvimento de clientes que acessem os *web services*.

2.2.1 Zbra

As empresas que desenvolvem soluções comerciais bem sucedidas para clientes que procuram competitividade no mercado geralmente trabalham com foco em várias plataformas. A Zbra Solutions é uma empresa localizada no estado de São Paulo - Brasil que atua não só no Brasil e também fora do país, é uma desenvolvedora de *software* que trabalha com desenvolvimento de *softwares* para celulares a Zbra Solutions possui um software de provisionamento de rede móvel e uma ferramenta de análise anti-fraude de operadores móveis.

A Zbra Solutions é um exemplo de uma empresa que atua com alguns dos conceitos citados acima, no caso o uso das tecnologias Android e *Web Service*, a Figura 8 exibe a página inicial da empresa.

Figura 8 – Pagina inicial da ZBRA Solutions
Fonte: Adaptado de ZBRA Solutions (2012)

A Figura 8 exibe a página principal da ZBRA Solutions e apresenta alguns de seus clientes e as soluções que são de sua posse, isto mostra que empresas não se limitam apenas ás que contém um espaço físico.

2.2.2 Google Play

Existem muitos sites na internet que atuam como empresas virtuais e disponibilizam para *download* aplicativos tanto para smartphones como para *tablets* também, o Google Play é um *web store* da Google, sendo esta a empresa desenvolvedora do Android e neste site é possível encontrar milhares de aplicativos para dispositivos Android, mas nem todas aplicações existentes no site são gratuitas e para adquiri-las é necessário comprar.

O acesso ao site permite o *download* do conteúdo desejado, seja a necessidade do usuário um jogo para entretenimento ou aplicativos, que podem vir a auxiliar hábitos cotidiano (GOOGLEPLAY, 2012).

Os estudos realizados por pessoas que se propõe a escrever documentos, auxiliam as que estão iniciando em tal área a se familiarizarem com os paradigmas da tecnologia, seja por meio de explicações escritas ou exemplos de diagramas, fluxogramas ou códigos.

Rabello (2007) apresenta um artigo sobre o desenvolvimento de um cliente para o sistema Android, este tem por objetivo acessar um web service para realizar consultas de CEPs, no artigo o cliente é integrado ao Google Maps para que a aplicação possua uma aparência mais profissional.

O artigo do autor citado acima exibe passo a passo como consumir um *web service*, para tal ele utiliza a biblioteca KSOAP2 também utilizada no desenvolvimento deste trabalho. O código da aplicação é exibido no artigo e o usuário que está em etapa de aprendizado podem visualiza-lo para uma melhor compreensão.

O conceito do uso do Google Maps não se aplica a este trabalho, porém ele é apenas utilizado como um exemplo no artigo de Rabello (2007), ele tem como objetivo detalhar todos os métodos utilizados para o acesso a um *Web Service*, visa também mostrar as tecnologias necessárias para que tal aconteça, é utilizado o

conceito de XML e protocolo SOAP que foram muito utilizados na construção deste trabalho.

Ainda sobre o artigo do autor citado acima, há uma breve explicação sobre Android SDK e dos conceitos de um projeto Android no ambiente de desenvolvimento Eclipse IDE.

Por fim, o autor conclui que não é possível criar uma conexão com *web service* a partir de uma API já pronta, e que é necessário utilizar a biblioteca KSOAP2 que utiliza o protocolo SOAP.

A pesquisa bibliográfica teve fator fundamental no desenvolvimento do trabalho, pois a partir desta foram tomadas as decisões para executar a implementação do aplicativo cliente e *web service*.

3 MATERIAIS E MÉTODOS

Neste capitulo serão apresentados os materiais utilizados no desenvolvimento deste trabalho bem como o método de construção, tendo como base os tópicos vistos anteriormente, mostrando a utilização das tecnologias em questão. Na seção 3.1 será apresentado a estrutura de desenvolvimento do trabalho, a seção 3.2 apresenta a aplicação que foi desenvolvida, esta seção se subdivide onde a subseção 3.2.1 exibe o funcionamento do software.

3.1 ESTRUTURA DE PESQUISA E DESENVOLVIMENTO

Esta seção tem como objetivo mostrar o método de desenvolvimento do projeto através de um fluxograma que propõe os pontos eminentes, a Figura 9 apresenta o fluxograma correspondente a este tópico.

Figura 9 – Estrutura de pesquisa e desenvolvimento
Fonte: (Autoria Própria)

Na primeira etapa se refere ao levantamento das tecnologias, onde a equipe de trabalho fez uma pesquisa sobre as tecnologias existentes atualmente na mesma

área de desenvolvimento do trabalho e que poderiam ser utilizadas no desenvolvimento.

A segunda etapa visa definir dentro das possibilidades levantadas na etapa anterior quais as tecnologias realmente seriam utilizadas no desenvolvimento do projeto, as conclusões a respeito das escolhas foram baseadas na sessão teórica do trabalho.

Na terceira etapa foram desenvolvidos os projetos de cliente para o Android e o web service para a transferência de dados do seu respectivo cliente.

A ultima etapa é focada em testes, onde o cliente e o *web service* já estão desenvolvidos.

4 CASO DE USO

Será apresentado nesta seção o *software* desenvolvido para o sistema Android proposto neste trabalho utilizando os conceitos apresentados na seção 2, exibindo as telas com respectivas explicações a respeito do funcionamento e forma de utilização.

4.1 FUNCIONAMENTO

O aplicativo cliente que foi desenvolvido para o Android, foi nomeado de DroidBooks. Este software foi desenvolvido utilizando a plataforma 4.0 do Android com *API Level* 14, com adaptações para que funcione no Android 2.1, ele tem como função buscar *E-Books* no formato ePub e apresentar o texto para a leitura. Apesar de ser um aplicativo cliente que acessa um *web service* ele também possui uma base de dados, pois os livros baixados são armazenados no celular, assim o usuário pode utiliza-los sem uma conexão com a internet, caso um *E-Book* tiver que ser baixado todas as vezes que precisar ser lido então isto não tornaria o *software* funcional e eficaz.

A Figura 10 mostra a inicialização da base de dados, isto acontece assim que o aplicativo é executado e é portanto a primeira tela que o usuário visualiza.

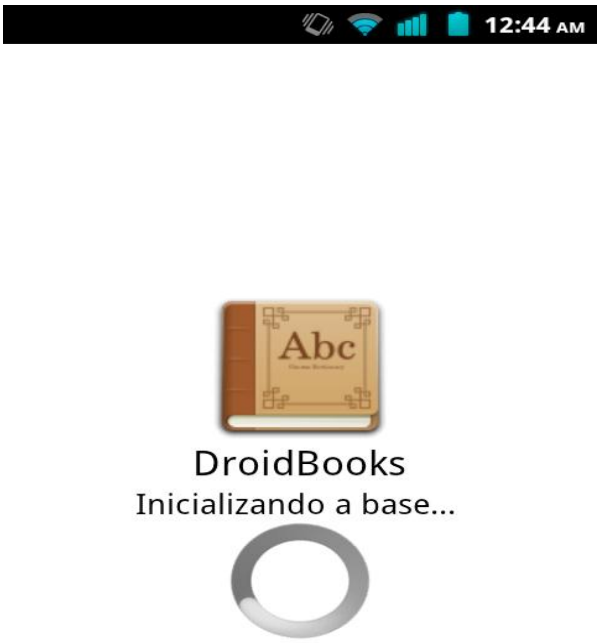

Figura 10 – Tela de entrada do aplicativo
Fonte: (Autoria Própria)

Quando o aplicativo é instalado pela primeira vez no *smartphone* ou em um *tablet* é possível que esse usuário seja o primeiro a utilizar o *software* e não haja até o momento nenhum usuário cadastrado no banco local da aplicação, por este motivo quando o aplicativo é executado pela primeira vez ele exige que o usuário cadastre-se ou execute o login sobre um conta já existente no *Web Service*.

Se a verificação no banco local retornar que nenhum usuário foi encontrado, o aplicativo é redirecionado para a tela de *Login*, esta contém o botão Cadastrar que envia os dados preenchidos ao *Web Service* para que seja gravado na base de dados o novo usuário, caso o usuário já possua uma conta então ele pode preencher os campos Usuário e Senha e clicar no botão Entrar para efetuar o *Logon* no *Web Service*, ao efetuar o *logon* o aplicativo sincroniza os livros já atrelados a esta conta.

A Figura 11 exibe a tela de *Login* conforme citado no parágrafo acima.

Figura 11 – Tela de login e cadastro
Fonte: (Autoria Própria)

Tendo um cadastro no DroidBooks, o usuário não precisa mais informar os dados de *login* a cada vez que o aplicativo for executado, pois uma consulta é feita na base de dados local e o *login* é feito automaticamente. Outro ponto importante é que caso o usuário desinstale o aplicativo e futuramente instale-o novamente caso haja conexão com o *Web Service*, deacordo com o *login* todos os títulos do usuário serão baixados novamente, isto facilita o processo de procura pelos *E-Books* novamente, que também pode ocorrer caso o usuário troque o aparelho celular.

Após o *login* no aplicativo, o usuário pode visualizar uma lista com todos os *E-Books* já baixados com seus respectivos autores, sendo necessário apenas um toque em cima do titulo desejado para que o aplicativo exiba na tela o texto do livro.

Pode ser observado na Figura abaixo a tela em que aparece a lista de livros, neste caso esta exibindo apenas um livro:

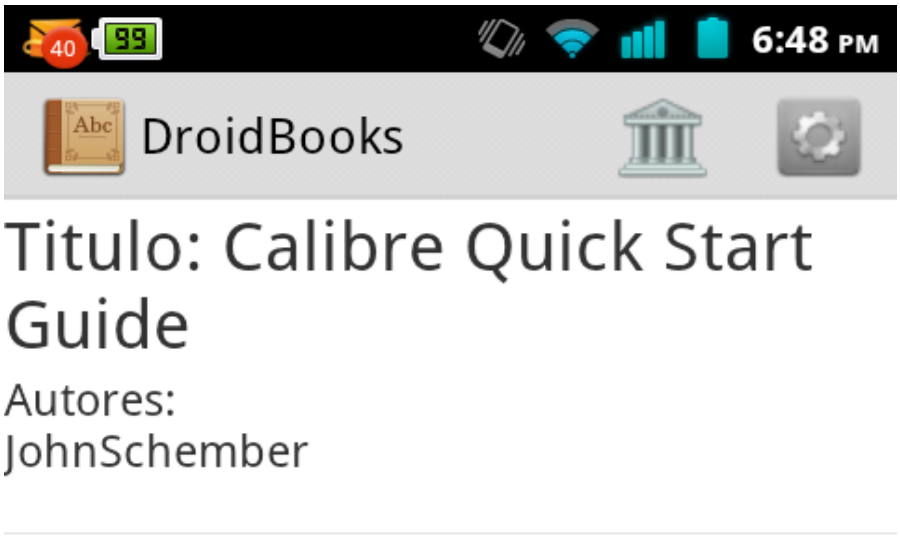

Figura 12 – Lista de livros no celular
Fonte: (Autoria Própria)

Conforme é possível observar na Figura 12, existem ícones no canto superior direito, o ícone que possui a imagem de uma biblioteca acessa o *Web Service* e exibe os títulos existentes para o usuário copia-los para seu aparelho, a Figura 13 exibe essa biblioteca de livros digitais, outro detalhe a ser observado é o campo de busca, pois a tendência é que o numero de *E-Books* aumente no *Web Service* com o passar do tempo, dessa maneira o usuário tem em mente um livro que deseja procurar, basta clicar sobre a lupa que está posicionada no campo superior direito.

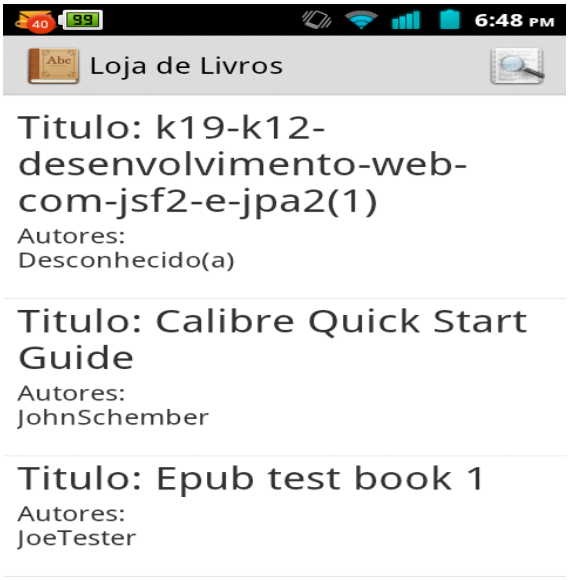

Figura 13 – Lista dos E-Books presentes no Web Service
Fonte: (Autoria Própria).

A Figura 14 mostra o exemplo da busca de um título. Nota-se que a barra superior aparece um campo de busca logo após o usuário clicar sobre a lupa que contém a função de auto completar o texto digitado.

Figura 14 – Exemplo de busca de E-Book
Fonte: (Autoria Própria).

É importante ressaltar que o *software* cliente foi desenvolvido para que o usuário esteja sempre informado sobre os acontecimentos do seu aplicativo, visto que muitas funcionalidades do cliente depende do acesso à internet e do *web service*. Pode ocorrer, por algum motivo do cliente não conseguir estabelecer uma conexão. A Figura 15 exibe a mensagem que o usuário recebe caso a conexão não possa ser estabelecida.

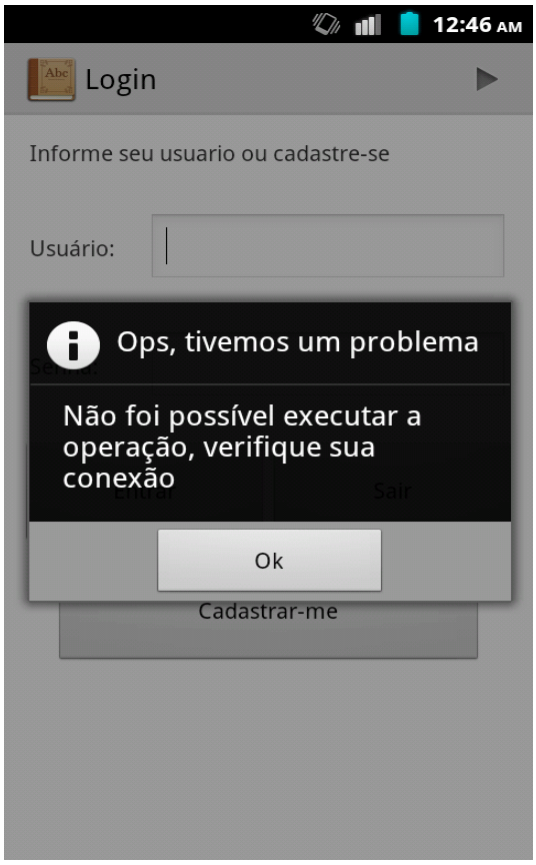

Figura 15 – Erro ao tentar conectar o Web Service
Fonte: (Autoria Própria).

O DroidBooks também faz o tratamento de processos que podem ser demorados e o usuário acaba não sabendo como proceder, por isso ele mostra uma tela ao usuário para que tal saiba que o processo solicitado ainda está em andamento.

A Figura 16 mostra uma tela que é exibida ao usuário quando ele faz uma solicitação para o *web service* mostrar os livros existentes em sua base de dados, isso acontece quando o ícone que possui a imagem de uma biblioteca é pressionado, o tempo em que esta tela fica no *display* do *smartphone* ou *tablet* depende da quantidade de registros de livros presentes na base de dados, a conexão com a internet e a qualidade do sinal são fatores que podem interferir e consequentemente aumentar o tempo de resposta do servidor. O importante é salientar que o usuário não fica sem receber um retorno e sabe que o aplicativo continua em execução.

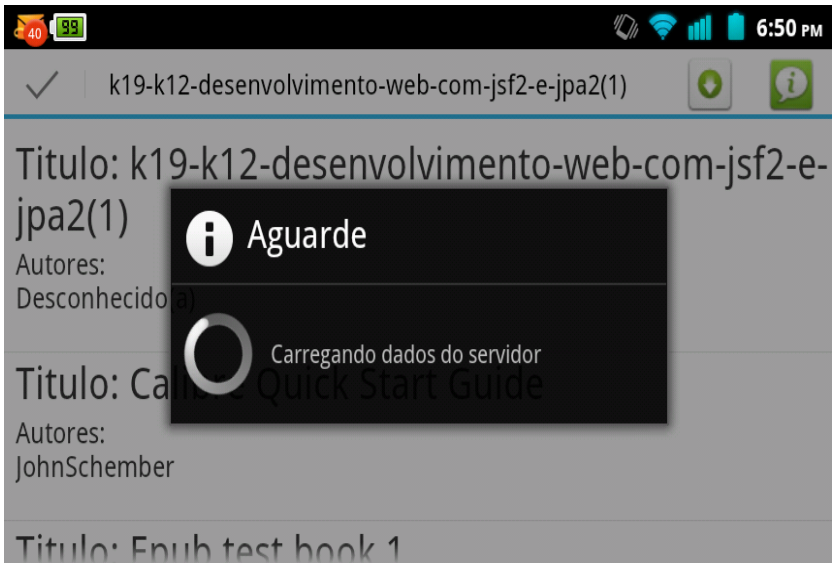

Figura 16 – Carregamento dos livros do Web Service
Fonte: (Autoria Própria).

5 RESULTADOS

Este tópico exibe os resultados obtidos durante o período de desenvolvimento da aplicação referida, tendo visto a complexidade da interação das tecnologias supracitadas é de grandeza proporcional as facilidades propiciadas por estas, e para maior entendimento subdividiremos em grupos as conclusões obtidas.

5.1 SERVIDOR

Durante o desenvolvimento da aplicação do lado do servidor foi implementado como interface externa *um web service* utilizando a especificação da Sun denominada JAX-WS, que viabilizou a disponibilização dos métodos de negócio contidos na classe que detém a responsabilidade de execução da regra de negócio do servidor, essa externalização é facilitada pela tecnologia referida acima pois esta utiliza anotações para referenciar metadados, portanto uma simples anotação sobre a classe externaliza esta, sendo também importante salientar a semelhança desta classe com a especificação do design pattern facade.

Houve também auxilio na implementação da persistência, que foi simplificada pela abstração da complexidade referente a conversão de paradigmas orientada a objeto e relacional, considerando que não foi utilizado uma base relacional e sim uma base orientada a objeto, e não houve conflito de paradigma evitando portanto perca de tempo com conversão e adaptação de dados focando apenas na regra de negócio. Também deve se notar dentre as facilidades obtidas na implementação da base de dados orientada a objetos tivemos como alternativa a utilização da base embarcada, gerando um ganho de desempenho durante a transferência de grande volume de dados como os objetos que continham os livros.

Para a leitura dos ePubs foi utilizado um processo assíncrono que tem por função ler os livros contidos em um determinado diretório e converte-los em objetos para serem armazenados na base. Com o objetivo de facilitar o processo de interpretação do ePub foi utilizado uma biblioteca que realiza essa função provendo um interfaceamento simplificado com os dados contidos no disco.

Como referido nos parágrafos acima podemos notar que a utilização das tecnologias teve por intuito principal abstrair toda a complexidade envolvida com

infraestrutura e seus pormenores viabilizando portanto um foco muito maior na regra de negócio da aplicação, delegando preocupações que não se encaixem no escopo do problema para implementações de terceiros, tendo como resultado mais visível alta produtividade e um software de alta coesão e baixo acoplamento entre seus módulos, otimizando portanto a manutenibilidade do *software* final.

Foi constatado também que o protocolo SOAP não suporta uma transferência de dados muito grande, visto que com *E-Books* de grande tamanho, tem-se um estouro de memória do dispositivo na transferência, a solução para isso seria dividir o arquivo em partes.

5.2 CLIENTE

Considerando que o real enfoque do projeto é a aplicação cliente, e que devida a plataforma Android apresentar peculiaridades, os resultados obtidos serão subdivididos em tópicos.

5.2.1 Plataforma Android

Durante o desenvolvimento baseado na plataforma Android utilizando o respectivo SDK notou-se que apesar das semelhanças com a plataforma JSE existem diferenças relevantes, implícitas no paradigma móvel sobre a qual a plataforma foi projetada.

Uma das diferenças mais notáveis é o modo de tratar componentes gráficos cujos quais são encapsulados por uma classe *Activity* que representa a tela que o usuário vê, e esta detém componentes internos que são especializações da classe *View*, isto implica em uma certa dificuldade ao utilizar o padrão MVC propriamente dito, pois acaba gerando um alto acoplamento entre a *Activity* e a classe de regra de negócio que ela utiliza. Para solucionar este problema foi utilizado o padrão de projeto denominado inversão de controle, cuja implementação foi facilitada pelo *RoboGuice* e que será explicado em um tópico abaixo.

5.2.2 Persistência

Considerando que persistência é uma parte muito peculiar na implementação de um sistema, pois afeta diretamente na modelagem dos dados, afetando também toda estrutura do sistema como regra e visão que estão intimamente atrelado aos dados. Existem duas alternativas que foram já citadas em uma seção deste trabalho, sendo essas db4o e SQLite.

No que tange ao desempenho, foi observado que o db4o apresentou uma vantagem relativamente maior sobre o SQLite, conforme pode ser notado na Figura 17, os resultados foram tomados a partir de um objeto complexo, os números presentes são unidades para comparação.

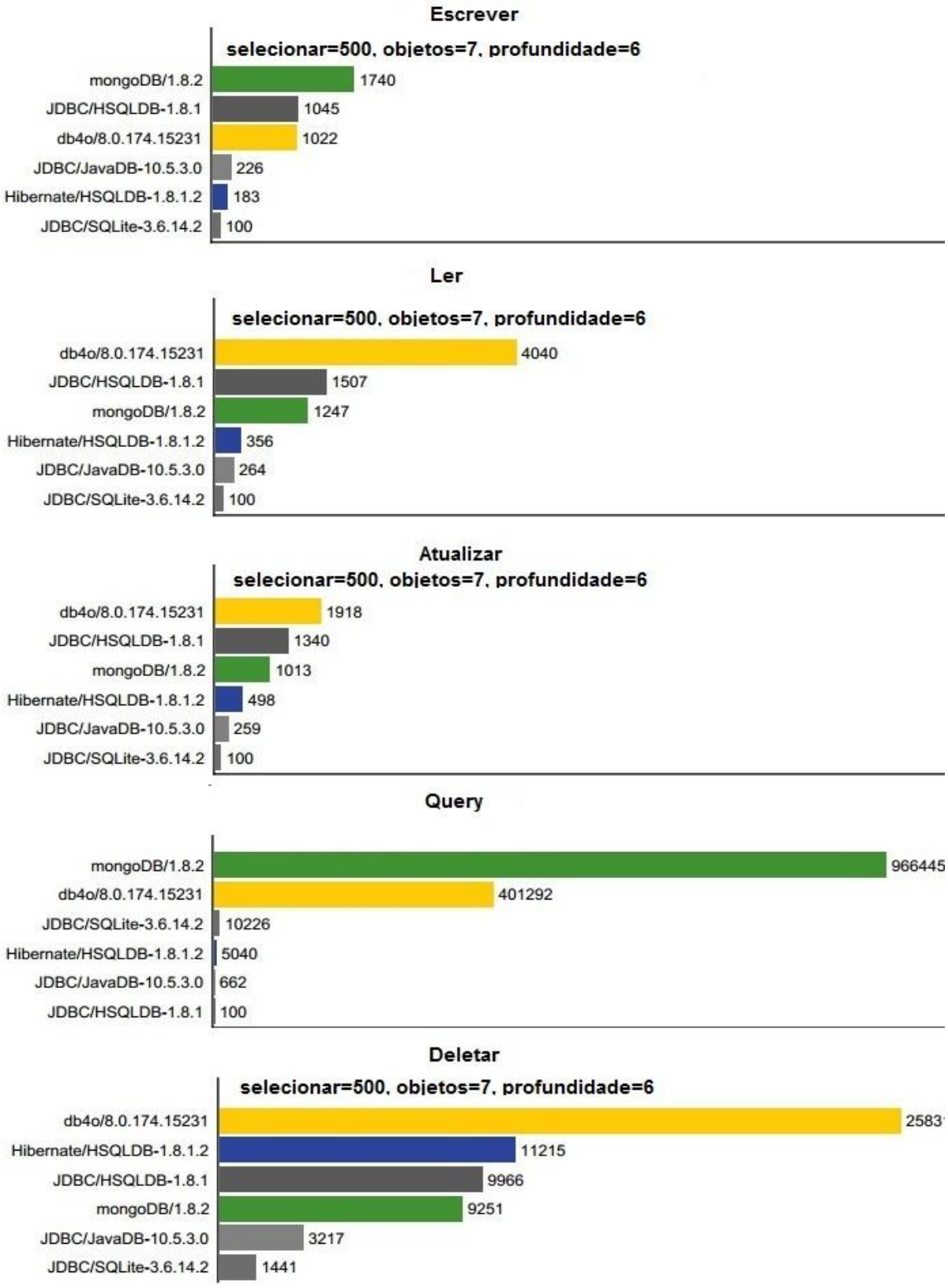

Figura 17 – Comparação de desempenho SGBDs
Fonte: Adaptado de PolePosition (2012)

Portanto considerando a limitação do dispositivo sobre a qual o *software* irá ser executado, foi optado pelo db4o, pelas vantagens já apresentadas.

Quanto à modelagem, pode se observar que o db4o também apresentou uma vantagem considerável sobre o SQLite: este é de paradigma relacional, portanto apresenta uma certa incompatibilidade com o paradigma orientado a objetos que é utilizado como paradigma principal do aplicativo. Já o db4o é nativamente orientado a objeto, consequentemente torna a camada de persistência altamente transparente, permitindo uma modelagem de dados mais homogênea.

No que diz respeito a complexidade da implementação, há uma larga vantagem do db4o sobre o SQLite, pois como o SQLite é um banco de dados relacional faz-se necessário a implementação de uma camada de DAO, que detém a responsabilidade de desmontar o objeto e montar uma instrução compatível com a base, normalmente na linguagem SQL ou sobre APIs específicas conforme pode ser visto no trecho de quadro abaixo.

```
public boolean inserirLivro(Livro livro) {

        this.conecta(cadastroLivro.getCtx());
        ContentValues values = new ContentValues();
        values.put("_id", livro.getCodigo());
        values.put("titulo", livro.getTitulo());
        values.put("autor", livro.getAutor());
        values.put("conteudo", livro.getConteudo());

        db.insert(NOME_TABELA, null, values);
        db.close();
        return true;
}
```

Quadro 11 – Exemplo de insert com SQLite
Fonte: (Autoria Própria)

Por outro lado utilizando uma base orientada a objeto o mesmo trecho de código ficaria muito mais simples, com menos linhas de código como pode ser notado no quadro adiante.

```
public boolean inserirLivro(Livro livro) {
      ObjectContainer db =
Db4oEmbedded.openFile(Db4oEmbedded
        .newConfiguration(), DB4OFILENAME);

      db.store(livro);

       db.close();
      return true;
 }
```

Quadro 12 – Exemplo de insert com db4o
Fonte: (Autoria Própria)

A respeito dos códigos citados acima, a figura abaixo pode demonstrar de forma mais simples a diferença existente entre as duas alternativas.

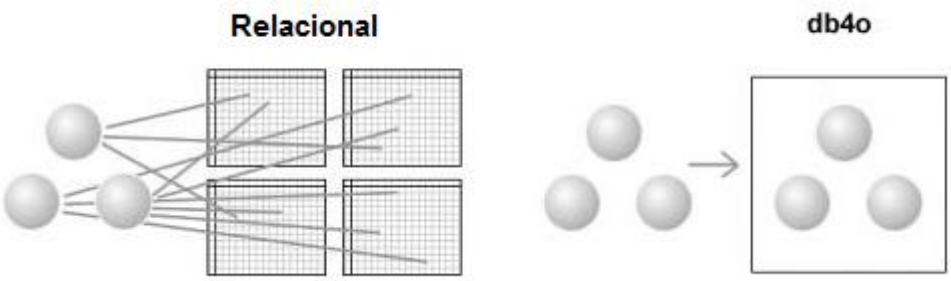

Figura 18 – Exemplo de diferença entre os modelos de persistência
Fonte: (Adaptado de Edlich *et. Al*, 2006, p. 05)

Portanto podemos notar que o db4o apresenta várias vantagens sobre o SQLite, sendo este o motivo de escolha do modelo de persistência utilizado no projeto.

5.2.3 RoboGuice

Pode-se notar que ao utilizar a API nativa do Android para desenvolver a aplicação as camadas ficam altamente acopladas, principalmente a *View*, que é representada na API pela classe *Activity*, a qual tende a comportar além de regras referentes a *View*, muita regra de negócio. Com o intuito de minimizar este problema foi utilizado o *framework RoboGuice*, que é baseado no *GoogleGuice*, um framework de CDI(Injeção de contexto e dependência), que prove uma fácil implementação do padrão de

inversão de controle, onde delegamos ao *framework* certas responsabilidades como o acoplamento entre classes e recursos, viabilizando um desenvolvimento mais focado na regra de negócio. Ao utilizar o *RoboGuice*, é delegado responsabilidades como instanciação de objetos de negócio e instanciação de referências a recursos externos, como as *views* descritas no XML, conforme pode ser visto no quadro 11:

```
@ContentView(R.layout.lista_livros)
public class BookListActivity extends AbstractActivity implements
OnItemClickListener {

	@Inject BookBean bean;
	@InjectView(R.id.listLivros) ListView listaLivrosLocais;
…
}
```

Quadro 13 – Instanciando objeto
Fonte: (Autoria Própria)

Além destas vantagens, foi obtido também o controle de contexto, que permite delimitar a política de instancições que o *framework* utilizará, conforme visto no exemplo abaixo:

```
@Singleton public class
BookBean {
...
}
```

Quadro 14 – Delimitação política de instanciações
Fonte: (Autoria Própria)

6 CONCLUSÃO

Tendo como números que a quantidade de aparelhos existentes no mundo é basicamente a metade da população mundial, e que muitos destes rodam aplicativos desenvolvidos em linguagem de programação Java, tornou-se muito atrativo o estudo das tecnologias citadas no embasamento teórico do trabalho. Como as empresas buscam desenvolver dispositivos móveis mais sofisticados, celulares comuns estão sendo substituídos por smartphones e estes exigem que o sistema operacional embarcado seja mais sofisticado.

O sistema Android apresenta um grande crescimento no mercado, por este motivo a implementação de um cliente para este sistema, e o conhecimento da linguagem Java para desenvolver aplicativos neste sistema tem como vantagem códigos que possam ser reutilizados.

Utilizar um *framework* que possa auxiliar o programador a focar nas regras de negócio permite um desenvolvimento mais eficaz. Portanto dentre as vantagens mais notáveis do *framework* apresentado está a facilidade de desacoplar código, permitindo que, ao programar baseado em interfaces, exista um baixo acoplamento entre as classes, facilitando portanto o reuso. Também deve-se notar que o desenvolvimento fica mais produtivo pois implementações como gerenciamento de contextos, e emissão de eventos ficam muito mais simples, e como resultado temos um produto final com um código mais limpo e claro, aumentando e facilitando assim posteriores manutenções.

Agregar outras tecnologias existentes podem melhorar a qualidade do software, mantê-lo mais organizado e tornar a sua produção mais rápida. Os Web Services são desenvolvidos para que distintas plataformas possam acessar a mesma regra de negócio.

Sendo assim implementar um cliente em Android com uso do Android SDK que proporcionou todas as ferramentas necessárias para o desenvolvimento em Android e que acessa um *Web Service* apresentaram várias vantagens, dentre elas, a de não sobrecarregar o *hardware* do dispositivo móvel, visto que estes tem menor capacidade de processamento devido a limitação física, pois servirão apenas os

resultados do processamento, cabendo ao dispositivo móvel apenas fazer a chamada dos métodos e a passagem dos parâmetros.

Por fim, utilizar uma base de dados orientada a objeto como o db4o, apresentou um desempenho superior a uma base de dados relacional conforme foi apresentado nos gráficos. A camada de persistência se mostrou transparente, e a modelagem dos dados não se mostrou heterogênea, o que significa que durante a implementação do aplicativo não há a necessidade de chaves estrangeiras de uma classe em outras classes agregadas.

6.1 TRABALHOS FUTUROS

Sugere-se para futuros trabalhos utilizar o protocolo Rest ao invés do SOAP. Sendo assim é possível utilizar a proposta deste trabalho porém implementando com o protocolo Rest para a transferência de livros que possuem tamanho grande do arquivo de extensão .epub e o tratamento da transferência de imagens dos *E-Books* da aplicação do servidor para a aplicação cliente.

Outra opção para futuros trabalhos sugere-se a implementação de uma aplicação que utilize as tecnologias deste, mas fazendo uma integração entre outros aplicativos do Android utilizando mais a classe *Intent* citada, como o rastreamento por GPS, onde dispositivos móveis enviam sua localização de latitude e longitude a um *Web Service* que armazena em uma base de dados as informações como o IMEI para identificação do dispositivo que enviou as coordenadas.

Sugere-se também desenvolver também um aplicativo que faça a chamada ao *Web Service* que envia as coordenadas e exibe no Google Maps presente no Android os pontos indicando onde estão os aparelhos que emitem as coordenadas em tempo real. O *Web Service* deve criar um log das coordenadas dos dispositivos. Estes enviam em um tempo determinado, assim o cliente que fizer a requisição pode, se for de sua preferencia, visualizar os caminhos feitos pelo aparelhos.

REFERÊNCIAS

AGUIAR, Ítalo Fernandes et al. (2008) **"Interoperabilidade de Web Services por meio do desenvolvimento de uma arquitetura orientada a serviço - AOS"**, XIV Encontro de Iniciação Científica e Pós-Graduação do ITA, SP, outubro de 2008.

AQUINO, Juliana França Santos. **"Plataformas de Desenvolvimento para Dispositivos Móveis".** Rio de Janeiro – RJ, 2007.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário.** 2. ed. Rio de Janeiro: Elsevier, 2005.

DEITEL, Harvey M.. **XML Como Programar.** São Paulo: Bookman, 2001.
EDLICH, Stefan et al. **The Definitive Guide to db40.** Berkeley: Apress, 2006.

EPUBR. Disponível em: <http://epubr.com.br/epub/> Acesso em 03 mai. 2012.

FARIA, Fernanda B. et al. **Evolução e Principais Características do IDE Eclipse.**Disponível em: <http://www.enacomp.com.br/2010/cd/artigos/completos/enacomp2010_23.pdf>. Acesso em: 02 abr. 2012.

FREIRE, Herval. **Web Services: A nova arquitetura da internet.** Disponível em: <http://www.cnnt.com.br/files/webservices.pdf> Acesso em: 08 abr. 2012.
GOOGLEPLAY. Disponível em: <https://play.google.com/store> Acesso em: 12 mar. 2012.

HANSEN, Roseli P; Pinto, Sérgio Crespo S. C. (2003) **"Construindo Ambientes de Educação Baseada na Web Através de Web Services Educacionais"**, XIV Simpósio Brasileiro de Informática na Educação, RJ, novembro de 2003.

HONORATO, Mauro Jacob. **"Descrição de Documentos na Internete e em eBooks".** Umberlândia - MG, 2010.

LECHETA, Ricardo R.. **Google Android:** Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 2. ed. São Paulo: Novatec, 2010.

LIMA, Lucas Albertins de et al. **ECLIPSE TOOLS - FERRAMENTA PARA AUXÍLIO À COMPOSIÇÃO DINÂMICA DE SOFTWARE.** Disponível em: <http://dsc.ufcg.edu.br/~pet/atividades/Artigos/ARTIGO_ECLIPSETOOLS.pdf>. Acesso em: 31 mar. 2012.

MELO, Eduardo. **O Formato ePub:** Por Onde Começar?. Disponível em: <http://www.simplissimo.com.br/o-formato-epub-por-onde-comecar/>. Acesso em: 15 abr. 2012.

MONTEIRO, André et al. **"Banco de Dados Orientado a Objetos".** BA, 2009. POLEPOSITION. Disponível em: <http://polepos.sourceforge.net/results/PolePositionEmbedded.pdf> Acesso em: 03 mai. 2012.

RABELLO, Ramon Ribeiro. **"Utilização Web Services no Google Android".** Recife - PE, 2007.

SANTANA, Alex Novaes et al. **"Banco de dados orientado a objetos".** BA, 2009.

SILVA, Luciano Édipo Pereira. **"Utilização da plataforma Android no desenvolvimento de um aplicativo para o cálculo do Balanço Hídrico Climatológico".** Dourados - MS, 2009.

SIMÃO, André Morum de L. et al. **Aquisição de Evidências Digitais em Smartphones Android.** Disponível em: <http://www.icofcs.org/2011/ICoFCS2011-PP09.pdf>. Acesso em: 07 abr. 2012

TOSHI. **Injeção de dependências no Android com RoboGuice.** Disponível em: <http://blog.caelum.com.br/injecao-de-dependencias-no-android-com-roboguice/>. Acesso em: 18 abr. 2012.

VERSANT. Disponível em: <http://www.versant.com/> Acesso em 28 mar. 2012

ZBRA SOLUTIONS. Disponível em: <http://zbra.com.br/> Acesso em: 13 mar. 2012