



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE  
CENTRO DE CIÊNCIAS EXATAS E DA TERRA  
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA  
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO  
MESTRADO ACADÊMICO EM SISTEMAS E COMPUTAÇÃO



# Uma Proposta de Solução para Funcionamento Offline em Aplicações Android

Jean Guerethes Fernandes Guedes

Natal-RN  
Agosto/2015

Catálogo da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial  
Centro de Ciências Exatas e da Terra – CCET.

Guedes, Jean Guerethes Fernandes.

Uma proposta de solução para funcionamento offline em aplicações android /  
Jean Guerethes Fernandes Guedes. - Natal, 2015.  
88 f.: il.

Orientador: Prof. Dr. Gibeon Soares de Aquino Júnior.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro  
de Ciências Exatas e da Terra. Programa de Pós-Graduação em Sistemas e  
Computação.

1. Engenharia de software – Dissertação. 2. Framework – Dissertação. 3.  
Persistência – Dissertação. 4. Sincronização – Dissertação. 5. Aplicações móveis –  
Dissertação. 6. Replicação – Dissertação. 7. Offline – Dissertação. I. Aquino Júnior,  
Gibeon Soares de II. Título.

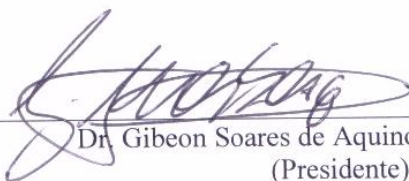
RN/UF/BSE-CCET

CDU: 004.41

JEAN GUERETHES FERNANDES GUEDES

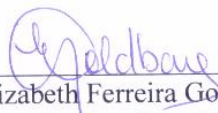
Uma Proposta de Solução para Funcionamento Offline em Aplicações  
Android

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.



---


Dr. Gibeon Soares de Aquino Junior – UFRN  
(Presidente)



---

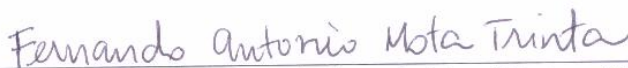
Dr.<sup>a</sup>. Elizabeth Ferreira Gouvêa – UFRN  
(Vice-coordenadora do Programa)

Banca Examinadora



---

Dr. Eduardo Henrique da Silva Aranha – UFRN  
(Examinador)



---

Dr. Fernando Antonio Mota Trinta – UFC  
(Examinador)

Jean Guerethes Fernandes Guedes

# Uma Proposta de Solução para Funcionamento Offline em Aplicações Android

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Sistemas e Computação.

*Linha de pesquisa:*

Engenharia de Software

Orientador

Prof. Dr. Gibeon Soares de Aquino Jr.

PPGSC – PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO  
DIMAP – DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA  
CCET – CENTRO DE CIÊNCIAS EXATAS E DA TERRA  
UFRN – UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Natal-RN

Agosto/2015

Dissertação de Mestrado sob o título *Uma Proposta de Solução para Funcionamento Offline em Aplicações Android* apresentada por Jean Guerethes Fernandes Guedes e aceita pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

---

Prof. Dr. Gibeon Soares de Aquino Jr.

Presidente

DIMAp – Departamento de Informática e Matemática Aplicada

UFRN – Universidade Federal do Rio Grande do Norte

---

Prof. Dr. Eduardo Henrique da Silva Aranha

Examinador

DIMAp – Departamento de Informática e Matemática Aplicada

UFRN – Universidade Federal do Rio Grande do Norte

---

Prof. Dr. Fernando Antônio Mota Trinta

Examinador

Departamento de Computação

Universidade Federal do Ceará

Natal-RN, 28 de Agosto de 2015.

Dedico esta dissertação a minha esposa, Mariana, fonte permanente de apoio, inspiração e pela positividade depositada nesse meu projeto. Com amor, meu eterno agradecimento.

# Agradecimentos

Agradeço a Deus acima de tudo, por permitir a existência daqueles a quem agradeço em seguida, por me fortalecer sempre e me transmitir fé no caminho que me mostra incessantemente.

À minha esposa, parceira e companheira, Mariana Cruz, por ser tão importante na minha vida. Sempre ao meu lado, me fazendo acreditar que posso mais que imagino. A ela também agradeço por ter me presenteado com minha segunda família, que tenho imenso carinho e admiração.

Agradeço aos meus pais, por sempre me proporcionarem totais condições para a realização dos meus estudos, em especial, a minha mãe, Gorethe, por ter feito de mim o que sou, cuja devoção dos filhos é digna de um relicário.

Ao meu irmão, pelo companherismo a mim dedicado desde seu nascimento.

À minha querida baixinha Si, por ter sido minha segunda mãe e por ter doado sua vida à nossa família, sendo integrante e parte dela.

Ao meu orientador, Prof. Dr. Gibeon Aquino, pelas lições ensinadas e pelo tempo a mim dedicado na orientação dessa dissertação, entendendo minhas dificuldades e me apoiando sempre que preciso. Trata-se de um professor extremamente preocupado com seus alunos. Obrigado mesmo Professor.

*“Tu te tornas eternamente responsável por aquilo que cativas.”*

Antoine de Saint-Exupéry



# Uma Proposta de Solução para Funcionamento Offline em Aplicações Android

Autor: Jean Guerethes Fernandes Guedes

Orientador: Prof. Dr. Gibeon Soares de Aquino Jr.

## RESUMO

Diante da crescente demanda pela criação de aplicativos móveis, impulsionada pelo uso cada vez mais frequente de smartphones e tablets, cresceu na sociedade a necessidade por acesso a dados remotos de forma integral na utilização do aplicativo móvel em ambientes sem conectividade, em que não há disponibilização de acesso à rede em todos os momentos. Diante dessa realidade, esse trabalho teve como objetivo o desenvolvimento de uma solução através de um framework, que apresente como principais funções o provimento de um mecanismo de persistência, replicação e sincronização dos dados, contemplando a criação, remoção, atualização e visualização dos dados persistidos ou requisitados, mesmo estando o dispositivo móvel sem conectividade com a rede. Do ponto de vista das práticas de programação e arquitetura, isso reflete em definir estratégias para que as principais funções do framework sejam atendidas. Através de um estudo controlado foi possível validar a solução proposta, sendo constatado ganhos como a redução na quantidade de linhas de código e de quantidade de tempo necessários para realizar o desenvolvimento de um aplicativo sem que houvesse aumento significativo para a realização das operações.

*Palavras-chave:* Framework; Persistência; Sincronização; Aplicações Móveis; Replicação; Offline.

# Uma Proposta de Solução para Funcionamento Offline em Aplicações Android

Author: Jean Guerethes Fernandes Guedes

Supervisor: Prof. Dr. Gibeon Soares de Aquino Jr.

## ABSTRACT

Given the growing demand for the development of mobile applications, driven by use increasingly common in smartphones and tablets grew in society the need for remote data access in full in the use of mobile application without connectivity environments where there is no provision network access at all times. Given this reality, this work proposes a framework that present main functions are the provision of a persistence mechanism, replication and data synchronization, contemplating the creation, deletion, update and display persisted or requested data, even though the mobile device without connectivity with the network. From the point of view of the architecture and programming practices, it reflected in defining strategies for the main functions of the framework are met. Through a controlled study was to validate the solution proposal, being found as the gains in reducing the number of lines code and the amount of time required to perform the development of an application without there being significant increase for the operations.

*Keywords:* Framework; Persistence; Synchronization; Mobile applications; replication; Offline.

# Lista de figuras

1	Arquitetura do Android (ANDROID, 2014). . . . .	p. 23
2	Ciclo de Vida da Activity (GARGENTA, 2011). . . . .	p. 25
3	Intent (GARGENTA, 2011). . . . .	p. 26
4	Banco de Dados Distribuído (DATE, 2004). . . . .	p. 30
5	Ilustração de um Experimento. . . . .	p. 32
6	Arquitetura de Software . . . . .	p. 38
7	Relacionamento entre as Classes do Utils e os demais Módulos . . . . .	p. 39
8	Módulo Comunicação com o Servidor . . . . .	p. 40
9	Módulo de Persistência . . . . .	p. 42
10	Classe responsável pelos dados a serem sincronizados . . . . .	p. 43
11	Diagrama de Sequência <i>@OnlyOnLine</i> . . . . .	p. 50
12	Diagrama de Sequência <i>@OnlyLocalStorage</i> . . . . .	p. 50
13	Diagrama de Sequência para as Classes sem anotação . . . . .	p. 51
14	Box-Plot Tempo de Desenvolvimento . . . . .	p. 66
15	Estrutura da primeira versão do Coletor de Presença Mobile . . . . .	p. 75
16	Estrutura da nova versão do Coletor de Presença Mobile . . . . .	p. 76
17	Arquitetura do MCSync (SEDIVY et al., 2012). . . . .	p. 78
18	Arquitetura do SyncML (STAGE, 2005). . . . .	p. 80
19	Estrutura do algoritmo SAMD (CHOI et al., 2010). . . . .	p. 81

# Lista de tabelas

1	Tabela com as possíveis combinações do Insert . . . . .	p. 45
2	Tabela com as possíveis combinações da Busca . . . . .	p. 45
3	Tabela com as possíveis combinações do Update . . . . .	p. 46
4	Tabela com as possíveis combinações do Delete . . . . .	p. 47
5	Definição da Sequência dos Problemas . . . . .	p. 63
6	Preparação do Experimento . . . . .	p. 63
7	Resultados do Experimento . . . . .	p. 65
8	Projeto 1 Mann-Whitney . . . . .	p. 66
9	Projeto 2 Mann-Whitney . . . . .	p. 66
10	Performance das Requisições . . . . .	p. 67
11	Linhas de código desenvolvido para solucionar o Problema . . . . .	p. 68
12	Comparativo entre as versões do Coletor de Presença Mobile. . . . .	p. 76

# Lista de abreviaturas e siglas

REST – Representational State Transfer

CIO – Chief Information Officer

API – Application Programming Interface

SO – Sistemas Operacionais

IDE – Integrated Development Environment

SQL – Structured Query Language

HTTP – Hypertext Transfer Protocol

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

SGBD – Sistema de Gerenciamento de Banco de Dados

SGBD – Sistema de Gerenciamento de Banco de Dados Distribuídos

JSON – JavaScript Object Notation

JPA – Java Persistence API

GQM – Goal/- Question/Metric

IDE – Integrated Development Environment

GAE – Google App Engine

XML – eXtensible Markup Language

DB4o – Database for Objects

# Lista de listagens

3.1	Exemplo de Mapeamento de uma Classe de Domínio . . . . .	p. 47
3.2	Consulta de Dados de uma Classe @OnlyOnLine . . . . .	p. 49
3.3	Exemplo do Uso do EqualsUtils e hashCodeUtils . . . . .	p. 52
3.4	Consulta Geral . . . . .	p. 53
3.5	Consulta Realizada pelo Atributo Informado . . . . .	p. 53
3.6	Consulta Realizada pelos Atributos Informados . . . . .	p. 54
3.7	Consulta Realizada retornando apenas um registro . . . . .	p. 55
3.8	Método do Insert . . . . .	p. 55
3.9	Implementação do método que força a sincronização . . . . .	p. 56
3.10	Possibilidade de executar uma instrução <i>SQL</i> . . . . .	p. 58
5.1	Inicialização do Framework . . . . .	p. 71
5.2	Carrega os eventos permitidos para o dispositivo . . . . .	p. 72
5.3	Carrega todas as atividades do evento . . . . .	p. 72
5.4	Salvando a presença do participante . . . . .	p. 73

# Sumário

<b>1</b>	<b>Introdução</b>	p. 15
1.1	Contexto . . . . .	p. 16
1.2	Problema . . . . .	p. 17
1.3	Objetivos . . . . .	p. 18
1.4	Metodologia . . . . .	p. 18
1.5	Estrutura do Trabalho . . . . .	p. 20
<b>2</b>	<b>Fundamentação Teórica</b>	p. 21
2.1	Computação Móvel . . . . .	p. 21
2.2	Android . . . . .	p. 22
2.2.1	Aplicativos Android . . . . .	p. 23
2.2.2	Activity . . . . .	p. 24
2.2.3	Intents . . . . .	p. 25
2.2.4	SQLite . . . . .	p. 26
2.2.5	DbHelper . . . . .	p. 27
2.2.6	Notificações . . . . .	p. 27
2.3	Tecnologias Empregadas . . . . .	p. 27
2.3.1	Aplicações Web . . . . .	p. 27
2.3.2	RestFul Web Services . . . . .	p. 28
2.3.3	Banco de Dados Distribuídos . . . . .	p. 28
2.4	Framework . . . . .	p. 30
2.5	Experimento na Engenharia de Software . . . . .	p. 31

2.5.1	Organização do Experimento . . . . .	p. 32
<b>3</b>	<b>Proposta do Framework Offdroid</b>	p. 34
3.1	Requisitos . . . . .	p. 34
3.2	Arquitetura . . . . .	p. 36
3.2.1	Utils . . . . .	p. 38
3.2.2	Comunicação com o Servidor . . . . .	p. 40
3.2.3	Persistência . . . . .	p. 41
3.2.4	Sincronização . . . . .	p. 43
3.3	Estratégias de Configuração . . . . .	p. 44
3.4	Mapeamento das Classes . . . . .	p. 47
3.5	Comportamento do Framework . . . . .	p. 49
3.6	Classes Utilitárias . . . . .	p. 52
3.6.1	EqualsUtils e hashCodeUtils . . . . .	p. 52
3.7	Métodos Existentes . . . . .	p. 53
3.8	Conclusões . . . . .	p. 58
<b>4</b>	<b>Experimento para Avaliação do Framework</b>	p. 59
4.1	Definição do Experimento . . . . .	p. 59
4.1.1	Objetivos do Experimento . . . . .	p. 59
4.1.2	Questões e Métricas . . . . .	p. 60
4.2	Planejamento . . . . .	p. 60
4.2.1	Formulação das Hipóteses . . . . .	p. 61
4.2.2	Seleção das Variáveis . . . . .	p. 61
4.2.3	Seleção dos Participantes . . . . .	p. 62
4.3	Design do Experimento . . . . .	p. 62
4.4	Preparação do Experimento . . . . .	p. 63



4.5	Processo de Experimentação . . . . .	p. 63
4.5.1	Avaliação de Validade . . . . .	p. 64
4.5.2	Análise dos Resultados . . . . .	p. 65
4.6	Considerações Finais . . . . .	p. 68
<b>5</b>	<b>Prova de Conceito: Coletor Presença Mobile</b>	p. 70
5.1	Introdução . . . . .	p. 70
5.2	Inicialização . . . . .	p. 70
5.3	Utilização . . . . .	p. 71
5.4	Comparativo entre as versões . . . . .	p. 74
5.5	Conclusão . . . . .	p. 77
<b>6</b>	<b>Trabalhos Relacionados</b>	p. 78
<b>7</b>	<b>Considerações Finais</b>	p. 83
7.1	Contribuições . . . . .	p. 84
7.2	Limitações . . . . .	p. 84
7.3	Trabalhos Futuros . . . . .	p. 85
	<b>Referências</b>	p. 86

# 1 Introdução

A sociedade contemporânea vivência um período tecnológico em desenvolvimento. Isso ocorre diante da crescente evolução da computação móvel, traduzida no uso constante dos smartphones e tablets, que gera um aumento nas demandas da criação de novos aplicativos.

Diante dos novos hábitos da sociedade em portar dispositivos móveis, um dos obstáculos que as empresas vêm enfrentando consiste na tentativa de migrar versões de seus sistemas e aplicações para a computação móvel, pois há instabilidade no acesso à rede, não existe uma solução de replicação e de sincronização de dados e, por fim, não há um padrão quanto a persistência dos dados na plataforma Android.

Particularmente, um problema de implementação a ser abordado é a replicação de dados quando o dispositivo móvel estiver operando em modo offline. Algumas soluções fechadas e proprietárias foram definidas para a replicação de arquivos e sincronização, como (SMITH, 2013), (MICROSOFT, 2015) e (DRIVE, 2015), mas estas foram consideradas inadequadas para o tratamento de dados porque, entre outros fatores, apresentaram implementações que foram estritamente direcionadas a arquivos de replicação e sincronização, quando na verdade deveriam ser direcionados a dados.

Para minimizar esses obstáculos, este trabalho propõe um framework chamado OffDroid, cuja função principal é a persistência, replicação e sincronização dos dados envolvidos na interação entre a aplicação móvel e dos seus serviços de back-end da web. O framework permite, de forma transparente, que aplicações executem a inserção, remoção, atualização e consulta de dados persistidos localmente ou solicitados via REST (FIELDING, 2000), mesmo quando o dispositivo móvel estiver funcionando sem conectividade.

O OffDroid baseia-se na plataforma android, e tem a intenção de apoiar o desenvolvimento de aplicativos que precisam se adaptar a inconstância do acesso à rede. As aplicações serão capazes de executar operações sem a necessidade de conexão com a internet, fazendo apenas uso da base de dados local. Assim, o aplicativo pode ter mais

cobertura, abrangendo tanto os usuários com acesso contínuo à rede como aqueles que não têm acesso em todos os momentos.

Embora existam várias plataformas para dispositivos móveis, para este trabalho a plataforma escolhida foi a Android, uma vez que tem uma licença de código aberto; é desenvolvida utilizando linguagem de programação Java, que é uma das mais populares no mundo; e também pelo fato de existir uma maior diversidade de dispositivos móveis que utilizam esta plataforma (SOWAH; FIAWOO, 2013). Entretanto a solução aqui apresentada pode ser aproveitada e migradas para as demais plataformas existentes.

## 1.1 Contexto

No cenário social hodierno é notório o crescimento da computação móvel, fenômeno este impulsionado pelas vendas dos smartphones e tablets, que se tornaram mercadorias de baixo custo, massificando esses dispositivos inclusive nas camadas mais carentes da sociedade.

Em todo o mundo, as vendas de celulares para usuários finais totalizaram 460,3 milhões de unidades durante o primeiro trimestre de 2015, um aumento de 2,5% por cento em relação ao mesmo período de 2014. O maior crescimento registrado ocorreu no mercado emergente, excluindo a China, que apresentou um aumento de 40% por cento nas vendas durante o primeiro trimestre de 2015 (GARTNER, 2015).

Esse aumento da quantidade de dispositivos é acompanhado pelo aumento da quantidade de usuários que tem acesso a internet por meio da computação móvel, fazendo com que as empresas tenham que acompanhar esses novos hábitos tecnológicos. De acordo com o estudo apresentado pela empresa Micro Focus (FOCUS, 2013), espera-se que dentro de três anos 50% das aplicações negociais estejam presentes também nos dispositivos móveis.

No que tange o estudo realizado no Brasil, os CIOs e diretores de TI brasileiros consultados na pesquisa afirmaram que a perspectiva é aumentar para 88% o número no volume de aplicativos desenvolvidos para smartphones, aduzindo ainda ser possível o aumento em 72% no número de apps de negócios voltados a tablets. A pesquisa identificou também a existência em cerca de 25,62% das aplicações de negócios que são voltadas para dispositivos móveis, contra uma média de 30,90% no mundo.

Diante da existência das inúmeras plataformas para os mais diversos dispositivos móveis, três delas se destacam pela quantidade de smartphones e tablets no mercado, são

elas Android(Google), Windows Phone(Microsoft) e o IOS(Apple). Para o desenvolvimento desse trabalho foi escolhida a plataforma Android por diversos motivos, seja em virtude da sua licença ser open-source, seja por ser desenvolvida sobre a linguagem de programação java, que é uma das mais populares do mundo, ou ainda pelo simples fato de existir uma maior diversidade de dispositivos móveis utilizando essa plataforma, e, conforme (SOWAH; FIAWOO, 2013) ainda por apresentar cerca de 500.000 dispositivos ativados diariamente.

## 1.2 Problema

O problema consiste em portar uma aplicação para o contexto móvel \*\*\*, dentre as inúmeras problemáticas existentes, a tentativa de conversão dos tradicionais sistemas para aplicações móveis mostra-se relevante, visto que não seria viável armazenar todas as informações contidas na base de dados Web no dispositivo móvel, assim como a impossibilidade de visualização de todas as informações contidas na versão Web para mobile devido ao tamanho reduzido da tela deste. Devendo ser ressaltado o fato de que algumas funcionalidades são relevantes apenas no contexto da computação móvel (FILHO, 2014).

Outra problemática a ser abordada consiste na replicação de dados quando o dispositivo móvel se encontrar operando no modo offline, devido a não existência de um padrão de sincronização bilateral e replicação dos dados na plataforma android. Por esse motivo, cada aplicação desenvolvida que necessita apresentar essas características acaba sendo levada a desenvolver sua própria abordagem para tratar essa problemática, fazendo com que não haja um padrão que possa ser seguido pelos desenvolvedores que se deparam com tais problemas.

Já quanto a gerência dos dados que precisam ser sincronizados, também não foi encontrado um padrão para gerenciar e sinalizar quais dados devem ser sincronizados quando modificado no dispositivo, uma vez que os dados devem ser sincronizados do dispositivo móvel para o servidor de aplicação e tratado de forma única.

Outro problema encontrado foi a não existência de um framework que pudesse auxiliar no desenvolvimento dos aplicativos, tomando como base as configurações das classes e da conectividade do dispositivo móvel, bem como todo esse trabalho de criação do banco de dados, das tabelas e das classes responsáveis pela comunicação com servidor sempre que necessário.

Diante desta carência, se faz necessário implementar as classes responsáveis pela per-

sistência quando necessário, bem como o desenvolvimento do mecanismo de sincronização dos dados entre o dispositivo móvel e o servidor de aplicação.

## 1.3 Objetivos

O presente trabalho tem como objetivo o desenvolvimento de uma solução, chamada de OffDroid, para auxiliar os desenvolvedores na construção de aplicativos móveis, promovendo um conjunto de funcionalidade que auxiliará na manipulação de dados corporativos nos aplicativos móveis.

Baseado no objetivo geral, este trabalho tem os seguintes objetivos específicos:

- Implementar um framework capaz de funcionar em modo offline, replicando e sincronizando os dados;
- Projetar e descrever o uso do framework e os métodos nele contemplados;
- Estudo de Caso;
- Avaliar o framework com base em métricas de qualidade de software;

## 1.4 Metodologia

Inicialmente foi realizado um levantamento de requisitos no intuito de descobrir e listar as principais funcionalidades que o framework deveria contemplar. Uma vez que estes foram devidamente coletados, mediante a uma entrevista com alguns stakeholders e alguns desenvolvedores da plataforma Android, uma pesquisa foi realizada com a finalidade de encontrar soluções que apresentassem as funcionalidades de persistência, de replicação e sincronização de dados, especialmente para atender a aplicação em modo offline, no entanto nenhuma solução atendeu completamente os requisitos desejados.

Diante da ausência de solução que apresentasse todos os requisitos almejados, foi iniciada a construção de um framework próprio que pudesse atender o persistência, replicação e sincronização de dados na ausência de conectividade no dispositivo móvel.

Quanto a replicação dos dados, ficou definido que só ocorrerá se dois requisitos forem atendidos haver uma requisição junto ao servidor de aplicação e esse retornar dados e a classe tiver sido configurada para que os dados sejam persistidos no banco de dados do dispositivo móvel.

Durante o processo de levantamento de requisitos para as aplicações multiusuários, ficará a cargo do desenvolvedor realizar o mapeamento do banco de dados, podendo ser realizado de duas formas, utilizando um único banco de dados para todos os usuários ou criando um banco de dados para cada usuário da aplicação.

Mesmo não havendo padrão referente para replicação e sincronização de dados no contexto móvel, mais precisamente para a plataforma Android, há conhecimento da existência de algumas soluções fechadas e proprietárias para a replicação e sincronização de arquivos, tendo sido realizado um estudo no intuito de identificar as abordagens utilizadas nesse padrão para tentar aplicá-las ao cenário da replicação e sincronização de dados.

Em seguida foi realizada a construção da solução, apresentando como principal foco fornecer ao desenvolvedor um conjunto de funcionalidades que contemple a persistência, replicação e sincronização de dados no modo offline.

Na base da implementação referente a persistência relacional foi utilizada a API *h4Android*<sup>1</sup>, tendo sido realizada algumas modificações para atender as necessidades levantadas, a serem explanadas no Capítulo 3. Já na implementação referente a comunicação entre o dispositivo e o servidor de aplicação não foi utilizada nenhuma API externa, foram utilizadas as API já existentes na plataforma *Android*.

O experimento controlado foi realizado com quatro desenvolvedores com conhecimento acerca da plataforma Android, cuja finalidade era responder a três perguntas levantadas sobre o framework, em que as respostas apresentadas eram embasadas no desenvolvimento de problemas similares, utilizando ou não o framework. A primeira pergunta consistiu em verificar se houve ganho na produtividade, a segunda tinha como intuito comparar a quantidade de linhas códigos desenvolvidas e a terceira pergunta consistia na medição do tempo médio das operações existentes nas soluções.

A Prova de Conceito utilizou o aplicativo coletor de presença, já existente e previamente desenvolvido pela SINFO, órgão da Universidade Federal do Rio Grande do Norte - UFRN, diretamente subordinada à Reitoria, responsável por coordenar atividades de administração, projeto e desenvolvimento dos sistemas computacionais de natureza corporativa, pelo gerenciamento da infraestrutura de redes da UFRN e por elaborar a política de informática da UFRN, o aplicativo consiste na coleta de presenças de todos os participantes de um determinado evento, em que novas funcionalidades foram incorporadas, através do uso do framework, como por exemplo, a sincronização automática dos dados.

---

<sup>1</sup><https://code.google.com/p/h4android/wiki/h4Android>

Por fim foi realizada uma avaliação entre as duas versões do Coletor de Presença, uma aplicando uma com o framework e outra sem, tendo como o intuito descobrir quais foram os ganhos obtidos entre as duas versões. A avaliação consistiu em analisar a quantidade de classes implementadas e a quantidade linhas de código.

## 1.5 Estrutura do Trabalho

A continuação desta dissertação está organizada conforme se segue:

- O Capítulo 2 aborda a fundamentação teórica, contextualizando a temática do Android e do Framework, fazendo interação com a computação móvel;
- O Capítulo 3 apresenta a proposta do framework Offdroid, descrevendo de forma pormenorizada a proposta da arquitetura, as estratégias de configuração e o comportamento do framework;
- O Capítulo 4 descreve o experimento controlado, tendo como base a descrição da sua realização, o planejamento e a análise dos resultados obtidos;
- O Capítulo 5 descreve a prova de conceito, mostrando um comparativo de duas abordagens utilizando ou não o framework;
- O Capítulo 6 apresentar os trabalhos relacionados acerca da temática apresentada;
- O Capítulo 7, nas considerações finais, mostrará as principais contribuições desta pesquisa e discussões acerca das possíveis soluções encontradas;

## 2 Fundamentação Teórica

Este capítulo aborda conceitos relevantes acerca da computação móvel, possibilitando o entendimento do contexto ao qual o trabalho está inserido. Serão abordadas definições em relação a aplicação Web, aos serviços RestFul, ao banco de dados distribuídos e o conceito de framework, que será o produto final deste trabalho.

### 2.1 Computação Móvel

A Computação móvel representa um novo paradigma computacional. Surge como uma quarta revolução na computação, antecedida pelos grandes centros de processamento de dados da década de sessenta, o surgimento dos terminais nos anos setenta e as redes de computadores na década de oitenta. O novo paradigma permite que usuários desse ambiente tenham acesso a serviços independente de onde estão localizados enfatizando a mobilidade. Dessa forma a computação móvel amplia o conceito tradicional de computação distribuída. Tecnicamente falando, computação móvel é um conceito que envolve comunicação sem fio, mobilidade, interação e processamento, apresentando um tamanho reduzido, com acesso a dado através de rede sem fio, desprendendo-se cada vez mais de rede elétrica e seus cabos conectores.

Um sistema distribuído com computadores móveis é formado por uma parte tradicional, uma infra-estrutura de comunicação fixa com computadores estáticos, que está interligada a uma parte móvel representada por uma área ou célula onde existe a comunicação sem fio dos elementos computacionais móveis. Com a diminuição dos custos desses dispositivos, a computação móvel se tornou viável não somente para o segmento empresarial, mas para as pessoas de uma forma geral. A disponibilidade dos equipamentos e a solução de antigos problemas relativos a ruído e interferência em sistemas de comunicação sem fio abriram o interesse pelo tema.

A questão principal na computação móvel é a mobilidade, que tem como objetivo



principal prover para os usuários um ambiente computacional com um conjunto de serviços comparáveis aos existentes num sistema distribuído de computadores estáticos, mas que permita a mobilidade. Possibilitando assim um novo meio de comunicação interativa, obtida através do uso de smartphones, tablets ou até de sistemas embarcados, que apresentam em sua grande maioria ambiente multimídia. Diferentes dispositivos móveis se comunicam entre si através de uma rede de comunicação sem fio, independentemente de sua localização.

Com o constante avanço tecnológico é possível constatar uma significativa mudança no processamento e armazenamento dos dados, principalmente em relação a possibilidade de realização de diversas atividades, tal como a troca de informação no uso aplicativos do tipo Messenger. Essa mudança ocorreu, dentre outros fatores, pelas recentes melhorias realizadas na interface, deixando-a mais interativa e, principalmente, pelo fato dos dispositivos móveis terem sido popularizados e acessíveis a grande parte da sociedade.

## 2.2 Android

A plataforma Android, pertencente a empresa Google, foi lançada em novembro de 2007. Antes, a empresa possuía um framework sob responsabilidade da Open Handset Alliance, tendo como intuito fornecer um código aberto para estimular o desenvolvimento de aplicações para a plataforma (GRØNLI et al., 2014).

A arquitetura do SO Android, como pode ser vista na Figura 1, é sub-divida em quatro camadas, estratégia que já é utilizada em diversos outros SO. A linguagem Java é utilizada para o desenvolvimento das aplicações, já as bibliotecas utilizadas foram desenvolvidas utilizando as linguagens C/C++ (WU; LUO; LUO, 2010).

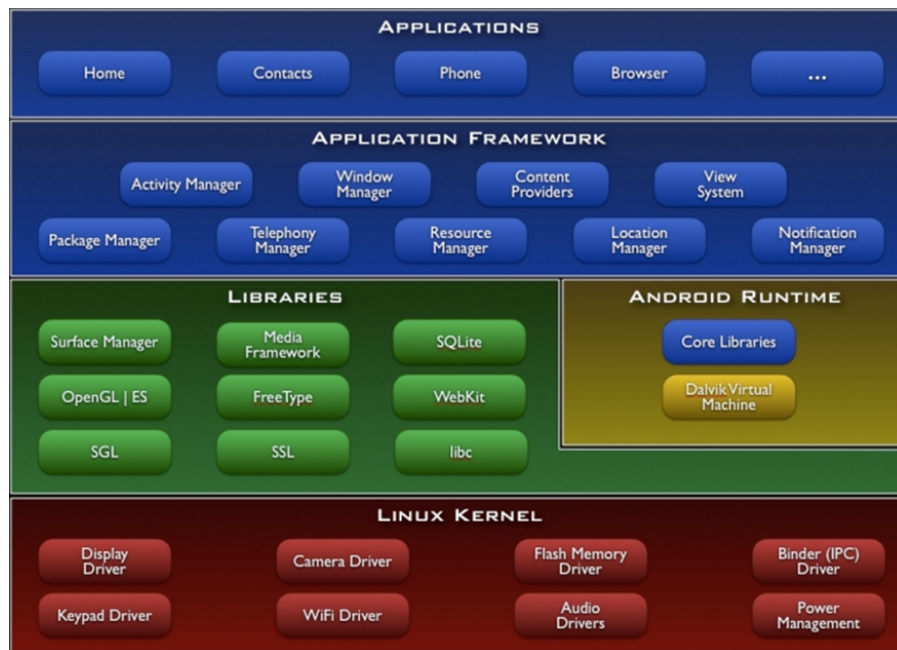


Figura 1: Arquitetura do Android (ANDROID, 2014).

Ao executar um aplicativo na plataforma Android, a máquina Virtual Dalvik, por padrão, realizará a criação de um processo único para o aplicativo que o executará no formato Dalvik Executable, tornando possível executar o aplicativo de forma otimizada (WU; LUO; LUO, 2010).

Atualmente o desenvolvimento de aplicativos voltados para a plataforma Android possui apenas duas IDE para o desenvolvimento: Eclipse e Android Studio (ANDROID, 2014).

### 2.2.1 Aplicativos Android

Aplicativos Android são os programas desenvolvidos para executar sobre a camada de aplicativos, conforme demonstrado na Figura 1, utilizando a linguagem de programação Java. O código fonte, quando compilado, gera os bytecodes Dalvik, que serão executados na Máquina virtual Dalvik e executarão os programas desenvolvidos independentemente do processador utilizado (PEREIRA; SILVA, 2012).

Vale salientar que cada aplicativo desenvolvido para a plataforma Android gera uma nova instância da máquina virtual Dalvik, visando um melhor desempenho e maior integração, possibilitando assim que diversas máquinas virtuais possam ser executadas em paralelo (PEREIRA; SILVA, 2012).

### 2.2.2 Activity

Segundo (GARGENTA, 2011), uma *activity* consiste na interface entre o dispositivo Android e o usuário, a navegação ocorrerá intercalando *activity's* mediante a ação do usuário. É possível realizar uma analogia entre uma aplicação Android e um sistema web, onde cada elemento de visualização dos sistemas web podem ser comparados a uma *activity*, bem como a existência de uma *home page* que equivale a uma *activity main*, geralmente exibida pela primeira vez ao iniciar o aplicativo.

Ainda de acordo com (GARGENTA, 2011), a inicialização de uma *activity* é bastante onerosa, pois envolve a criação de um novo processo no linux, alocação de memória e inicialização do layout. Para que se possa evitar esse desperdício de recursos empregados, o gerenciamento do ciclo de vida de uma *activity* será de responsabilidade do *Activity Manager*.

O *Activity Manager* tem como responsabilidade criar, destruir e gerenciar as atividades da aplicação. Na inicialização do aplicativo, este é o responsável por inicializar a *activity* principal e realizar a alternância entre as *activity's*, movendo a *activity* para um local de armazenamento por um tempo determinado, cuja finalidade é o usuário retornar para a *activity* anterior e esta ser carregada mais rapidamente. Caso contrário, a *activity* será destruída para que se possa liberar espaço, melhorando assim a velocidade da interface do usuário.

Dessa forma, quando abordado o ciclo de vida de uma *activity*, não é possível precisar o seu estado atual. Este ciclo de vida é descrito na Figura 2, onde são mostrados os estados que uma *activity* pode passar.

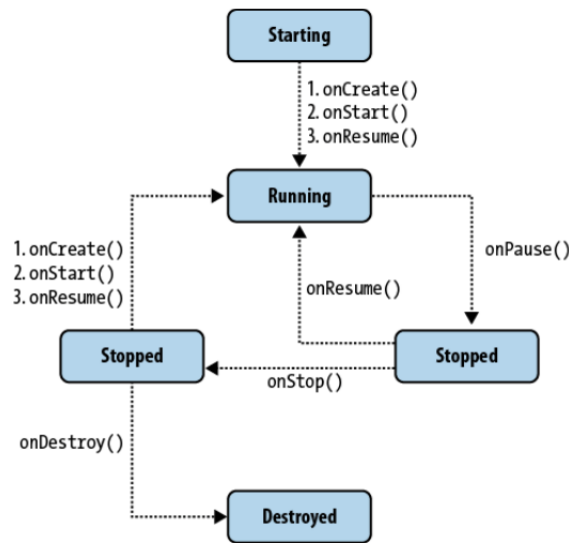


Figura 2: Ciclo de Vida da Activity (GARGENTA, 2011).

### 2.2.3 Intents

Uma *Intents*, segundo (GARGENTA, 2011), consiste na troca de mensagens enviadas entre os principais blocos de construção do aplicativo. Essa troca de mensagens realizada via *Intents* ocorre de forma assíncrona, ou seja, não se faz necessário esperar a conclusão ação.

O utilização de *Intent* pode ocorrer em aplicações concorrentes, onde o sistema deverá requisitar uma determinada informação para que se torne possível a realização de uma determinada ação ou passar alguma determinada informação entre *Intents* de aplicação.

A Figura 3 exemplifica a utilização de uma *Intent* entre aplicações. Inicialmente ocorrerá a comunicação entre a aplicação do correio eletrônico e o browser do dispositivo móvel, que renderizará a informação passada via *Intent*, que, por sua vez, também realizará a comunicação via *Intent* com o sítio armazenador de vídeos.

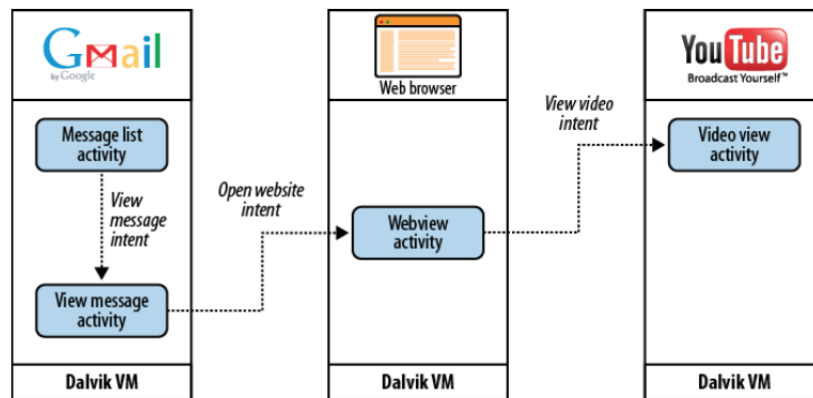


Figura 3: Intent (GARGENTA, 2011).

## 2.2.4 SQLite

Segundo (SQLITE, 2015) o *SQLite* é uma biblioteca que implementa um, sem servidor de banco de dados SQL transacional auto-suficiente. O código para *SQLite* é de domínio público, livre para ser usado em qualquer fim, comercial ou privado. *SQLite* é o banco de dados mais implantado no mundo, com inúmeras aplicações, incluindo vários projetos de alto perfil.

O *SQLite* é um motor de banco de dados SQL embutido, diferentemente da maioria dos outros bancos de dados SQL, ele lê e escreve diretamente para arquivos de disco comuns. Um banco de dados SQL completo com várias tabelas, índices, gatilhos e pontos de vista, está contido em um único arquivo em disco. O formato do arquivo de banco de dados é multi-plataforma, em que é possível copiar livremente um banco de dados entre sistemas de 32 bits e de 64 bits ou entre arquiteturas big-endian e little-endian. Estas características tornam *SQLite* uma escolha popular como um formato de arquivo do aplicativo. Pense no *SQLite* não como um substituto para a Oracle, mas como um substituto para `fopen()`.

O *SQLite* é uma biblioteca compacta, com todos os recursos habilitados em que o tamanho da biblioteca pode ser inferior a 500KiB, dependendo das configurações de plataforma-alvo e de otimização do compilador. Se características opcionais são omitidas, o tamanho da biblioteca *SQLite* pode ser reduzida abaixo 300KiB. *SQLite* também pode ser feito para funcionar em espaço mínimo pilha (4KiB) e muito pouco heap (100KiB), tornando *SQLite* uma escolha popular de banco de dados em memória restrita gadgets como celulares, PDAs e MP3 players. Há uma troca entre o uso de memória e velocidade, o *SQLite* geralmente corre mais rápido quando há mais memória disponível para ele. No

entanto, o desempenho é geralmente muito bom mesmo em ambientes de pouca memória.

O *SQLite* é um banco de dados de código aberto que vem, ao longo do tempo, se tornando bastante estável e popular, principalmente nos dispositivos com menor poder de processamento (GARGENTA, 2011).

### 2.2.5 DbHelper

A plataforma *Android* oferece, de forma original, uma interface para que os aplicativos desenvolvidos possam acessar o *SQLite*. Também está disponível de forma nativa na plataforma *Android* uma outra classe, a classe *helper*, que tem como função fornecer a conexão junto ao banco de dados, denominada *SQLiteOpenHelper* (GARGENTA, 2011).

A classe *DbHelper* tem como finalidade oferecer uma interface de alto nível para as instruções SQL, uma vez que os aplicativos, em sua grande maioria, utilizam apenas as quatro operações básicas: criação, remoção, atualização e leitura dos dados.

### 2.2.6 Notificações

A plataforma *Android* oferece algumas formas de notificação, duas delas serão abordadas nesse capítulo: o *Toast* e o *Notification*. O *Toast* é uma pequena exibição que contém uma mensagem rápida para o usuário, que não persiste porque geralmente é disponível apenas por poucos segundos, por isso nunca recebe o foco. Já o *Notification* é usado para notificar um usuário de que um ou mais eventos tenham ocorrido. Estes eventos podem ser colocado na área de notificação, que está localizada na parte superior da tela. O *Notification* podem conter vários itens de aplicativos distintos e são identificados pelos ícones.

## 2.3 Tecnologias Empregadas

Esta seção tem como finalidade apresentar os conceitos das tecnologias empregadas durante o processo de desenvolvimento do framework.

### 2.3.1 Aplicações Web

As aplicações web são programas executados em um servidor web, fazendo uso da rede de computadores acessada através de um navegador. Esse tipo de abordagem é interessante, por não ser necessário a construção ou atualização do mesmo aplicativo para

as diversas plataformas existentes, porém, uma alteração nesse tipo de abordagem pode ocasionar uma mudança que afetará todos os usuários da aplicação.

Outro ponto a destacar é a manipulação dos dados, pois quando é realizado um processamento, os dados manipulados são salvos remotamente, permitindo assim que os demais usuários, em um outro dispositivo, possam ter acesso ao mesmo dado processado anteriormente. Resaltando ainda que as aplicações web são totalmente dependentes dos navegadores e da rede para que possam funcionar.

### 2.3.2 RestFul Web Services

Os serviços *RESTful* apresentam quatro operações inerentes ao HTTP : *GET*, *POST*, *PUT* e *DELETE*, todas possuindo significados bem definidos. A solicitação *GET* só deve recuperar informações, devendo ser independente, em que cliente deve poder requisitar a URI tantas vezes quanto necessário que não mudará o estado do sistema.

Já o método *POST* consiste no envio da informação presente no seu corpo, para que possa ser realizada a criação do objeto no servidor de aplicação. No que tange ao método *PUT*, este tem como finalidade realizar a atualização do objeto informado. Por fim, o método *DELETE* será usado para remover um recurso no servidor da aplicação.

Em um serviço *RESTful*, a URL proverá os serviços de forma confiável para identificar o recurso a ser utilizado. Os serviços podem ser construídos de duas formas diferentes: ou seguindo o padrão *PathParam* ou *QueryParam*. No *PathParam* o valor do parâmetro é extraído e enviado dentro da URI, enquanto que no *QueryParam* o valor é extraído e fornecido na *query string* da requisição.

### 2.3.3 Banco de Dados Distribuídos

Banco de Dados Distribuídos consiste na capacidade de uma aplicação operar de forma transparente sobre dados dispersos, gerenciados ou não por diferentes SGBD , ainda havendo a possibilidade de ser executado em diferentes plataformas, podendo estar conectadas entre si por diferentes formas de comunicação, operando com se fosse um único SGBD central (DATE, 2004).

A idéia de SGBDD é atrativa sob muitos aspectos. Sob ponto de vista administrativo, é permitido a cada setor uma organização geograficamente dispersa mantenha controle de seus próprios dados, mesmo oferecendo compartilhamento a nível global no uso destes

dados. Do ponto de vista econômico, SGBDDs podem diminuir os custos de comunicações, que hoje em dia tendem a ser maiores do que o próprio custo de equipamento, com o tradicional declínio dos custos de "hardware". Finalmente, SGBDDs também são atrativos sob um ponto de vista técnico, pois facilitam o crescimento modular do sistema, aumentam a confiabilidade através da replicação das partes críticas do banco em mais de um nó e podem aumentar a eficiência através de um critério particionamento e replicação, que coloque os dados próximos do local onde são mais freqüentemente usados (em contraste com acesso remoto a um banco de dados centralizado).

A arquitetura de sistemas utilizando banco de dados tem sido tradicionalmente centralizada, em que o banco de dados reside em um único computador onde são executados todos os programas acessando o banco. Os usuários podem estar ligados diretamente, ou através de uma rede, ao computador onde o banco reside.

Usualmente, alinham-se a favor de sistemas centralizados argumentos que incluem fatores de economia de escala, em que o custo por unidade de trabalho decresce à medida que o tamanho do processador cresce (Lei de Grosch), pela facilidade de controle de segurança, integridade e implantação de padrões, além de disponibilidade de dados para a gerência.

Especificamente, os argumentos que tornam BDDs atrativos podem ser postos da seguinte forma: BDDs podem refletir a estrutura organizacional ou geográfica do empreendimento dando maior autonomia e responsabilidade local ao usuário, mas preservando uma visão unificada dos dados. Do lado tecnológico, o desenvolvimento de redes de comunicação de dados permite a interligação de um grande número de processadores independentes de forma confiável e com custo previsível, podem ser projetados de tal forma a melhorar a disponibilidade e confiabilidade do sistema através da replicação de dados, além de permitirem um crescimento modular da aplicação simplesmente acrescentando novos processadores e novos módulos do banco ao sistema.



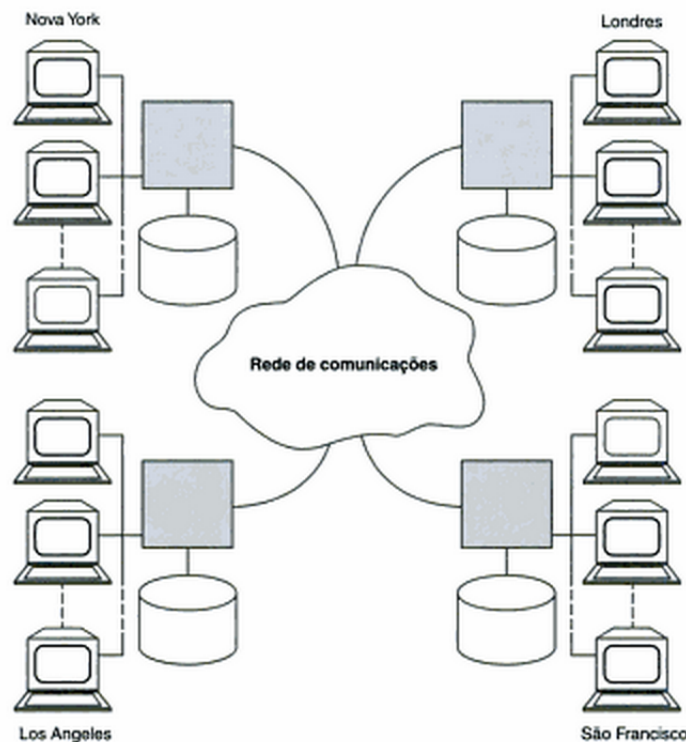


Figura 4: Banco de Dados Distribuído (DATE, 2004).

A Figura 4 exemplifica um típico sistema de banco de dados distribuído, onde é possível constatar que existem quatro bancos replicados se comunicando através da rede de comunicação. Cada um desses bancos de dados é responsável por responder as requisições de determinadas máquinas de forma satisfatória.

## 2.4 Framework

Framework consiste na construção de um conjunto de métodos, tendo como objetivo principal o reuso do seu código. A reutilização se dá no reuso de componentes baseados em interfaces complexas, porém de fácil customização (JOHNSON, 1997).

Os frameworks Java são construídos utilizando interfaces e classes abstratas, onde a sua inicialização pode variar de acordo com a necessidade requisitada. Apresentam pontos fixos, em que serviços já implementados por ele podem realizar chamadas indiretas a determinados serviços ou funcionalidades que devem ser implementadas de acordo com a necessidade do desenvolvedor. Dessa forma é possível dizer que framework é composto de pontos fixos, de pontos de extensão ou ainda de pontos que necessitam de complementação.

A utilização do framework pode trazer alguns benefícios, conforme apresentado por

(FAYAD; SCHMIDT; JOHNSON, 1999):

- *Melhora a modularização* – encapsulando os detalhes voláteis da implementação, fazendo uso de interfaces ou classe abstrata;
- *Aumenta a reutilização* – criando componentes genéricos que podem ser replicados, podendo fazer parte de qualquer novo sistema;
- *Extensibilidade* – fazer uso de métodos chave que permita que as aplicações possam fazer uso de interfaces estáveis;
- *Inversão de controle* – Ocorre quando o código da aplicação é invocado pelo framework, ficando a cargo deste controlar a estrutura e o fluxo de execução dos programas.

## 2.5 Experimento na Engenharia de Software

Segundo (BASILI; SHULL; LANUBILE, 1999), a Engenharia de Software Experimental é um segmento que utiliza experimentos para verificação e validação de teorias, hipóteses e relacionamentos, tendo como consequência a correção e aprimoramento das técnicas utilizadas pela equipe de desenvolvedores de software. A execução de experimentos tem como objetivos a caracterização, a avaliação, a previsão, o controle e a melhoria de produtos, processos, recursos, teorias, modelos, entre outros. Segundo (TRAVASSOS; GUROV; AMARAL, 2002), “somente experimentos verificam teorias, somente experimentos podem explorar os fatores críticos e dar luz ao fenômeno novo para que as teorias possam ser formuladas e corrigidas”.

Segundo (WOHLIN et al., 2000), para a realização de um experimento é necessário que seja definido alguns elementos básicos como: as variáveis independentes (Fatores), as variáveis dependentes (Resultados), objetivos, participantes, contexto, hipóteses e objetos. Cada um deles é descrito a seguir.

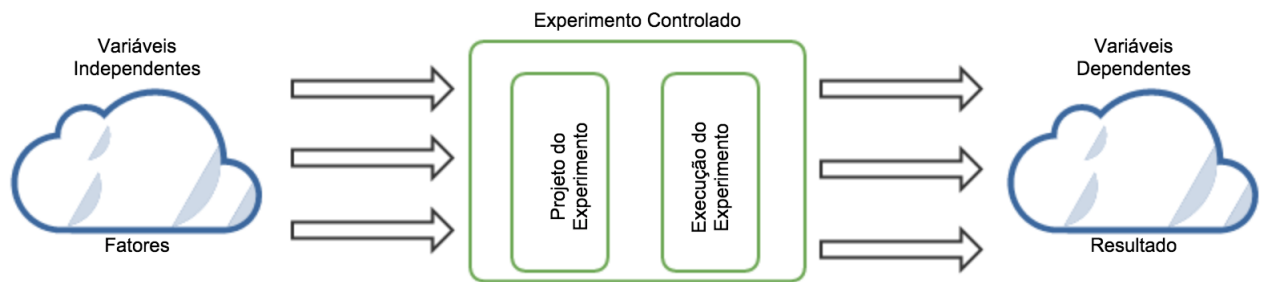


Figura 5: Ilustração de um Experimento.

- *Variáveis Independentes ou fatores*: correspondem aos valores de entrada do experimento, ou seja, representam a causa dos dados do experimento. Essas variáveis são manipuladas e controladas durante todo o processo do experimento (LOPES; TRAVASSOS, 2009), conforme mostrado na Figura 5;
- *Variáveis Dependentes ou Resultado*: correspondem do resultado dos dados ou valores de saída do experimento. Geralmente, só há uma variável dependente em um experimento. Referem-se às variáveis a serem estudadas (LOPES; TRAVASSOS, 2009);
- *Objetivos*: compõem parte da instrumentação do experimento, ou seja, parte do objetivo geral do experimento.
- *Participantes*: são os indivíduos que atuaram no experimento, fornecendo dados, parâmetros para este.
- *Contexto*: composto das condições sobre o qual o experimento é executado, pode ser um ambiente real ou um ambiente simulado.
- *Hipóteses*: suposição que se deseja avaliar.
- *Projeto*: segundo (TRAVASSOS; GUROV; AMARAL, 2002), determina como um experimento deverá ser conduzido. A decisão sobre alocação dos objetos e dos participantes é feita nesse momento, além da definição da maneira como os tratamentos serão aplicados.

### 2.5.1 Organização do Experimento

Segundo (WOHLIN et al., 2000) e (TRAVASSOS; GUROV; AMARAL, 2002) os experimentos devem possuir as metas definidas para assegurar que a intenção do estudo experi-

mental possa ser realizada durante sua execução. Também passa por um planejamento, englobando seleção de contexto, formulação de hipótese, seleção de variáveis, seleção de participantes, projeto de estudo experimental, instrumentação e validação. Os projetos de estudos experimentais possuem alguns princípios gerais de organização:

- *Aleatoriedade*: métodos estatísticos utilizados com a finalidade de analisar os dados e garantir variáveis independentes e aleatórias. A principal contribuição da aleatoriedade é evitar que um determinado valor interfira em outro e também para a seleção dos participantes do experimento;
- *Agrupamento*: tem como principal contribuição a eliminação dos efeitos indesejados do estudo, para aqueles onde o fator sobre o qual não há interesse e que pode vir a influenciar sobre a resposta do experimento. O agrupamento geralmente é utilizado quando existe um valor não esperado no experimento que influencia sobre o resultado, ocasionando assim informações incorretas.
- *Balanceamento*: atribui para cada tratamento um número igual de participantes, contribuindo para a melhora da análise estatística;

Com a execução do experimento, obtém-se a medição e, com ela, as medidas de um experimento são utilizadas para que seja possível analisar o seu resultado. As hipóteses levantadas são verificadas e um dimensionamento do problema é obtido. O resultado das medidas chama-se medição e pode pertencer às escalas nominal, ordinal, intervalo ou razão (TRAVASSOS; GUROV; AMARAL, 2002).

## 3 Proposta do Framework Offdroid

Atualmente, as aplicações móveis possuem grande dependência pela conectividade, por isso o framework *OffDroid* foi projetado e implementado para fornecer um conjunto de funções de persistência de dados, replicação e de sincronização, possibilitando o funcionamento do dispositivo móvel no modo off-line.

Ele possui as funcionalidades de criação, remoção, atualização e visualização dos dados armazenados ou solicitados através do serviço REST.

Permitindo que as requisições sejam capazes de executar operações sem a necessidade de conexão com a internet, fazendo apenas uso da base de dados local, o que permite que o aplicativo tenha mais cobertura, abrangendo tanto os usuários com acesso contínuo à rede como aqueles que não têm acesso em todos os momentos.

Neste capítulo será abordada a construção do framework, desde a definição dos requisitos funcionais e não funcionais, construção da arquitetura do framework e a implementação da solução proposta.

### 3.1 Requisitos

Na construção do framework foi implementado as funções de persistência, sincronização dos dados e replicação, funções serão necessárias para possibilitar o uso do aplicativo quando o dispositivo estiver no modo offline.

O objetivo principal do modelo de aplicação consiste no desenvolvimento de aplicações móveis onde ao apresentar conectividade com a internet, o aplicativo realize as requisições junto ao servidor de aplicação para que, na ausência de conectividade, o mesmo realize as consultas, cadastros, remoções e atualizações no banco de dados do dispositivo. Sem que houvesse a necessidade da realização da escalabilidade do banco de dados do dispositivo.

O desenvolvimento do framework sofreu forte influência do JPA, tendo como limitação

os tipos de dados existente no banco de dados *SQLite*, caso o desenvolvedor não quiser ficar limitado aos tipos de dados presentes no *SQLite* pode modificar a forma de persistir os dados, realizando uma nova implementação da classe responsável pela persistência.

O modelo original da aplicação não apresentava nenhuma preocupação quando a resolução de conflitos, para o framework a informação retornada do servidor de aplicação sempre deveria a informação que irá prevalecer.

Mediante interações com os desenvolvedores dos aplicativos móveis da SINFO, foram levantados os requisitos funcionais e não funcionais que o framework deveria apresentar. Por se tratar de um framework para dispositivos móveis, alguns requisitos não funcionais deveriam ser atendidos, pois trata-se de características e aspectos internos do sistema que envolve a parte técnica. Os requisitos não funcionais levantados foram:

- **Desempenho:** O framework deverá ter um baixo tempo de resposta para as operações de cadastro, leitura, atualização e remoção, que não deverá ser superior à media atual que é de 250 milissegundos cada operação. O desempenho foi o padrão estabelecido mediante os estudo realizado por meio de um experimento controlado;
- **Confiabilidade:** O framework deverá ser capaz de contornar diversos eventos inesperados, como por exemplo, se ocorrer timeout ele deverá ser capaz de realizar a mesma operação na base de dados do dispositivo; ao tentar realizar uma requisição junto ao servidor de aplicação e esse não responder dentro de tempo máximo definido, o framework realizará a mesma operação na base de dados do dispositivo móvel para que o aplicativo;

Já os requisitos funcionais expressam funções ou serviços que um software deve ser capaz de executar ou fornecer, que em geral são, processos que utilizam entradas para produzir saídas. Por esse motivo foram levantados os seguintes requisitos funcionais:

- **Criação do banco:** O framework deverá ser capaz de criar o banco de dados para o armazenamento das informações das entidades mapeadas.
- **Cadastrar:** O sistema deverá ser capaz de realizar o cadastro das entidades mapeadas no framework;
- **Buscar:** O sistema deverá ser capaz de realizar a consulta das entidades mapeadas no framework com base nas anotações presentes na classe;

- **Remover:** O sistema deverá ser capaz de realizar a remoção das entidades mapeadas no framework.
- **Atualizar:** O sistema deverá ser capaz de realizar a atualização das entidades mapeadas no framework.
- **Comunicação com o servidor:** Na presença de conectividade o sistema deverá ser capaz de realizar a comunicação com o servidor.
- **Sincronização dos dados:** A sincronização é bilateral, ocorrendo nas duas direções, tanto entre o servidor de aplicação e o dispositivo móvel, quanto entre o dispositivo móvel e o servidor de aplicação. A primeira ocorre quando o dispositivo móvel realiza alguma requisição REST junto ao servidor, retornando um dado que será armazenado na base de dados local do dispositivo móvel. A segunda sincronização só ocorrerá se algum dado presente na base de dados do dispositivo móvel sofrer alguma modificação para que a alteração seja enviada para o servidor de aplicação.

## 3.2 Arquitetura

Segundo (SHOSHANI et al., 1996), a utilização de reflexão em um framework pode torná-lo mais adaptável a diferentes contextos e aplicações, sendo capaz de implementar a interface do framework ou prolongar uma de suas superclasses. Implementações mal realizadas nas classes estendidas do framework podem ocasionar a quebra do código da aplicação. Dessa forma é possível constatar que quanto menor for o acoplamento, mais facilmente será o enquadramento necessário para evoluir de forma independente da aplicação e vice-versa.

Durante o processo de desenvolvimento do framework foi utilizado alguns padrões de projetos, como o padrão estrutural, que tem como objetivo se preocupar como as classes e objetos são compostos, ou seja, como é a sua estrutura. O objetivo destes padrões é facilitar o design do sistema identificando maneiras de realizar o relacionamento entre as entidades, deixando o desenvolvedor livre desta preocupação.

Durante o processo de desenvolvimento foram utilizados padrões de projeto estrutural, que tem como finalidade se preocupar em como as classes e objetos são estruturados, este padrão facilita o design do sistema identificando o relacionamento entre as entidades. Dentre os padrões estrutural existentes, foram utilizados os padrões bridge e façade.

Já quanto o padrão comportamental, que atua sobre como responsabilidades são atribuídas as entidades, ou seja, qual o comportamento das entidades, que facilitam a comunicação entre os objetos distribuindo as responsabilidades e definindo a comunicação interna. Dentre os padrões comportamentais existentes, foram utilizados os padrões Chain of Responsibility e Strategy.

Já os padrões de criação tem como intenção principal abstrair o processo de criação de objetos, ou seja, a sua instanciação, não precisando se preocupar com questões sobre como o objeto é criado, como é composto, qual a sua representação real. Isto quer dizer que se, ocorrer alguma mudança neste ponto, o sistema em geral não será afetado, alcançando flexibilidade que os padrões de projeto buscam. Dentre os padrões de criação existentes fora utilizado os padrões prototype e singleton.

Com base nesse entendimento acima citado, o framework foi desenvolvido fazendo uso intenso de reflexão e utilização de interfaces, passando a ter suas funcionalidades facilmente estendidas para os diversos cenários. Para facilitar sua evolução de forma independente, reduzindo seu acoplamento, o framework foi dividido em quatro módulos: Utils, Comunicação com o servidor, Persistência e Sincronização. Os módulos da arquitetura são autônomos e cooperantes entre si. Cada um dos módulos supracitado será exemplificado em detalhes nas seções que seguem.

A arquitetura de framework é apresentada na Figura 6.



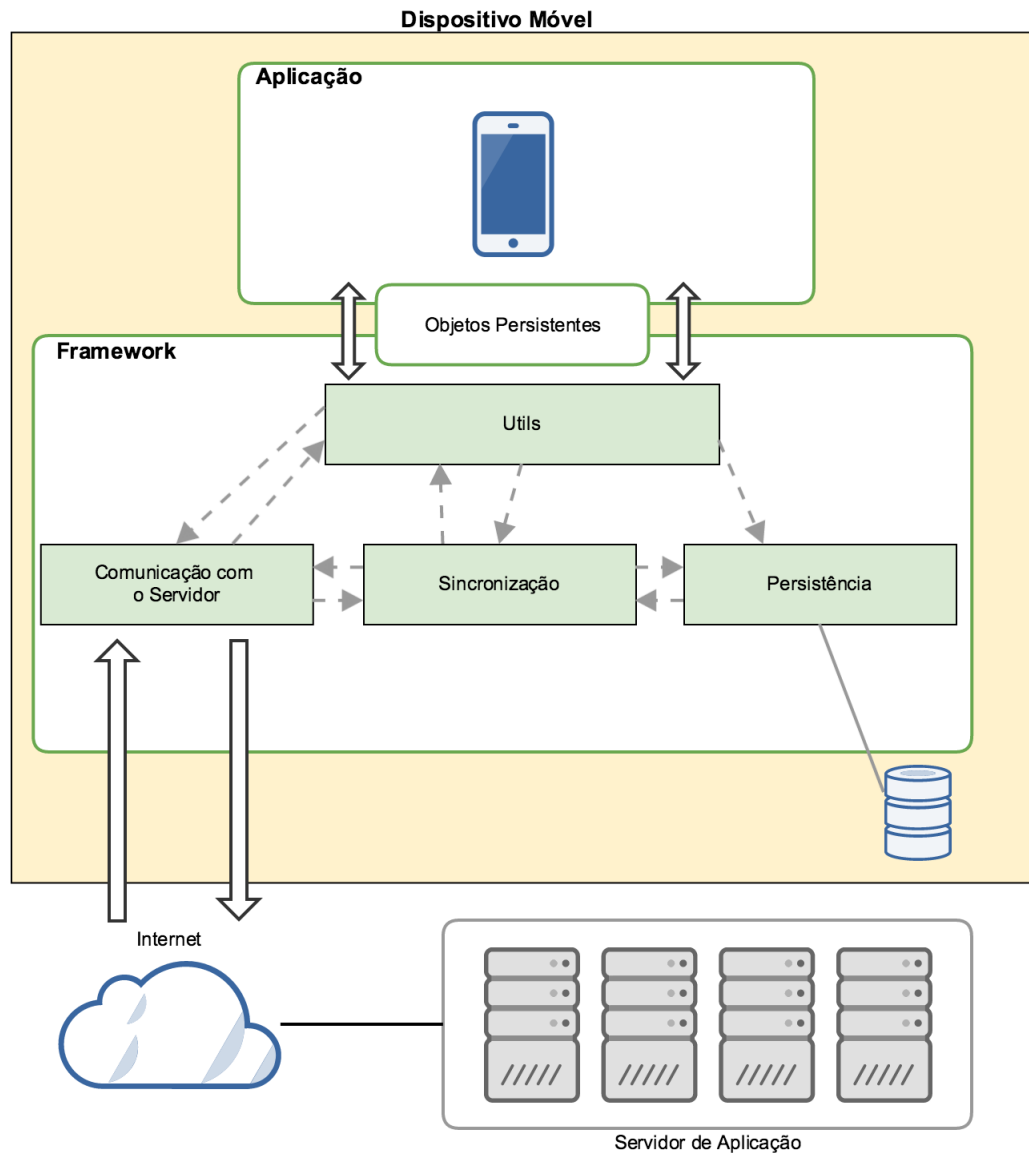


Figura 6: Arquitetura de Software

### 3.2.1 Utils

O módulo **Utils**, que se relaciona diretamente com a aplicação desenvolvida, faz uso do padrão de projeto *facade* definindo as operações a serem realizadas nos demais módulos. Dessa forma é possível definir uma operação padrão, como por exemplo, de geração do banco de dados ou de persistência dos dados, evitando a necessidade de chamar os métodos

responsáveis a cada nova solicitação. Esse módulo contém três classes na sua estrutura: *OffLineManager*, *NotificationUtils* e *NetWorkUtils*, conforme apresentado na Figura 7.

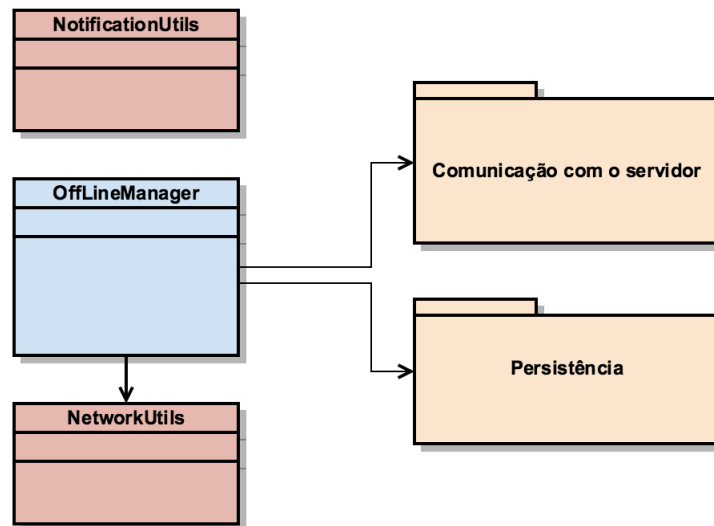


Figura 7: Relacionamento entre as Classes do Utils e os demais Módulos

A classe *NotificationUtils* é tida como autônoma, pois não está relacionada com nenhuma outra classe do módulo, sendo ela responsável por exibir as notificações geradas para o usuário na forma *Toast* ou *Notification*. Já a classe *NetWorkUtils* é responsável por verificar o estado da conectividade do dispositivo para auxiliar na tomada de decisão, sendo esta feita sempre que uma nova operação é solicitada. A classe *OffLineManager* funciona como uma fachada se comunicando com os demais módulos do framework, simplificando as dependências entre eles e fazendo com que se comuniquem uns com os outros, exclusivamente através das suas fachadas.

A classe *OffLineManager* irá se comunicar com o módulo de Persistência ou de Comunicação com o Servidor, dependendo da estratégia de configuração adotada pela classe. Se a requisição for para o módulo Comunicação com o Servidor, esse terá a responsabilidade de realizar a requisição no servidor de aplicação, se a requisição apresentar sucesso, o módulo Comunicação com o Servidor irá se comunicar com a fachada do módulo de Persistência, enviando o dado retornado. Já se a requisição for para o módulo de Persistência, esse terá a finalidade de atualizar a base de dados do dispositivo e sinalizar se o dado precisa ser sincronizado com o servidor de aplicação.

Por fim, a classe *OffLineManager* tem como atribuição verificar, na presença de co-

nectividade, se há algum dado que precisa ser sincronizado entre o dispositivo e o servidor, se houver, a classe iniciará o processo de sincronização de forma assíncrona.

### 3.2.2 Comunicação com o Servidor

Quanto a comunicação com o servidor da aplicação, o padrão utilizado foi o protocolo REST, por ser muito comum em aplicações móveis. Foram encontradas algumas API, tais como *RESTDroid*<sup>1</sup> e *Retrofit*<sup>2</sup>, que tinham como propósito encapsular as requisições, simplificando as chamadas aos web services REST.

Tais APIs foram encontradas e estudadas, entretanto, por serem insatisfatórias, foram descartadas. Estas foram consideradas insatisfatória por não apresentarem uma vasta documentação nem uma fácil customização, tendo sido desenvolvido como solução a criação de três classes: *WebServiceImpl*, *JSONProcessor*, *HttpClientSingleton*. Estas realizam o envio e o recebimento das requisições entre o dispositivo móvel e o servidor de aplicação, conforme apresentado da Figura 8.

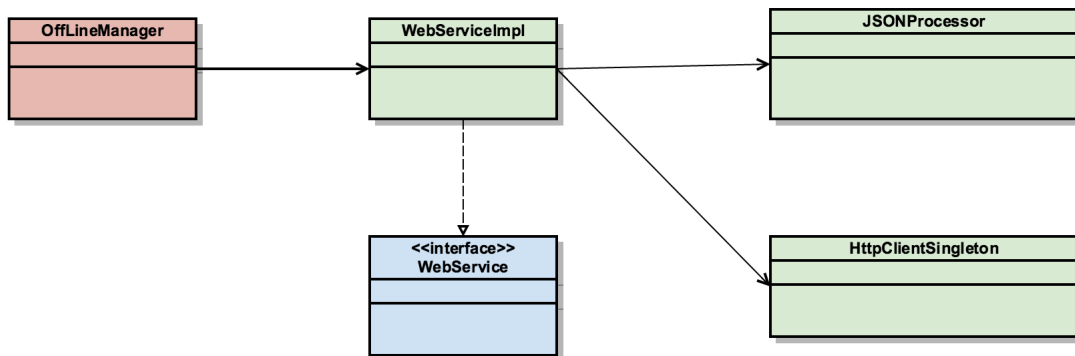


Figura 8: Módulo Comunicação com o Servidor

A Figura 8 apresenta as classes que compõem o sistema de envio e de recebimento das requisições entre o dispositivo móvel e o servidor de aplicação. O processo é inicializado quando utilizado alguns dos métodos existente na classe *OffLineManager*, presente no módulo Utils, que tem como objetivo encaminhar a requisição para a classe *WebServiceImpl*, cuja função é realizar a requisição junto ao servidor de aplicação.

A classe *WebServiceImpl*, que consiste na implementação da interface *WebService*, contempla uma série de métodos existentes que já estão predefinidos, tais como: get,

<sup>1</sup><https://github.com/PCreations/RESTDroid>

<sup>2</sup><http://square.github.io/retrofit/>

post, put e delete. Se houver a necessidade, o desenvolvedor poderá realizar uma nova implementação da classe que se adeque a sua realidade, basta que realize uma nova implementação da interface *WebService* e dos seus métodos ali contidos. Pode ser modificado também o protocolo a ser utilizado nas trocas de mensagens entre o servidor de aplicação e o dispositivo móvel.

A classe *HttpClientSingleton* apresenta diversos atributos, que define desde o tempo de espera do serviço (timeout) até o tempo de espera do socket, adota como padrão de projeto o singleton, que consiste em manter uma única instância da classe durante todo o uso do aplicativo. Caso seja necessário modificar o timeout basta modificar os valores dos atributos presentes na classe. Já a classe *JSONProcessor* tem como atribuição realizar a serialização dos objetos e a de serialização do JSON para os objetos persistentes.

### 3.2.3 Persistência

Quanto a Persistência, inicialmente foi realizado um estudo com o intuito de encontrar alguma API que apresentasse características como: a realização da persistência de forma não intrusiva; a capacidade de realizar as operações básicas, como criar, remover, atualizar e buscar; realizar a criação das tabelas e do banco de dados quando inicializado; e, principalmente, que apresentasse licença permitindo total liberdade para realizar as modificações necessárias.

No estudo foram encontradas algumas API, tais como *h4Android*<sup>3</sup>, *OrmLite*<sup>4</sup>, *Sugar ORM*<sup>5</sup> e o *andOrm*<sup>6</sup>, que apresentavam todas essas características ou parte delas, sendo então realizado um estudo mais detalhado nestas APIs, com a finalidade de analisar e comprovar se tudo o que era prometido seria de fato cumprindo.

A API selecionada foi o *h4Android*, que tem como características a persistência mapeamento objeto relacional, relacionamento entre as classes, a criação das tabelas do banco de dados e as operações básicas de criação, remoção, atualização, busca e não seja intrusiva. Porém, apesar desta apresentar grande parte dos objetivos almejados, ainda assim foi necessário realizar algumas modificações no *h4Android* para que trabalhasse conforme o planejado e fosse incorporado ao framework.

Foram realizadas as seguinte modificações:

---

<sup>3</sup><https://code.google.com/p/h4android/wiki/h4Android>

<sup>4</sup><http://ormlite.com/>

<sup>5</sup><http://satyan.github.io/sugar/>

<sup>6</sup><https://github.com/jonatasdaniel/andorm>

- Implementação do *insert* em cascata das classes mapeadas;
- Implementação do *update* em cascata das classes mapeadas;
- Implementação do *findAll* em cascata das classes mapeadas, que consiste na consulta de todos os dados da Classe em questão;
- Implementação do *find*, sendo possível informar quais os atributos devem ser considerados no momento da busca, realizando esta em cascata nas classes mapeadas, em que todas as entidades das classes serão retornadas;
- O método *updateAll* foi desenvolvido, uma vez que o mesmo não apresentava implementação padrão do *h4Android*. A sua utilização também ocorrerá na forma de cascata nas classes mapeadas;
- Criação da interface, denominada *PersistDB*, que se faz necessária para a persistência dos dados, sendo necessária para a definição da chave primária de todas as classes mapeadas.

A Figura 9 exemplifica as classes que são responsáveis pela persistência do framework.

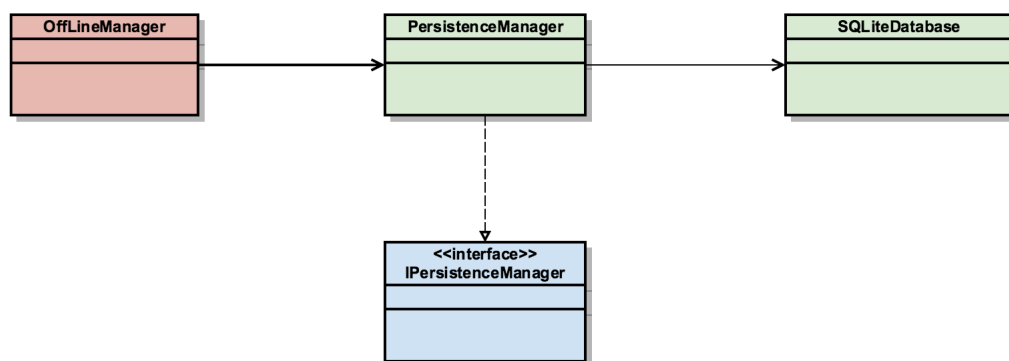


Figura 9: Módulo de Persistência

A classe *OffLineManager*, responsável por delegar as requisições da aplicação, irá delegar a função de persistência para classe *PersistenceManager*, que apresenta os métodos necessários para a realização das operações básicas de persistências dos dados no *SQLite*, utilizando o mapeamento objeto relacional, padrão esse adotado pelo framework.

Todos os métodos existentes na classe *PersistenceManager* estão presentes na interface *IPersistenceManager*, podendo ser reimplementada pelo desenvolvedor caso a mesma não apresente as operações conforme desejado. Referente a classe *PersistenceManager*, esta

tem como herança a classe *SQLiteDatabase* que provê a comunicação de acesso a base de dados *SQLite*.

### 3.2.4 Sincronização

Quanto a sincronização, o framework adota como padrão a sincronização automática e assíncrona. Durante o uso do aplicativo, ao realizar uma operação, o framework em background verificará se há algum dado que necessite ser sincronizado junto ao servidor de aplicação, sendo constatado esta necessidade, serão carregados todos os dados presentes na tabela sincronização e será inicializado o processo. A sincronização é bilateral, podendo ser realizada do servidor de aplicação para o dispositivo móvel, durante as requisições REST ou durante a sincronização do dado alterado quando offline para o servidor.

A Figura 10 exemplifica as classes responsáveis pelo mecanismo padrão de persistência dos dados que irão ser sincronizados.

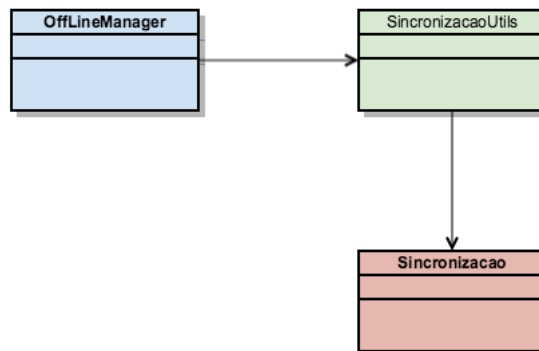


Figura 10: Classe responsável pelos dados a serem sincronizados

A classe *OffLineManager*, por ser a principal classe do framework, tem como responsabilidade delegar a realização de determinadas operações, uma delas é que seja realizada a sincronização dos dados, que invocará o método *sync*, sempre que for detectado conectividade, este está presente na classe *SincronizacaoUtils*, tendo como finalidade verificar se há algum registro que necessita ser sincronizado e atualizado na base de dados local do dispositivo. O serviço de Sincronização está intrinsecamente ligado com o Módulo Persistência e com o Módulo Comunicação com o servidor.

### 3.3 Estratégias de Configuração

Durante a concepção do framework foram definidas três possíveis estratégias para a manipulação das classes de domínio, tendo como finalidade informar para o framework quais módulos deverão ser utilizados durante a manipulação da classe em questão. A primeira estratégia possibilita a obtenção de dados apenas no contexto do dispositivo móvel que apresentarem a anotação *@OnlyLocalStorage*. Na segunda, os dados só existirão no contexto onde o dispositivo apresente conectividade, que apresentarão a anotação *@OnlyOnLine*. Já na terceira, os dados devem fazer parte dos dois contextos, tanto do dispositivo móvel quanto do servidor de aplicação, nesse caso as classes de domínio não requerem nenhuma anotação específica.

As três possíveis estratégias para as classes de domínio são:

- **Apenas Online:** Essa estratégia deverá ser aplicada na classe de domínio quando não se julgar necessário realizar a persistência da classe para o banco de dados do dispositivo móvel. Neste caso, a classe não apresentará representatividade no banco de dados do dispositivo móvel e esse objeto só poderá ser preenchido quando o dispositivo apresentar conectividade. Para fazer uso dessa condição basta que o desenvolvedor utilize anotação *@OnlyOnLine*;
- **Apenas Local:** Essa estratégia deve ser usada quando a classe apresentar representatividade somente no banco de dados do dispositivo móvel. Essa classe não deverá ser sincronizada nem enviada para o servidor da aplicação. Para que se possa fazer uso dessa estratégia é necessário que o desenvolvedor faça uso de uma anotação específica *@OnlyLocalStorage*.
- **Local e OnLine:** A terceira estratégia prevista no desenvolvimento do framework foi a necessidade dos dados apresentarem representatividade no banco de dados do dispositivo móvel e esses mesmos dados serem sincronizados e enviados para o servidor de aplicação. Por julgar que essa seria a condição padrão utilizada pelos desenvolvedores que fazem uso do framework, não se faz necessário realizar nenhuma anotação para que essa estratégia possa ser utilizada.

A forma como o framework tinha sido desenvolvido irá definir o tipo de estratégia de configuração na presença ou na ausência de conexão para a operação de Inserção conforme apresentado na Tabela 1.

Tabela 1: Tabela com as possíveis combinações do Insert

	OnlyOnLine		OnlyLocalStorage		Default	
	OnLine	OffLine	OnLine	OffLine	OnLine	OffLine
INSERT	Realizar a requisição (PUT)	Lança uma Exceção	Realizar a operação no banco local	Realizar a operação no banco local	Tentará realizar a requisição (PUT); Se retornar o registro o framework realizará o insert no banco no banco local	Realizará a operação de insert no banco local; Marca o registro para sincronização.

Já a Tabela 2 representa como o framework desenvolvido irá se comportar para cada tipo de estratégia de configuração, na presença ou na ausência de conexão, para a operação de Consulta.

Tabela 2: Tabela com as possíveis combinações da Busca

	OnlyOnLine		OnlyLocalStorage		Default	
	OnLine	OffLine	OnLine	OffLine	OnLine	OffLine
CONSULTA	Realizar a requisição (GET)	Lançar uma Exceção	Realizar a operação no banco local	Realizar a operação no banco local	Tentará realizar a requisição (GET); Se trouxer dado tentará atualizar a base local.	Realizar a operação no banco local



Já a Tabela 3 representa como o framework desenvolvido irá se comportar para cada tipo de estratégia de configuração, na presença ou na ausência de conexão, para a operação de Atualização.

Tabela 3: Tabela com as possíveis combinações do Update

	OnlyOnLine		OnlyLocalStorage		Default	
	<b>OnLine</b>	<b>OffLine</b>	<b>OnLine</b>	<b>OffLine</b>	<b>OnLine</b>	<b>OffLine</b>
ATUALIZAÇÃO	Realizar a requisição (POST)	Lançar uma Exceção	Realizar a operação no banco local	Realizar a operação no banco local	Tentará realizar a requisição (POST); posteriormente operação de atualização no banco local	Realizará a operação de atualização no banco local; Marca o registro para sincronização.

Enquanto que a Tabela 4 representa como o framework desenvolvido irá se comportar para cada tipo de estratégia de configuração, na presença ou na ausência de conexão, para a operação de remoção.

Tabela 4: Tabela com as possíveis combinações do Delete

	OnlyOnLine		OnlyLocalStorage		Default	
	OnLine	OffLine	OnLine	OffLine	OnLine	OffLine
DELETE	Realizar a requisição (DELETE)	Lançar uma Exceção	Realizar a operação no banco local	Realizar a operação no banco local	Tentará realizar a requisição (DELETE); posteriormente operará a requisição de delete no banco local	Realizará a operação de delete no banco local; Marca o registro para sincronização.

### 3.4 Mapeamento das Classes

No que se refere ao mapeamento das classes, as anotações foram baseadas na API de persistência JPA <sup>7</sup>, que consiste num padrão encontrado nos principais frameworks de persistência, tais como o Hibernate, o Oracle TopLink e o Java Data Objects (BISWAS; ORT, 2006). Porém, para atender o cenário da computação móvel, foram realizadas algumas simplificações para que fossem atendidas as necessidades do framework, onde determinados tipos de dados, por não apresentarem compatibilidade no *SQLite*, foram descartados. Essa simplificação ocorreu diante do fato da plataforma utilizada apresentar um baixo processamento em relação a um servidor de aplicação.

```

1 @Entity
2 @Table("atividade")
3 public class Atividade implements PersistDB {
4
5     @Id
6     private Integer id;
7
8     @Column(name="codigo", length=14, allowNulls=false, typeColumn =
          TypeColumn.VARCHAR)

```

<sup>7</sup><http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

```

9      private String codigo;
10
11      @Column(name="titulo", length=14, allowNulls=false, typeColumn =
        TypeColumn.VARCHAR)
12      private String titulo;
13
14      @ManyToOne
15      @Column(name="id_evento", typeColumn = TypeColumn.INTEGER)
16      private Evento evento;
17
18      @ManyToOne
19      @Column(name="id_tipo_atividade", typeColumn = TypeColumn.
        INTEGER)
20      private TipoAtividade tipoAtividade;
21
22      @Transient
23      private int idResponsavel;

```

Listagem 3.1: Exemplo de Mapeamento de uma Classe de Domínio

A Listagem 3.1 exemplifica o mapeamento de uma classe que será gerenciada pelo framework. Antes de exemplificar o uso das anotações, deve ser ressaltado que toda classe de domínio deve realizar a implementação da interface *PersistDB*. Quanto ao uso das anotações, foi realizado a divisão destas em dois grupos para uma melhor explanação: anotações aplicadas a classe persistente e anotações inerentes aos atributos que compõem a classe.

As anotações aplicadas à classe persistente são *@Table*, *@Entity*, *@OnlyOnLine* e *@OnlyLocalStorage*. A anotação *@Entity* informa para o framework que essa classe terá representatividade no banco de dados local do dispositivo. A anotação *@Table*, que serve como um complemento da anotação anterior, tem como finalidade informar o nome da tabela do banco de dados que será criada para o armazenamento das informações da entidade. As duas outras anotações apresentadas já foram anteriormente abordadas.

As anotações inerentes aos atributos que compõem a classe são *@Id*, *@Column*, *@ManyToOne* e *@Transient*. A anotação *@Id* serve para indicar que o atributo anotado será a chave primária da classe, este terá uma restrição cadastrada de forma automática pelo framework, impossibilitando assim que se possa ter uma duplicidade de uma mesma chave primária. A anotação *@Column* tem como finalidade informar o nome que o atributo deverá ser referenciado no banco de dados, bem como o seu tamanho máximo e o tipo de dado armazenado. A anotação *@ManyToOne*, combinada com a *@Column*, terá a finali-

dade de indicar que a classe atual apresenta um relacionamento, isto é, que a classe atual armazena uma referência para a classe do relacionamento. Desse modo, conforme informando anteriormente, todas as operações realizadas nesta classe também serão realizadas nas classes mapeadas. No que tange a anotação *@Transient*, esta indica ao framework que o dado contido nesse atributo não deverá ser persistido nem criado uma coluna para o armazenamento da informação no banco de dados do dispositivo.

### 3.5 Comportamento do Framework

O framework tem como comportamento padrão realizar as requisições aos serviços REST quando apresentar conectividade. Já diante da ausência de conectividade, essa a realização da operação ocorrerá no banco de dados do dispositivo móvel.

O framework disponibiliza um conjunto de métodos pré-definidos cuja finalidade é auxiliar o desenvolvedor na construção de um aplicativo que apresente como requisito funcional a possibilidade de trabalhar tanto online quanto offline.

```

1  Atividade atividade = (Atividade) listView().getItemAtPosition(
    position);
2  Participacao participacao = new Participacao();
3  participacao.setAtividade(atividade);
4
5  try {
6      List<Atividade> atividades = (List<Atividade>) OfflineManager.
          findAllExactyField(participacao, "atividade.id");
7  } catch(Exception e) {
8      e.printStackTrace();
9  }

```

Listagem 3.2: Consulta de Dados de uma Classe @OnlyOnline

Na Listagem 3.2 é apresentado um exemplo da realização de uma consulta utilizando o framework. Caso o dispositivo apresente conectividade e a classe apresente a anotação *@OnlyOnline*, o framework irá realizar requisição junto ao servidor de aplicação, e, posteriormente, apresentará para o usuário os dados retornados da requisição. Para esse caso em particular não será persistido os dados retornados, uma vez que a classe em questão apresenta a anotação *@OnlyOnline*, onde os dados a ela pertencentes só fazem sentido quando requisitados ao servidor da aplicação. Para melhor exemplificar, a Figura 11 descreve o funcionamento através de uma diagrama de sequência, onde estão descritos todo o passo-a-passo explanado.

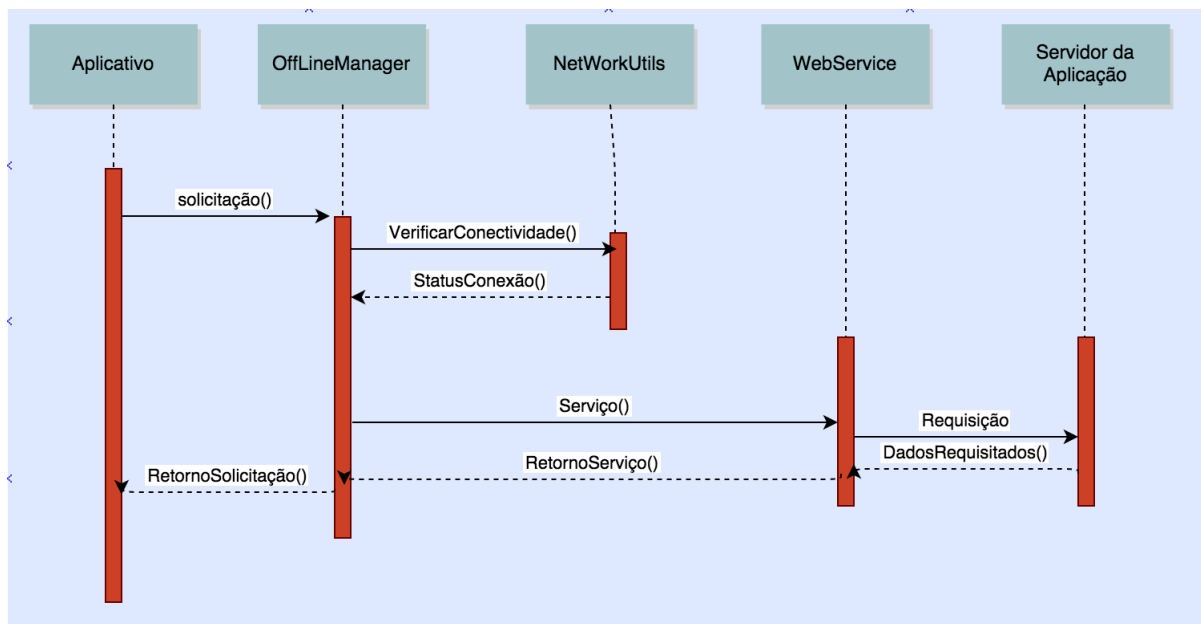


Figura 11: Diagrama de Sequência *@OnlyOnline*

Se a classe apresentar a anotação *@OnlyLocalStorage*, o framework não realizará uma requisição ao servidor da aplicação, pois os dados pertencentes a essa classe só fazem sentido quando consultados na base de dados do dispositivo móvel.

Com o intuito de coletar as informações desejadas, o desenvolvedor deverá realizar o mapeamento da classe indicando que os dados da mesma estão presentes apenas na base de dados do dispositivo. Dessa forma, toda consulta da classe que apresentar a anotação *@OnlyLocalStorage*, independente da presença de conectividade, terá a operação realizada no banco de dados do dispositivo. Para melhor exemplificar, a Figura 12 demonstra o funcionamento através de uma diagrama de sequência, onde estão descritos todo o passo-a-passo explanado.

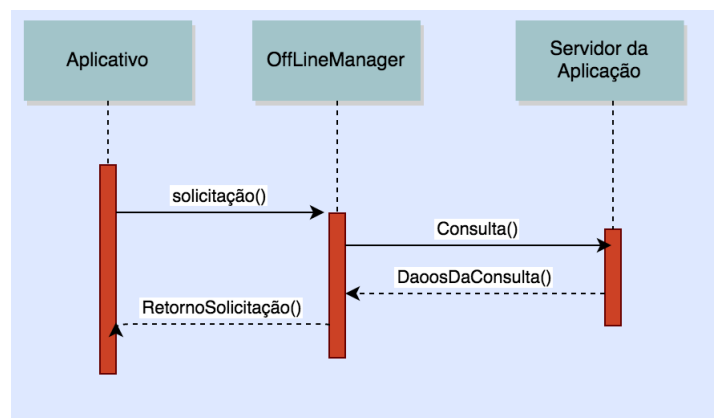


Figura 12: Diagrama de Sequência *@OnlyLocalStorage*

Quando o dispositivo apresentar conectividade e a classe não apresentar as anotações *@OnlyOnLine* ou *@OnlyLocalStorage*, o framework poderá realizar diversas requisições, inclusive a do tipo *GET*, junto ao servidor de aplicação. Havendo retorno nesta requisição, o framework realizará a mesma requisição na base local do dispositivo móvel com o intuito de verificar se há algum dado novo ainda não armazenado, conforme demonstrado na Figura 13.

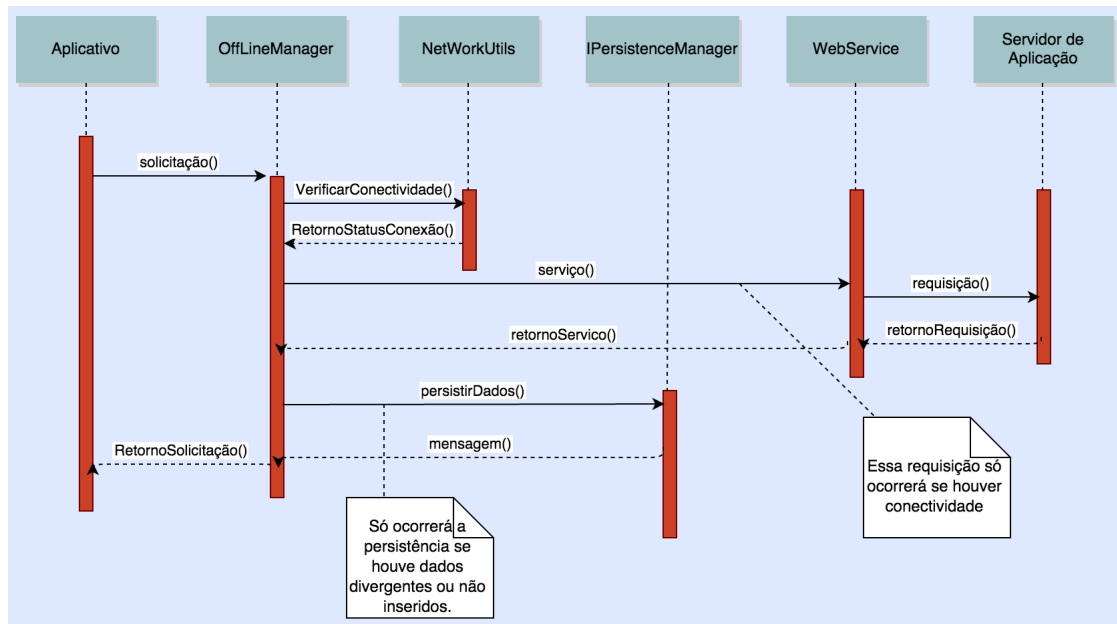


Figura 13: Diagrama de Sequência para as Classes sem anotação

Para saber se os dados já estão presentes na base interna no dispositivo móvel, ocorrerá uma verificação utilizando os métodos equals e hashCode presentes na classe da requisição. Estando os dados presentes e apresentando alguma informação desatualizada, o framework realizará a atualização dos dados na base de dados interna do dispositivo, porém, se o dado não estiver presente na base de dados interna do dispositivo, o framework realizará a persistência da entidade e de seus relacionamentos no banco de dados.

Todas as operações de criação e atualização realizadas no dispositivo, quando o mesmo estiver no modo offline, resultará na criação de um registro no banco de dados na tabela sincronização. Ao retornar a conectividade, registros presentes no banco de dados serão lidos para que seja realizada a sincronização junto ao servidor de aplicação dos dados presentes na tabela.

## 3.6 Classes Utilitárias

Nessa seção será abordado as classes utilitárias *EqualsUtils* empregados na construção do framework na verificação da igualdade entre os registros presentes na base de dados do dispositivo móvel e dos dados retornados do serviço.

### 3.6.1 EqualsUtils e hashCodeUtils

Para que seja possível realizar as operações fornecidas pelo framework, criação, atualização, remoção e busca, é necessário que as classes de domínio realizem a implementação da interface *PersistDB* que, dentre outros métodos, fazem com que o desenvolvedor possa implementar na classe de domínio os métodos *Equals* e *HashCode*.

O *Equals* tem como função definir a relação de igualdade e comparação dos objetos, fazendo uso das referências presentes na classe. Já o *HashCode*, que apresenta funcionalidade similar ao *Equals*, tem a sua implementação consistente em converter o endereço interno do objeto em um número inteiro (AFFILIATES, 2006).

Os métodos *equals* e *hashCode* têm como finalidade verificar se o objeto que está realizando a operação já se encontra cadastrado no banco de dados do dispositivo móvel. Caso esses dados não estejam presentes eles serão armazenados no banco de dados, só assim será possível visualizar o dado quando o dispositivo móvel não mais apresentar conectividade.

As classes *EqualsUtils* e *HashCodeUtils* têm como finalidade auxiliar na implementação dos métodos *equals* e *hashCode*, possibilitando ao desenvolvedor deverá apenas informar quais os atributos deverão ser considerados.

Na Listagem 3.3 será exemplificado a utilização dessas classes supracitadas.

```
1  @Override
2  public boolean equals(Object other){
3      return EqualsUtils.equalsUtils(this, other, "id", "nome");
4
5  }
6
7  @Override
8  public boolean hashCode(){
9      return HashCodeUtils.hashCodeUtils(this, "id", "nome");
10
11 }
```

---

### Listagem 3.3: Exemplo do Uso do EqualsUtils e hashCodeUtils

O método equals pode ser facilmente implementado fazendo uso da Classe *EqualsUtils*, sendo necessário informar qual a classe deverá ser considerada, qual a classe deverá ser verificada a igualdade e, por fim, quais atributos deverão ser considerados para que se possa comparar e informar se os objetos são similares.

O classe *HashCodeUtils* tem como finalidade auxiliar o desenvolvimento das aplicações utilizando o framework. Dessa forma é necessário que o desenvolvedor informe para o método *HasCodeUtils* a classe que se deseja realizar a verificação e quais os campos que devem ser considerados para que se possa realizar o teste de similaridade.

## 3.7 Métodos Existentes

Atualmente o framework contempla diversos métodos essenciais, cujo objetivo principal é reduzir a complexidade da construção das aplicações na plataforma Android para que as classes mapeadas possam trabalhar de forma satisfatória. Dessa forma, é possível realizar desde uma busca simples até uma mais complexa.

```
1 private void montarListaSelecao() throws Exception {  
2  
3     List<TipoAtividade> tipos = (List<TipoAtividade>) OffLineManager  
        .findAll(TipoAtividade.class);  
4  
5     if ( tipos != null ) {
```

### Listagem 3.4: Consulta Geral

Na Listagem 3.4 é demonstrado uma busca simples, onde é necessário informar apenas a classe, sendo retornado todos os registros encontrados daquelas classes no banco de dados ou no serviço. Já a Listagem 3.5 apresenta a realização de uma busca mais detalhada, em que é informado quais campos devem ser considerados para a realização da busca.

```
1 if ( getIntent().getExtras().getSerializable("evento") == null ) {  
2  
3     try {  
4         eventos = (Evento) OffLineManager.findAllExactyField(new  
            Evento(idEvento), "id");  
5     } catch (Exception e) {  
6         e.printStackTrace();
```



```

7         }
8
9     }

```

Listagem 3.5: Consulta Realizada pelo Atributo Informado

No método *findAll* o framework é informado acerca de qual classe se deseja realizar a busca. Se o dispositivo apresentar conectividade, o framework irá realizar a requisição servidor de aplicação sem a realização de nenhum filtro. Caso não seja encontrado, ou o dispositivo não apresente conectividade, este irá realizar uma busca com o intuito de retornar os registros da classe desejada no banco de dados do dispositivo e retorna-lo para o usuário.

```

1 try {
2
3     int id = Integer.parseInt(split[0]);
4     participacao = new Participacao();
5     participacao.setAtividade(atividade);
6     participacao.setParticipante(new Participante());
7     participacao.getParticipante.setId(id);
8     participacao = (Participacao) OffLineManager.findExactyField(
        participacao, "participante.id", "atividade.id");

```

Listagem 3.6: Consulta Realizada pelos Atributos Informados

Na Listagem 3.6 é exemplificado o método *findExactyField*, informando alguns parâmetros que deverão ser considerados para que se possa realizar a requisição. A informação fornecida quanto ao padrão utilizado de URL para a realização dos serviços é utilizada da seguinte forma: se o padrão adotado for o *PathParam*, a URL será criada considerando a configuração fornecida inicialmente ao framework, seguida do nome da classe que está realizando a requisição e, posteriormente, dos parâmetros desejados separados por uma /. Enquanto que, se o padrão utilizado tiver sido o *QueryParam*, a requisição se diferencia apenas na construção final da requisição, onde não mais será utilizado a / para separar os parâmetros da requisição. Esse padrão utilizará o nome dos atributos para compor a URL.

O framework ainda contempla um outro método, o *findExactyField*, que consiste no retorno de um objeto a partir do dado informado. A sintaxe do método é bem similar ao método *findAllExactyField*, como pode ser observado na Listagem 3.7. Porém, ao realizar a requisição junto ao servidor de aplicação e for retornado mais de um elemento, o framework realizará um tratamento para que sempre retorne o primeiro elemento da lista.

```

1 if ( getIntent().getExtras().getSerializable("evento") == null ) {
2
3     try {
4         evento = (Evento) OffLineManager.findExactyField(new
5             Evento(idEvento), "id");
6     } catch (Execption e) {
7         e.printStackTrace();
8     }
9 }

```

Listagem 3.7: Consulta Realizada retornando apenas um registro

Outro método existente no framework são as operações básicas de criação, atualização e remoção dos objetos. No caso da criação por padrão, será enviado para o servidor de aplicação um JSON com todas as informações do objeto em que se deseja cadastrar, assim como será enviado um JSON quando a operação oriunda do dispositivo móvel desejar atualizar um dado no servidor de aplicação. Entretanto, quando a operação se tratar de uma operação de delete o padrão utilizado pelo framework será enviar apenas o id do objeto para que possa ocorrer a remoção da base de dados. A Listagem 3.8 ilustra a operação de criação de um objeto.

```

1 try {
2
3     participacao.getRegistroParticipacao().setResponsavelDispositivo
4         ((ResponsavelDispositivo)
5             OffLineManager.findExactyField(new
6                 ResponsavelDispositivo(
7                     (Secure.getString(
8                         getContentResolver(), Secure.
9                         ANDROID_ID)), atividade.
10                        getEvento()),
11                        "serialDispositivo", "
12                        evento.id"));
13
14     try {
15
16         OffLineManager.insert(participacao);
17         Toast.makeText(getApplicationContext(), "
18             Participante tem permissão para entrar.",
19             Toast.LENGTH_LONG).show();
20     } catch (Exception e) {
21
22     }
23 }

```

```

12         Toast.makeText(getApplicationContext(), "Erro a
13             inserir", Toast.LENGTH_LONG).show();
14         e.printStackTrace();
    }

```

Listagem 3.8: Método do Insert

Em havendo conectividade, o framework apresentará a capacidade de realizar a verificação de existência de alguma informação armazenada no banco de dados do dispositivo que ainda não tenham sido sincronizada junto ao servidor de aplicação. A realização do envio dos dados ocorrerá de forma transparente, disponibilizando ao desenvolvedor a possibilidade de invocar um método que force o framework a realizar o envio das informações, como pode ser visto na Listagem 3.9, desde que o dispositivo apresente conectividade.

```

1 public static void sync() throws Exception {
2
3     List<Sincronizacao> bdLocal = pm.findAll(Sincronizacao.class);
4     if ( bdLocal != null && NetWorkUtils.isOnline() ) {
5         for( Sincronizacao sync : bdLocal ){
6
7             Object entity = Class.forName(sync.getClass()).
8                 getConstructor().newInstance();
9             FieldReflection.setValue(entity, entity.getClass()
10                (), FieldReflection.getField(sync.getClass(),
11                "idClasse"), sync.getIdClasse());
12             pm.find(entity);
13
14             //Jogando o valor do Id dos objetos
15             List<Field> fields = EntityReflection.
16                 getEntityFields(entity.getClass());
17             for ( Field field : fields ){
18                 if ( EntityReflection.isAnnotation(field
19                    , ManyToOne.class) ) {
20                     field.setAccessible(true);
21                     Object value = field.get(entity)
22                         ;
23                     FieldReflection.setValue(value,
24                         value.getClass(),
25                         FieldReflection.getField(
26                             value.getClass(), "id"), null
27                     );
28                 }
29             }
30         }
31     }
32 }

```

```

20
21      //Realizando a requisição
22      if ( sync.isCreate() ) {
23          FieldReflection.setValue(entity, entity.
24              getClass(), FieldReflection.getField(
25                  entity.getClass(), "id"), null);
26          entity = service.post(entity);
27      }
28
29      if ( sync.isUpdate() ) {
30          FieldReflection.setValue(entity, entity.
31              getClass(), FieldReflection.getField(
32                  entity.getClass(), "id"), null);
33          entity = service.put(entity);
34      }
35
36      for ( Field field : fields ){
37          if ( EntityReflection.isAnnotation(field
38              , ManyToOne.class) ) {
39              field.setAccessible(true);
40              Object value = field.get(entity)
41              ;
42              pm.updateIdClass(value, sync.
43                  getIdClasse(), (Integer)
44                  EntityReflection.getID(entity)
45                  ));
46          }
47      }
48
49      pm.updateIdClass(entity, sync.getIdClasse(), (
50          Integer) EntityReflection.getID(entity));
51      pm.remove(entity);
52  }
53
54  }
55
56  }

```

Listagem 3.9: Implementação do método que força a sincronização

Para dar uma maior liberdade para o desenvolvedor, foi criado o método *executeNativeSQL*, que possibilita a criação de instruções nativas do SQL, com a possibilidade de executar na base de dados do dispositivo móvel, como pode ser visto na Listagem

3.10, que consiste na alteração do evento que apresentar Id 40 para o Id do evento carregado anteriormente. Nesse caso a utilização do método está atrelado ao uso da Classe *EntityReflection* e *FieldReflection* já apresentadas na seção 3.7.

```

1 public void alterarId(Object entity, int id) {
2
3     OffLineManager.executarNativeSQL("UPDATE" + EntityReflection.
4         getTableName(entity.getClass()) +
5         " SET " + FieldReflection.getColumnNames(entity.
6             getClass(), "id") + " = " + id +
7         " WHERE " + FieldReflection.getColumnNames(entity.
8             getClass(), "id") + " = 40");
9 }

```

Listagem 3.10: Possibilidade de executar uma instrução *SQL*

## 3.8 Conclusões

A proposta de criação do Framework apresentada neste capítulo foi projetada para dar suporte as aplicações móveis, para que estas sejam capazes de executar operações sem a necessidade de conexão com a internet, fazendo apenas uso da base de dados local, o que permite que o aplicativo tenha mais cobertura, abrangendo tanto os usuários com acesso contínuo à rede como aqueles que não têm acesso em todos os momentos.

Na fase de levantamento de requisitos, que ocorreu antes do início da implementação, foram levantados diversos requisitos funcionais e não funcionais que deveriam ser atendidos e também que o framework sofreria um forte influência do JPA, não contemplando a preocupação com a resolução de conflitos.

O *OffDroid* tem como principais funções a persistência, a sincronização, o consumo de dados utilizando a tecnologia REST e principalmente a possibilidade de definir a forma como cada classe deverá se comportar durante o uso do aplicativo desenvolvido fazendo uso do framework. Atrelado a essas características, o framework é capaz de realizar a criação das tabelas no banco de dados do dispositivo, assim como os relacionamentos entre as tabelas, além de fornecer as operações essenciais para o desenvolvimento das mais diversas aplicações para a plataforma Android.

## 4 Experimento para Avaliação do Framework

Com o intuito de avaliar a produtividade e a eficácia do uso do framework *OffDroid*, um estudo experimental controlado foi conduzido utilizando o Processo de Experimentação proposto por (WOHLIN et al., 2000). O estudo precisou de um planejamento minucioso, composto por quatro participantes e sem a possibilidade de novas réplicas caso ocorresse alguma ameaça grave à sua validade.

Ao longo do experimento foi observado o tempo de desenvolvimento da solução com uso ou com o não do framework, resultando nas três métricas quantitativas realizando comparações tempo de desenvolvimento, quantidade de linhas de código desenvolvido e tempo médio gasto em cada uma das operações realizadas.

### 4.1 Definição do Experimento

Nessa seção será definidos os objetivos, as hipóteses do experimento, a seleção das variáveis de resposta, bem como a realização da seleção dos participantes.

#### 4.1.1 Objetivos do Experimento

O objetivo do experimento é definido seguindo o paradigma de melhoria contínua GQM, descrito a seguir:

- **Objetivo do Estudo:**
  - Utilização do framework **OffDroid**
- **Próposito:**
  - Avaliar a utilização do framework **OffDroid**;

- **Enfoque:**

- Tempo de desenvolvimento;
- Quantidade de linha de código;
- Entendimento do programa;

- **Contexo:**

- O experimento ocorreu dentro do laboratório da SINFO;

O objetivo do experimento consiste na construção do código para a resolução dos problemas com e sem a utilização do framework, para que se possa extrair as métricas para responder as questões levantadas na seção 4.1.2.

### 4.1.2 Questões e Métricas

Nessa seção serão levantadas algumas questões que devem ser verificadas (Q) e as suas respectivas métricas de respostas (M).

- **Q1:** Para solucionar o problema proposto, o tempo de desenvolvimento reduz utilizando o framework?
  - M1.1 Quantidade de minutos necessários para a solucionar o problema proposto.
- **Q2:** O tempo de comunicação com o servidor aumenta significativamente utilizando o framework pra solucionar problema proposto?
  - M2.1 Quantidade de milisegundos necessários para que o ocorra a comunicação entre o framework e o servidor.
- **Q3:** Há uma redução na quantidade de linhas de código quando utilizado o framework pra solucionar problema proposto?
  - M3.1 Medição da quantidade de linhas de código necessária pra solucionar o problema proposto, desconsiderando as classes de domínio.

## 4.2 Planejamento

Nas seções que seguem será exemplificado o planejamento e a transcrição do processo de experimentação, conforme proposto por (WOHLIN et al., 2000).

### 4.2.1 Formulação das Hipóteses

O framework foi analisado tanto em relação ao ganho da produtividade no desenvolvimento da solução quanto pela eficácia da solução proposta. As hipóteses a serem contrastadas referem-se, portanto, ao ganho da produtividade e a eficácia quanto o tempo de resposta entre o dispositivo e o servidor para as abordagens.

As hipóteses quanto a produtividade:

- **Hipótese Nula (H0):** Não há diferença no tempo de desenvolvimento do problema proposto utilizando o *OffDroid*.
- **Hipótese Alternativa (H1):** Há diferença no tempo de desenvolvimento do problema proposto utilizando o *OffDroid*.

As hipóteses relativas à eficácia na comunicação entre o framework e o servidor:

- **Hipótese Nula (H0):** Não há diferença troca de informação entre o servidor e o dispositivo utilizando o *OffDroid*.
- **Hipótese Alternativa (H1):** Há diferença troca de informação entre o servidor e o dispositivo utilizando o *OffDroid*.

As hipóteses relativas a quantidade de linhas de código necessárias para solucionar o problema proposto:

- **Hipótese Nula (H0):** Não há diferença na quantidade de linhas de código necessária para solucionar o problema.
- **Hipótese Alternativa (H1):** Há diferença na quantidade de linhas de código necessária para solucionar o problema.

### 4.2.2 Seleção das Variáveis

As variáveis consideradas no experimento foram definidas de acordo com os requisitos para a utilização do framework *OffDroid* e com as métricas definidas para avaliar o seu uso. Foram classificadas da seguinte maneira:

- **Variáveis independentes**



- A complexidade dos projetos e de casos do uso previamente inseridos no experimento;
- A seleção dos indivíduos selecionados, por apresentarem o mesmo conhecimento teórico sobre o framework.

- **Variáveis Dependentes**

- O tempo necessário para entregar a solução para o problema proposto;
- A eficiência da solução proposta para a solucionar o problema atrelado.

### 4.2.3 Seleção dos Participantes

A seleção dos participantes ocorreu mediante a seleção determinística, envolvendo os servidores, funcionários terceirizados e estagiários da SINFO. Os participantes deveriam ter o conhecimento teórico de testes unitários e o conhecimento prático na plataforma Android para participar das atividades do experimento.

Foram selecionados quatro funcionários da SINFO que apresentavam um alto grau de conhecimento sobre a plataforma Android, por isso não foi necessário ministrar um treinamento para nivelar os participantes na plataforma Android. Entretanto, como a grande maioria dos selecionados não apresentava conhecimento sobre o framework, foi realizado um treinamento com o intuito de prover condições mínimas para a realização do experimento. O treinamento consistiu na simulação da realização da execução do experimento, tendo ocorrido durante dois dias antes da execução do experimento, com duração de três horas e trezes minutos.

## 4.3 Design do Experimento

Definir o design do experimento controlado é fundamental para garantir a generalização das análises realizadas sobre os tratamentos avaliados, neste caso fazendo uso e desuso do framework para a resolução do problema proposto.

As três métricas utilizadas para avaliação foram: produtividade; a eficácia na comunicação entre o framework e o servidor; a quantidade de linha de código relativas para solucionar o problema. Para controlar os fatores externos foi utilizado o design totalmente aleatorizado.

Segundo (WOHLIN et al., 2000), o experimento conduzido foi o Tipo I que consiste em apresentar um fator que é a resolução do problema proposto e dois tratamentos que consistem no uso e não do framework.

Como ilustrado na Tabela 5, as linhas representam os programadores e as colunas os problemas aplicados nas réplicas geradas.

Tabela 5: Definição da Sequência dos Problemas

Desenv	Problema	Problema	Problema	Problema
Desenv 1	Problema 1	Problema 2	Problema 3	Problema 4
Desenv 2	Problema 1	Problema 2	Problema 3	Problema 4
Desenv 3	Problema 5	Problema 6	Problema 7	Problema 8
Desenv 4	Problema 5	Problema 6	Problema 7	Problema 8

## 4.4 Preparação do Experimento

Durante a preparação do experimento foi constatado que este deveria ser executado em quatro etapas, que consiste nos quatro problemas propostos, e com dois tratamentos distintos, com e sem o uso do framework, para todos os participantes. Cada etapa foi realizada em dias distintos, sem intervalo. A Tabela 6 apresenta a ordem em que os tratamentos foram aplicados para cada um dos participantes.

Tabela 6: Preparação do Experimento

Participante	Problema 1	Problema 2	Problema 3	Problema 4
Participante 1	Manual	Manual	Framework	Framework
Participante 2	Framework	Manual	Manual	Framework
Participante 3	Framework	Framework	Manual	Manual
Participante 4	Manual	Manual	Framework	Framework

## 4.5 Processo de Experimentação

O fator deste experimento consiste na análise do código fonte aplicado aos tratamentos distintos, com e sem o uso do framework *OffDroid*. De acordo com as hipóteses formuladas, seção 4.2.1, o framework *OffDroid* pode ou não apresentar uma maior produtividade para a resolução do problema; aumentar ou não significativamente a troca de mensagens entre o dispositivo e o servidor de aplicação; reduzir ou não significativamente a quantidade de linhas de código presentes na solução do problema.

Durante o processo de experimentação, para verificar a primeira hipótese nula formulada, referente ao ganho de produtividade com e sem o uso do framework *OffDroid*, foi utilizada a medição, em minutos, do tempo de desenvolvimento que cada participante necessitou para fornecer a solução do problema proposto. A medição inicializou em cronômetros distintos para cada um dos participantes para que não houvesse nenhum ruído externo. O cronômetro só era pausado quando a solução proposta pelo participantes já havia passado em todos os testes unitários.

Para verificar a segunda hipótese nula formulada, referente a eficácia na comunicação entre o framework e o servidor, foi adicionado, em cada operação, um temporizador com o intuito de medir o tempo gasto para a realização da operação. A coleta dessa medição foi repetida cinco vezes para cada operação, retirando delas a média entre essas execuções para que pudesse remover possíveis ruídos ou interferências. Deve ser ressaltado que a verificação da segunda hipótese só foi inicializada após o término do desenvolvimento da solução e de posse de todos os códigos fontes.

A terceira hipótese nula formulada, referente a quantidade de linhas de código necessárias no desenvolvimento da solução do problema proposto, só pôde ser iniciada com a verificação da segunda hipótese, depois que finalizado o processo de desenvolvimento do código fonte da solução. A medição consistiu na contabilização das linhas de código de cada uma das soluções entregues, sendo retirada a quantidade de linhas de código das classes de domínio.

#### 4.5.1 Avaliação de Validade

Na *Validade de Conclusão* todos os indivíduos, antes de entregar a solução proposta, tiveram que aplicar uma série de testes, cujo objetivo era verificar se a solução atendia aos requisitos desejados.

No que tange a *Validade de Construção*, conforme discussão apresentada na seção 4.2.3, foi necessário que o participante tivesse conhecimento prévio sobre a plataforma Android. Como nem todos os participantes selecionados apresentavam conhecimento sobre o framework, foi realizado um treinamento com o intuito de prover condições mínimas para a realização do experimento. O problema abordado no treinamento apresentava a mesma abordagem dos problemas que, posteriormente foram adotados no experimento, para que dessa forma os participantes apresentassem as mesmas condições. Todos os indivíduos utilizaram o mesmo ambiente de desenvolvimento (Eclipse) e máquinas com configurações equivalentes.

Com respeito a *Validade Externa*, como todos os participantes do processo de experimentação são servidores, funcionários terceirizados ou bolsistas da SINFO, que trabalham direta ou indiretamente no desenvolvimento de aplicativos móveis da plataforma Android, eles apresentaram as condições mínimas para participarem do experimento. Dessa forma, é possível generalizar os resultados obtidos entre os demais profissionais que atuam no mercado.

#### 4.5.2 Análise dos Resultados

Para a análise dos resultados do experimento houve a preocupação com a precisão dos dados e o problema da interpretação estatística de amostras pequenas. Por esse motivo foi utilizado o método não-paramétrico (Unilateral Mann-Whitney U), considerado a alternativa mais poderosa para o teste paramétrico para amostras independentes (LEVIN; FOX, 2006).

A Tabela 7 apresenta os dados coletados durante a execução do experimento. A primeira hipótese foi verificada mediante a resposta da pergunta e da métrica a ela atrelada.

Tabela 7: Resultados do Experimento

<b>Participante</b>	<b>Problema</b>	<b>Ferramenta</b>	<b>Tempo (min)</b>
Participante 1	Problema 1	Manual	98:45
Participante 2	Problema 1	Framework	36:07
Participante 1	Problema 2	Manual	94:25
Participante 2	Problema 2	Manual	186:01
Participante 1	Problema 3	Framework	30:24
Participante 2	Problema 3	Manual	67:34
Participante 1	Problema 4	Framework	18:31
Participante 2	Problema 4	Framework	14:50
Participante 3	Problema 5	Framework	41:33
Participante 4	Problema 5	Manual	194:28
Participante 3	Problema 6	Framework	40:51
Participante 4	Problema 6	Manual	165:07
Participante 3	Problema 7	Manual	144:13
Participante 4	Problema 7	Framework	44:02
Participante 3	Problema 8	Manual	76:26
Participante 4	Problema 8	Framework	21:31

Enquanto que as Tabelas 8 e 9 apresentam respectivamente a média de tempo necessário para solucionar os problemas para cada um dos tratamentos.

Tabela 8: Projeto 1 Mann-Whitney

	<b>Projeto 1</b>
Manual	1:44:11
Framework	21:15

Tabela 9: Projeto 2 Mann-Whitney

	<b>Projeto 2</b>
Manual	2:10:18
Framework	36:59

Ao realizar o teste *Mann-Whitney* foi constatado que o valor de p ficou em 0,029. Dessa forma é possível afirmar que o tempo de desenvolvimento utilizando o framework é inferior ao tempo de desenvolvimento tradicional. Em outras palavras, a hipótese nula  $H_0$  não está descartada, conforme também pode ser visto na Figura 14.

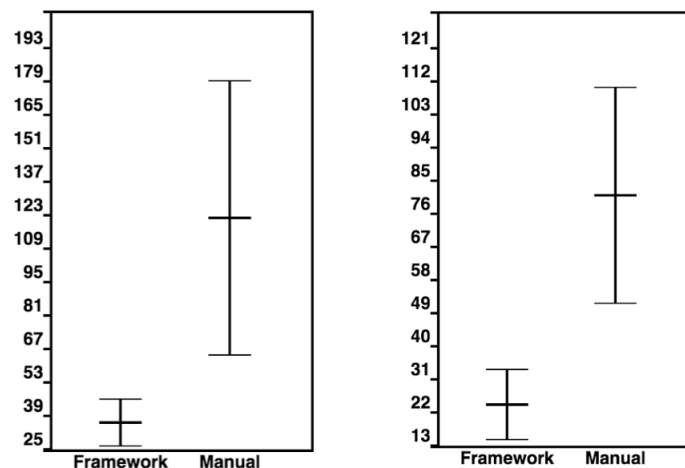


Figura 14: Box-Plot Tempo de Desenvolvimento

Já com o intuito de verificar a segunda hipótese, que consiste em responder a Questão 2, descrita no seção 4.1.2 e a sua respectiva métrica M2.1., foi coletado os dados apresentados na Tabela 10.

Tabela 10: Performance das Requisições

Participante	Problema	Tipo	Inserir	Ler	Atualizar	Remover
Participante 1	Problema 1	Manual	132ms	155ms	70ms	25ms
Participante 2	Problema 1	Framework	69ms	88ms	66ms	40ms
Participante 1	Problema 2	Manual	112ms	219ms	46ms	16ms
Participante 2	Problema 2	Manual	35ms	24ms	26ms	29ms
Participante 1	Problema 3	Framework	71ms	84ms	63ms	38ms
Participante 2	Problema 3	Manual	33ms	25ms	31ms	25ms
Participante 1	Problema 4	Framework	72ms	91ms	65ms	36ms
Participante 2	Problema 4	Framework	69ms	88ms	68ms	40ms
Participante 3	Problema 5	Framework	140ms	130ms	121ms	59ms
Participante 4	Problema 5	Manual	34ms	30ms	31ms	30ms
Participante 3	Problema 6	Framework	139ms	131ms	118ms	49ms
Participante 4	Problema 6	Manual	31ms	65ms	39ms	41ms
Participante 3	Problema 7	Manual	134ms	46ms	44ms	27ms
Participante 4	Problema 7	Framework	142ms	145ms	119ms	52ms
Participante 3	Problema 8	Manual	140ms	370ms	38ms	25ms
Participante 4	Problema 8	Framework	150ms	150ms	120ms	50ms

Como o valor de  $p$  varia entre 0,029 e 0,114, para a atualização e exclusão respectivamente, por ser este inferior a 5% é possível afirmar que não há uma diferença significativa no tempo de comunicação entre o framework e o servidor, quando comparado a abordagem tradicional e a utilizando o framework. Pois entre as operações básicas não foi encontrado nenhuma evidência de que o framework retarda a comunicação ou persistência dos dados. Portanto, a hipótese nula  $H_0$  está descartada para algumas operações.

A terceira hipótese, que tem objetivo de verificar se a quantidade de código desenvolvido apresentou uma diminuição significativa entre os tratamentos, só foi possível devido a contabilização das linhas de código, conforme apresentado na Table 7.

Tabela 11: Linhas de código desenvolvido para solucionar o Problema

Participante	Problema	Tipo	Total de linhas de código
Participante 1	Problema 1	Manual	394
Participante 2	Problema 1	Framework	129
Participante 1	Problema 2	Manual	383
Participante 2	Problema 2	Manual	604
Participante 1	Problema 3	Framework	150
Participante 2	Problema 3	Manual	468
Participante 1	Problema 4	Framework	145
Participante 2	Problema 4	Framework	124
Participante 3	Problema 5	Framework	312
Participante 4	Problema 5	Manual	724
Participante 3	Problema 6	Framework	342
Participante 4	Problema 6	Manual	721
Participante 3	Problema 7	Manual	638
Participante 4	Problema 7	Framework	80
Participante 3	Problema 8	Manual	581
Participante 4	Problema 8	Framework	97

Como o valor de  $p$  calculado foi 0,029 e este valor é inferior a 5%, não se pode rejeitar a hipótese nula. Ou seja, é possível afirmar que há uma diferença significativa na quantidade de linhas de código desenvolvido, quando comparado a abordagem tradicional e a abordagem utilizando o framework.

## 4.6 Considerações Finais

Como verificado na seção 4.5.2, em que foi apresentado os resultados obtidos do experimento, é possível constatar que duas três hipóteses nulas foram descartada, com base na estatística. Concluindo, dessa forma, que o framework *OffDroid* provê uma maior produtividade para a resolução de problemas, sem onerar significativamente a comunicação entre dispositivo móvel e o servidor, reduzindo significativamente a quantidade de linhas código necessária para o desenvolvimento da solução.

Durante as fases de planejamento e de preparação foi detectado todas as possíveis

influências que poderiam interferir na execução e, conseqüentemente, na avaliação das hipóteses. Por esse motivo foi realizado um prévio treinamento, antes da execução do experimento com o intuito de nivelar os participantes quanto ao uso do framework, bem como a definição da IDE, para que não houvesse influência sobre o resultado da avaliação das hipóteses.



## 5 Prova de Conceito: Coletor Presença Mobile

Essa seção tem como finalidade exemplificar a utilização do framework construído em um caso concreto. O framework foi utilizado nos aplicativos recém desenvolvidos, são eles: o Coletor de Presença Mobile, UFRN Notify, UFRN security e SigEventos.

### 5.1 Introdução

Dentre os aplicativos supramencionados, o Coletor de Presença Mobile foi o selecionado, apresentando como principal função realizar a coleta de presenças dos participantes de um evento e das atividades que o compõe. O aplicativo tem como função exportar as presenças coletadas para um servidor de aplicação, cuja finalidade é a emissão dos certificados dos participantes presentes. Porém, o aplicativo tem um requisito funcional, que é permitir que a coleta possa ser realizadas mesmo que o dispositivo não apresente conectividade. Essa presença deverá ficar armazenada no dispositivo e, quando o mesmo retornar a conectividade, as presenças coletadas serão sincronizadas para o servidor de aplicação.

Para utilizar o framework é necessário, inicialmente, importar o jar do framework. O aplicativo deverá apresentar duas permissões, uma de acesso a internet e a outra que possa ter acesso ao estado da conexão. Essas duas permissões são necessárias para que o framework possa funcionar, uma vez que o mesmo precisa verificar o estado da rede para definir a estratégia a ser utilizada a cada solicitação oriunda do aplicativo.

### 5.2 Inicialização

Para que o framework possa ser inicializado é necessário realizar a configuração presente na Listagem 5.1.

```

1  try {
2
3      List<String> classes = new ArrayList<String>();
4      classes.add("br.ufrn.coletor.model.Evento");
5      classes.add("br.ufrn.coletor.model.TipoAtividade");
6      classes.add("br.ufrn.coletor.model.TipoParticipacao");
7      classes.add("br.ufrn.coletor.model.ResponsavelDispositivo");
8      classes.add("br.ufrn.coletor.model.Participante");
9      classes.add("br.ufrn.coletor.model.Atividade");
10     classes.add("br.ufrn.coletor.model.RegistroParticipacao");
11     classes.add("br.ufrn.coletor.model.Participacao");
12
13     OffLineManager.createOffLineManager(this, "coletor.db", classes,
14                                         new WebServiceImpl("http://127.0.0.1:8080/
15                                                                ColetorPresencaWEB/rest/mobile/", true));
16 } catch (Exception e) {
17     e.printStackTrace();
18 }

```

Listagem 5.1: Inicialização do Framework

Para a inicialização do framework se faz necessário informar qual o nome do banco que deverá ser criado no dispositivo móvel, para só então serem armazenadas as informações do aplicativo. Também deverá ser informado uma lista das classes que o framework deverá realizar o gerenciamento da persistência, as operações a elas inerentes, e, por fim, deverá ser informado acerca de uma instância da classe `WebService`, contendo a URL do serviço que o aplicativo realizará as requisições e qual o padrão será utilizado `QueryParam` ou `PathParam`.

Uma vez inicializado o framework, não é necessário realizar nenhuma outra configuração adicional para que se possa utilizá-lo, pois o framework irá manter a configuração fornecida durante toda a sua utilização, já sendo possível utilizá-lo com todos os seus métodos.

## 5.3 Utilização

Nessa seção será exemplificado a utilização do framework através de trechos de código retirados do Coletor de Presença Mobile, para que se possa entender como os métodos

existentes no framework podem ser utilizados na construção dos aplicativos.

Será suprimido os mapeamentos realizados nas classes de domínio e a inicialização no aplicativo, conforme visto na seção 3.4.

Essa seção tem como foco a utilização do framework no caso prático e as suas implicações. No Coletor de Presença, os dispositivos que realizaram a coleta das presenças nos eventos precisam ser registrados e liberados. Esse registro e liberação ocorre vinculando o serial do dispositivo ao evento que o mesmo terá a responsabilidade de coletar as presenças. Dessa forma, a primeira tela que o aplicativo exibirá para o usuário será os eventos que o dispositivo apresente permissão de coleta. A Listagem 5.2 exemplifica o método utilizado para carregar todos os eventos que o dispositivo apresente a permissão de coletar as presenças.

```

1 try {
2     Evento evento = new Evento();
3     evento.setSerialDispositivo(Secure.getString(getContentResolver
4         (), Secure.ANDROID_ID));
5     eventos = (List<PersistDB>) OffLineManager.findAllExactyField(
6         evento, "serialDispositivo");
7 } catch(Exception e) {
8     e.printStackTrace();
9 }

```

Listagem 5.2: Carrega os eventos permitidos para o dispositivo

Para a realização da consulta é criado uma nova instância de um evento, sendo em seguida setado no atributo serialDispostivo o valor do serial do dispositivo que está fazendo uso do aplicativo, para que dessa forma só possa retornar os eventos no qual o dispositivo tenha permissão para acessar e realizar a coleta das presenças. Por fim é realizado a consulta, onde é passado os eventos anteriormente instanciados, informado qual o atributo deverá ser utilizado na realização da requisição junto ao servidor ou na consulta realizada na base de dados. Uma vez realizada a consulta, o framework retornará apenas aqueles eventos cuja o dispositivo apresente a permissão de registrar as presenças.

Uma vez carregado os eventos, também será necessário as atividades que compõem o eventos, para que se possa posteriormente coletar as presenças em cada uma das atividades existentes.

```

1 try {
2     Atividade atividade = new Atividade();
3     atividade.setEvento(evento);

```

```

4         atividade.setTipoAtividade(new TipoAtividade());
5         atividade.getTipoAtividade().setId(2);
6         atividades = (List<PersistDB>) OffLineManager.findAllExactyField
            (atividade, "evento.id", "tipoAtividade.id");
7     } catch(Exception e) {
8         e.printStackTrace();
9     }

```

Listagem 5.3: Carrega todas as atividades do evento

A Listagem 5.3 mostra uma forma de carregar todas as atividades existentes no evento pelo tipo, em que a primeira ação realizada consiste na instanciação de uma nova atividade para que seja utilizada na consulta que retornará do servidor ou da base de dado do dispositivo as atividades daquele tipo.

Na linha logo abaixo é setado na variável o evento anteriormente selecionado, sendo informado ao evento qual o tipo de atividade que se deseja. Posteriormente é realizado a busca, onde é passado para o framework o objeto e quais os atributos que o mesmo deverá levado em consideração, realizando a requisição no servidor de aplicação ou, se o dispositivo não apresentar conectividade, quais atributos deverão ser considerados na busca na base de dados do dispositivo.

```

1 try {
2     int id = Integer.parseInt(slit[0]);
3     participacao = new Participacao();
4     participacao.setAtividade(atividade);
5     participacao.setParticipante(new Participante());
6     participacao.getParticipante().setId(id);
7     participacao = (Participacao) OffLineManager.findExactyField(
8         participacao, "participante.id", "atividade.id");
9     participacao.getRegistroParticipacao().getResponsavelDispositivo
10        (
11            (DispositivoResponsavel) OffLineManager.
12                findExactField(new ResponsavelDispositivo(
13                    (Secure.getString(
14                        getContentResolver(), Secure.
15                            ANDROID_ID)), atividade.
16                            getEvento()), "
17                                serialDispositivo", "evento.
18                                    id"));
19
20     try {
21         OffLineManager.insert(participacao);

```

```
14         Toast.makeText(getApplicationContext(), "Participante  
        tem permissão para entrar.", Toast.LENGTH_LONG).show  
        ();  
15     } catch(Exception e) {  
16         Toast.makeText(getApplicationContext(), "Erro ao  
        cadastrar a participação", Toast.LENGTH_LONG).show();  
17         e.printStackTrace();  
18     }
```

Listagem 5.4: Salvando a presença do participante

Na Listagem 5.4, inicialmente é realizada uma busca com a finalidade de retornar a participação do participante na atividade do evento selecionado. Em seguida será criado um novo registro de participação, com a hora e a data que houve a coleta da informação. Posteriormente será realizada uma outra consulta, cuja finalidade consiste em retornar o responsável pela coleta da presença do participante, criando dessa forma um registro de participação do mesmo e realizando inserção desse registro na base de dados do dispositivo. Apresentando conectividade, será realizado uma requisição ao servidor da aplicação para que o registro possa ser persistido e armazenado no banco de dados do dispositivo, desde que a classe não apresente a anotação *@OnlyOnLine*.

## 5.4 Comparativo entre as versões

Existe uma primeira versão do Coletor de Presença que está em uso desde 2012, sendo a sua implementação realizada sem o uso do framework desenvolvido. Por esse motivo, o desenvolvedor precisou se ater a todas as situações de conectividade existentes, ou seja, foi necessário verificar, a cada requisição, o estado da conexão apresentada pelo dispositivo para selecionar a estratégia a ser realizada, isto é, ou seria feito uma requisição junto ao servidor de aplicação, ou na base de dados do dispositivo. Além dessas verificações, a primeira versão do coletor de Presença Mobile tinha ainda dependência de outros dois projetos para o seu funcionamento.

A Figura 15 mostra a estrutura final da primeira versão do coletor de presença Mobile.

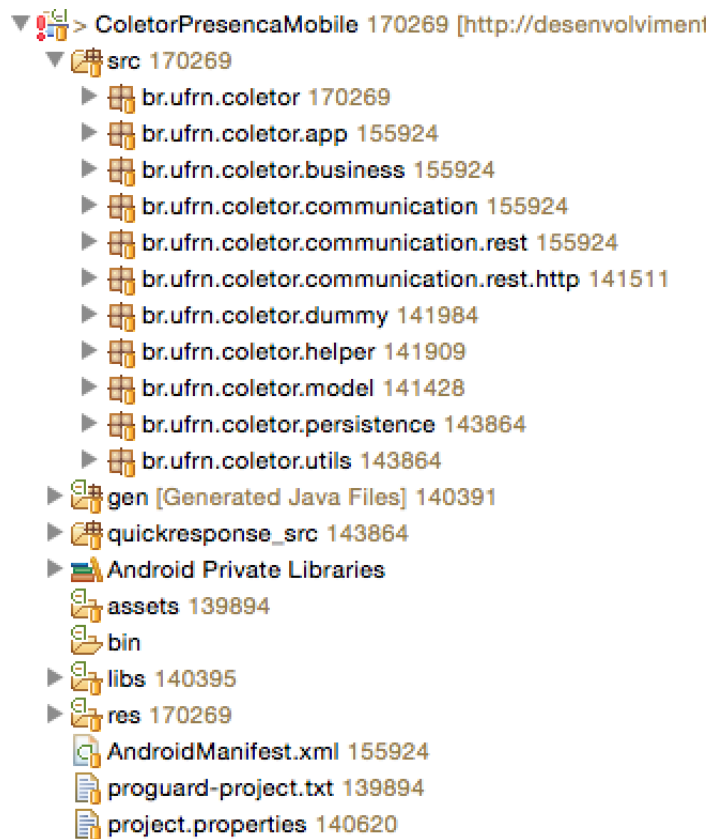


Figura 15: Estrutura da primeira versão do Coletor de Presença Mobile

A primeira versão do coletor era subdividido em 10 pacotes, em que cada pacote era responsável por agrupar determinadas funcionalidade existentes no aplicativo. O projeto contabilizava um total de 30 classes java e ainda 3427 linhas de código, para essa totalização não foi incluído as classes que fazem parte das Arquiteturas Mobile e da Arquitetura Android.

Para validar o framework foi desenvolvido uma nova versão do coletor, que deveria apresentar os mesmos requisitos funcionais, dentre eles, o principal, que consiste na possibilidade de coletar as presenças dos participantes mesmo o dispositivo não apresentando conectividade. A nova versão difere da versão antiga por já realizar a sincronização dos dados automaticamente ao detectar conexão e na existência de dados ainda não sincronizados. A estrutura final da segunda versão do aplicativo pode ser visualizado na Figura 16.

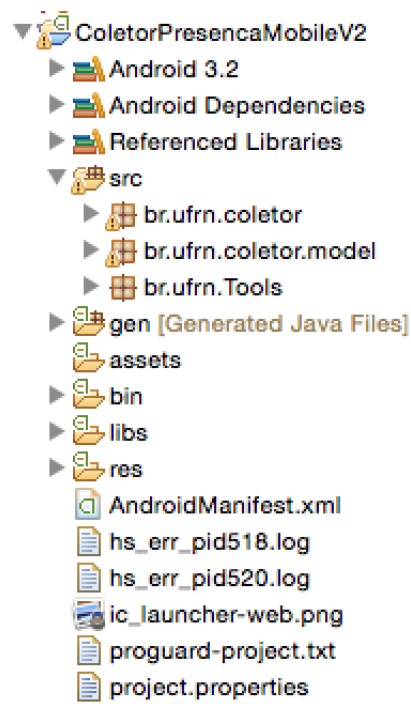


Figura 16: Estrutura da nova versão do Coletor de Presença Mobile

O novo projeto do aplicativo apresenta 3 pacotes, que corresponde na divisão das responsabilidades do aplicativo, e ainda apresenta 4 classes java e apenas 767 linhas de código, que realizam o mesmo trabalho das 30 classes e 3427 linhas de código que a primeira versão apresentava.

A Tabela 12 exibe um comparativo entre as versões do coletor de presença, em que a redução foi um pouco superior a 87% na quantidade de classes e de quase 78% na quantidade de linhas de código, mesmo apresentando toda redução na quantidade de código java. O projeto apresentou as mesmas funcionalidades.

Tabela 12: Comparativo entre as versões do Coletor de Presença Mobile.

Versão do Aplicativo	Quantidade de Pacotes	Quantidade de Classes	Quantidade de Linhas de Código
Versão 1.0	10	30	3427
Versão 2.0	2	4	767

## 5.5 Conclusão

.....



## 6 Trabalhos Relacionados

Neste capítulo serão apresentados os trabalhos alguns diretamente relacionados a esta dissertação, sendo destacado as principais contribuições e mostrando as principais diferenças entre as abordagens.

O trabalho apresentado por (SEDIVY et al., 2012) descreve um modelo denominado de MCSync, tendo como finalidade realizar a sincronização entre um dispositivo Android e um servidor de aplicação Web que faz uso dos serviços da Google, fornecendo a criação automática do banco de dados e das tabelas necessárias para a utilização do aplicativo. O MCSync opera no sistema operacional Android e oferece aos desenvolvedores a capacidade de construir bancos de dados SQLite, que podem ser sincronizado através do Google App Engine. O MCSync sincroniza os dados presentes no SQLite com os dados correspondentes no banco de dados no GAE, o protocolo Cloud Messaging é o responsável por notificar as mudanças ocorridas, bem como realizar a atualização da base de dados interna do dispositivo. A figura 17 mostra a arquitetura do MCSync.

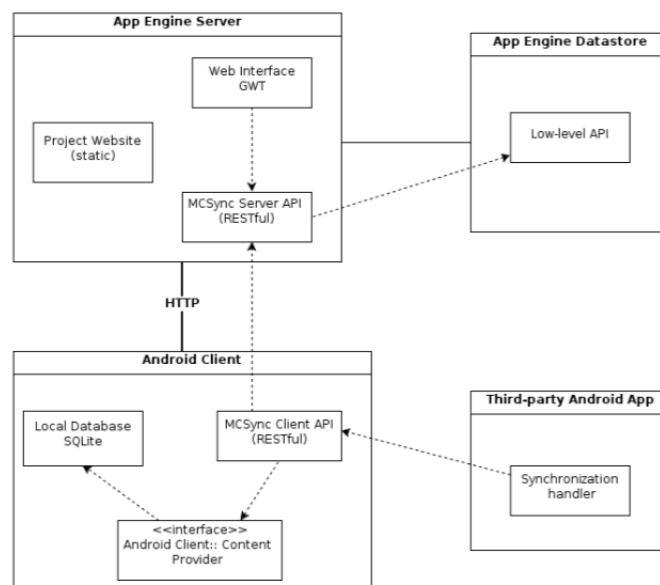


Figura 17: Arquitetura do MCSync (SEDIVY et al., 2012).

A principal diferença entre o framework e o trabalho apresentado por (SEDIVY et al., 2012) é que, enquanto o principal objetivo de SEDIVY é a realização da sincronização entre os dados do dispositivo móvel com o servidor utilizando exclusivamente a GAE, o Offdroid poderá realizar a sincronização com qualquer servidor de aplicação. Além disso, a abordagem de SEDIVY requer que o dispositivo móvel apresente conectividade permanente, enquanto que o Offdroid não necessita que os dispositivos móveis estejam conectados em tempo integral.

Outros dois pontos distintos entre as abordagens consiste no fato da replicação de SEDIVY ser total, uma vez que todo o banco de dados do GAE vai para o dispositivo móvel, ao contrário da replicação realizada pelo Offdroid que é parcial, em que nem todos os dados presentes no base de dados do servidor de aplicação vai para o dispositivo móvel. No Offdroid a atualização dos dados ocorre de forma gradativa, ocorrendo quando o dispositivo realizar alguma requisição junto ao servidor de aplicação ou se este disparar alguma mensagem diretamente para o dispositivo, ao contrário do que ocorre com a atualização dos dados de SEDIVY, que é feita na sua totalidade, em que sempre é requisitado todos os dados para verificar a existência ou não de alterações.

A semelhança entre a abordagem utilizada por SEDIVY e do framework Offdroid consiste no protocolo utilizado para a realização das trocas de mensagens entre o dispositivo móvel e o servidor de aplicação, que realizará as trocas utilizando o formato JSON.

(STAGE, 2005) apresenta o SyncML, que é um modelo de comunicação cliente-servidor responsável por realizar as trocas de mensagens utilizando o formato XML, sendo capaz de resolver conflitos durante o processo de sincronização. O modelo contempla ainda a sincronização bilateral ou unilateral. A Figura 18 mostra a arquitetura do SyncML.

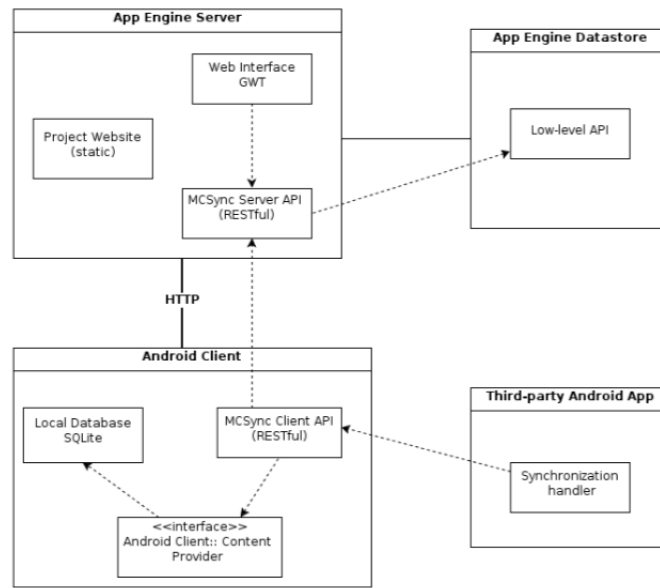


Figura 18: Arquitetura do SyncML (STAGE, 2005).

É possível constatar que a diferença encontrada entre o SyncML e o framework Offdroid está no formato das mensagens trocadas entre o servidor e o dispositivo móvel, pois no modelo SyncML é utilizado o formato XML, enquanto que no Offdroid as mensagens enviadas utilizam o formato JSON. Outra diferença está relacionada com a sincronização dos dados no SyncML, que apresenta técnicas de sincronização que podem ser utilizadas para atender os requisitos da aplicação. Já o Offdroid não apresenta a possibilidade de seleção das técnicas possíveis de seleção, pois ele deixa a cargo do servidor de aplicação a seleção da técnica a ser utilizada, uma vez que no dispositivo móvel o dado oriundo do servidor sempre será o que irá prevalecer.

A principal semelhança entre o SyncML e do framework Offdroid consiste na flexibilidade para a realização das sincronizações ou replicações entre o dispositivo móvel e o servidor de aplicação, onde a sincronização utilizada pode ser a bilateral ou a unilateral.

(CHOI et al., 2010) apresenta o algoritmo SAMD que trabalha apenas com instruções nativas SQL verificando a necessidade de realizar a atualização de uma tabela presente no dispositivo móvel. Para que o SAMD possa operacionalizar é necessário que a tabela esteja presente tanto no servidor de aplicação quanto no dispositivo móvel e que exista outras duas tabelas, uma para indicar ao dispositivo móvel caso algum dado seja atualizado e a outra para ser o elo de ligação entre os dados do banco de dados do servidor de aplicação e a tabela, que servirá como referência para sinalizar ao dispositivo móvel que apresenta alteração na base de dados. A Figura 19 exemplifica o funcionamento do algoritmo SAMD.

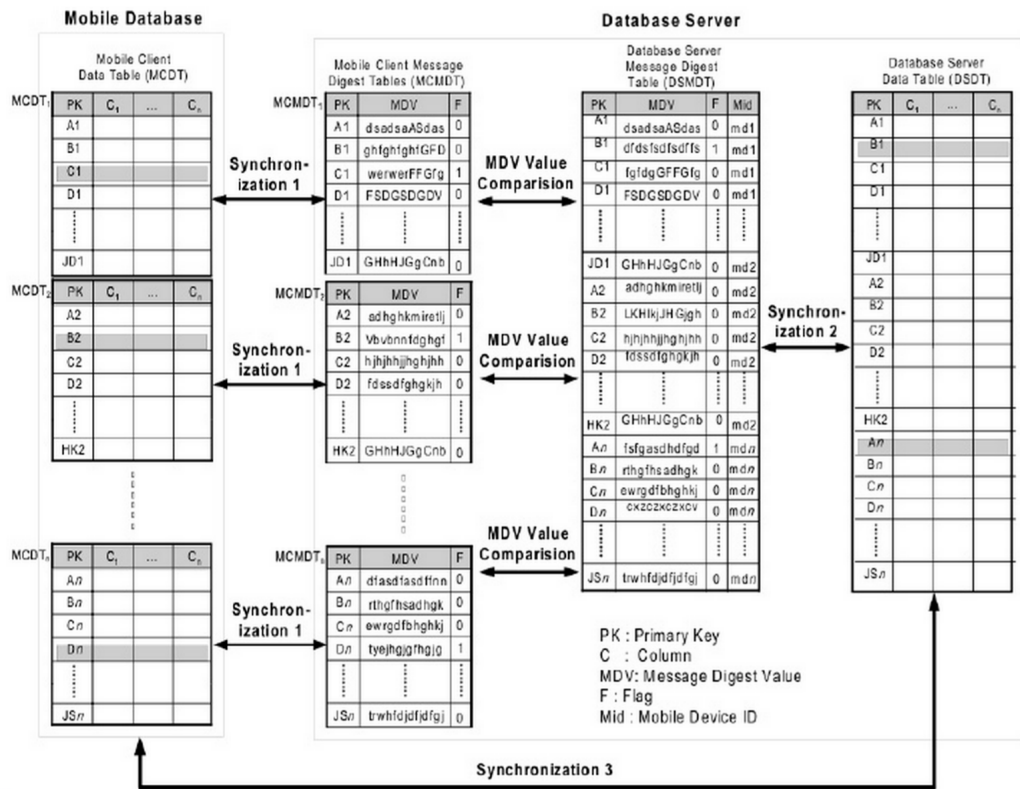


Figura 19: Estrutura do algoritmo SAMD (CHOI et al., 2010).

O algoritmo apresentado por CHOI difere do framework Offdroid no fato da sincronização do algoritmo ser unilateral, enquanto que no framework a sincronização pode ocorrer de forma bilateral. Outra diferença esta relacionada com o forma de verificação da tabela ou do registro que sofreu alguma atualização, pois no algoritmo SAMD essa verificação ocorre comparando o hash das tabelas, em havendo alteração uma flag será marcada. Já no framework essa mesma verificação ocorrerá em tempo de execução, sendo necessário apenas que o desenvolvedor selecione quais campos da classe devem ser verificados.

O algoritmo SAMD se assemelha ao Offdroid no modo de verificar se o dado sofreu ou não modificação, uma vez que no SAMD a verificação ocorre comparando os hash gerados pela tabela e no framework a verificação da semelhança entre os dados também ocorre comparando o hash gerado em tempo de execução.

Em (RUHE, 2012) é apresentada a solução denominada DB4o . Este framework destaca-se pela criação de soluções de persistência para outras plataformas, como o .NET da Microsoft e o Java da Oracle. O DB4a apresenta uma solução que se assemelha ao padrão de projeto Persistence Broker (SCHMIDT et al., 2013), onde é necessário implementar de uma interface para realizar as operações de persistência.

O Offdroid se assemelha com a solução de persistência apresentada por RUHE, que consiste na necessidade da classe de domínio realizar a implementação de uma interface para que se possa realizar as operações de persistência. Mas difere do RUHE quanto a abordagem genérica de persistência dos dados, uma vez que no framework é possível implementar uma nova abordagem para a persistência dos dados, sendo necessário apenas realizar a implementação da interface e inicializa-lo passando a nova classe responsável pela persistência dos dados.

## 7 Considerações Finais

O desenvolvimento deste trabalho foi baseado no crescente aumento da computação móvel no mundo, principalmente, diante do fato da população ter adquirido, em massa, aparelhos como os smartphones e tablets. Essa realidade gerou uma alta demanda na realização de *download* de aplicativos, criando a necessidade do uso destes em situações em que não houvesse conectividade.

Diante dos novos hábitos da sociedade em portar dispositivos móveis, a SINFO, assim como diversas outras empresas, vem enfrentando obstáculos na tentativa de migrar versões de seus sistemas para a computação móvel. Uma minuciosa análise foi realizada no intuito de dectetar as problemáticas enfrentadas por estas empresas, constatando-se que estes óbices consistiam na baixa qualidade da conectividade dos smartphones e tablet, na ausência de uma solução de replicação e sincronização de dados, e, pela não existência de um padrão de persistência.

A criação do Framework apresentado neste trabalho é a solução para as problemáticas encontradas, uma vez que o framework é capaz de realizar a persistência, a replicação e sincronização dos dados. Tais funcionalidades estão implementadas de forma não intrusiva, uma vez que há a possibilidade de configurar cada classe conforme desejado, ou seja, é possível ter classes de armazenamento apenas local; classes que não devem ser persistidas; classes que podem apresentar a persistência local e a realização da requisição junto ao servidor de aplicação. Para esses casos, ficará a cargo do framework verificar o estado da conectividade e assim definir se a operação será realizada utilizando os serviços REST's ou se a consulta ocorrerá na base de dados interna.

O framework foi desenvolvido e validado no aplicativo Coletor de Presença desenvolvido pela SINFO, que requisita os serviços junto ao servidor da aplicação e armazena as informações no banco do dispositivo. Este aplicativo foi selecionado em virtude da sua essência, que é a persistência, a replicação e a sincronização dos dados.

O framework atua de algumas maneiras neste aplicativo, tanto realizando a persistên-

cia das entidades necessárias para a realização da coleta das presenças, como realizando a sincronização destas presenças coletadas, uma vez que o dispositivo não esteja apresentando conectividade no ato da leitura.

Diante do estudo de caso apresentado, foi devidamente demonstrado que o framework desenvolvido é plenamente viável para solucionar as problemáticas listadas, pois ele já disponibiliza todas as funcionalidades de forma nativa, sem a necessidade de realizar nenhuma configuração adicional.

## 7.1 Contribuições

A criação do framework gerou diversos artefatos que podem ser citados como contribuições dessa dissertação. A seguir será listado as principais contribuições:

- Durante a definição das estratégias foi realizado um estudo para levantar os possíveis cenários inerente a computação móvel, em que, para cada um destes, foi criado uma estratégia para a configuração e manipulação dos objetos.
- Uma outra contribuição consistiu no projeto e desenvolvimento da solução do framework, sendo esse capaz de ter as suas funcionalidades estendidas para se adequar aos mais diversos cenários da computação móvel.
- A dissertação apresentou um experimento controlado onde foi apresentado os resultados obtidos para as três hipóteses levantadas, em que o framework apresentou um ganho de produtividade e redução do esforço necessário para o desenvolvimento, sem sobrecarregar a comunicação entre o dispositivo e o servidor de aplicação.
- Foi apresentado também uma avaliação que consistiu na migração de uma aplicação já existente e que não fazia uso do framework para uma nova versão, apresentando as mesmas funcionalidades porém, tendo sido feita fazendo uso do Offdroid.

## 7.2 Limitações

O presente trabalho apresenta algumas limitações referente a construção do framework. A biblioteca *h4android* está sob a licença LGPL, nesse caso o framework desenvolvido, por fazer uso do código fonte da API em questão, deverá apresentar o código fonte disponível e este deverá apresentar no mínimo a mesma licença LGPL.

Devido a baixa quantidade de participantes aptos para a realização do experimento não foi possível a realização de réplicas deste, por esse motivo não foi possível analisar e levantar novas questões que poderiam influenciar no resultado do experimento.

O framework *OffDroid* não passou por experimento ou por testes em outra abordagem que não fosse a de sistemas corporativos ou até mesmo em outro contexto que não o da SINFO, para ver se o mesmo atendia a todos os requisitos e especificidades do desenvolvedor ou da empresa.

Outra limitação encontrada no projeto do framework é a possibilidade de se trabalhar com os objetos sob demanda, ou seja, adicionar a hipótese de carregar determinados objetos a medida que for necessário, algo semelhante ao que ocorre *Lazy* e o *Eager* do hibernate. Atualmente o framework só trabalha com o mapeamento do tipo *Eager*, onde é retornado o objeto completo, ou seja, todos os seus relacionamentos do banco de dados local.

## 7.3 Trabalhos Futuros

Com relação aos trabalhos futuros, pode-se realizar mais estudos de casos baseados na implementação de uma solução do framework Offdroid em uma diferente abordagem tal com a de jogos, com o intuito de verificar se o Offdroid apresenta uma abordagem suficientemente genérica para se adaptar aos mais diversos cenários;

Também podem ser implementados estudos de casos baseados na implementação de uma solução do framework Offdroid em outras plataformas existentes, como Windows Phone e IOS, seguindo o modelo de persistência, sincronização e replicação dos dados como apresentados para a plataforma Android.

Outro trabalho futuro bastante relevante diz respeito a possibilidade de criação de query's customizadas para as classes que apresentam a anotação `@OnlyLocalStorage`, permitindo assim realizar consultas mais complexas na base de dados do dispositivo móvel.

Por fim, também é possível realizar a implantação do mecanismo de verificação e resolução de conflitos no Offdroid, para que durante o processo de sincronização o framework seja capaz de verificar e resolver possíveis conflitos.



# Referências

- AFFILIATES, O. and/or its. *For further API reference and developer documentation*. 2006. Disponível em: <<http://docs.oracle.com/javaee/7/index.html>>.
- ANDROID. *Introduction to Android*. 2014. Disponível em: <<http://developer.android.com/guide/index.html>>.
- BASILI, V. R.; SHULL, F.; LANUBILE, F. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, IEEE, v. 25, n. 4, p. 456–473, 1999.
- BISWAS, R.; ORT, E. *The Java Persistence API - A Simpler Programming Model for Entity Persistence*. 2006. Disponível em: <<http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html>>.
- CHOI, M.-Y. et al. A database synchronization algorithm for mobile devices. *Consumer Electronics, IEEE Transactions on*, IEEE, v. 56, n. 2, p. 392–398, 2010.
- DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004.
- DRIVE, G. *Google Drive REST API Overview*. 2015. Disponível em: <<https://developers.google.com/drive/>>.
- FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. *Building Application Frameworks: Object-oriented Foundations of Framework Design*. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN 0-471-24875-4.
- FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000.
- FILHO, I. de M. B. Projeto de Diplomação, *Desenvolvimento de aplicações móveis baseadas em sistemas de informações web existentes*. Natal, RN, Brasil: [s.n.], nov. 2014.
- FOCUS, M. *Gestores de TI projetam aumento de 50 aplicativos de negócios para dispositivos móveis, mas enfrentam desafios para migração*, <http://www.microfocus.com.br/sobre/pr/2013/pr02052013.aspx>. 2013. Disponível em: <<http://www.microfocus.com.br/sobre/pr/2013/pr02052013.aspx>>.
- GARGENTA, M. *Learning Android*. [S.l.]: O'Reilly Media, 2011. ISBN 978-1-4493-9050-1.
- GARTNER. *Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 19 Percent Growth in First Quarter of 2015*, <http://www.gartner.com/newsroom/id/3061917>. 2015. Disponível em: <<http://www.gartner.com/newsroom/id/3061917>>.

- GRØNLI, T.-M. et al. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In: *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*. [S.l.: s.n.], 2014. p. 635–641. ISSN 1550-445X.
- JOHNSON, R. E. Frameworks = (components + patterns). *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 10, p. 39–42, out. 1997. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/262793.262799>>.
- LEVIN, J.; FOX, J. A. *Elementary Statistics in Social Research*. [S.l.: s.n.], 2006.
- LOPES, V. P.; TRAVASSOS, G. H. Experimentação em engenharia de software: Glossário de termos. In: *Proceedings of 6th Experimental Software Engineering Latin American Workshop (ESELAW 2009)*. [S.l.: s.n.], 2009. p. 42.
- MICROSOFT. *Develop with the OneDrive API*. 2015. Disponível em: <<https://dev.onedrive.com/README.htm>>.
- PEREIRA, L. C. O.; SILVA, M. L. da. *Android Para Desenvolvedores*. [S.l.]: BRASPORT, 2012.
- RUHE, T. *Development of an Android Usage Study System*. Tese (Doutorado) — Master's Thesis, Hochschule Hannover, 2012.
- SCHMIDT, D. C. et al. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. [S.l.]: John Wiley & Sons, 2013.
- SEDIVY, J. et al. Mcsync - distributed, decentralized database for mobile devices. In: *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on*. [S.l.: s.n.], 2012. p. 1–6.
- SHOSHANI, A. et al. Characterization of temporal sequences in geophysical databases. In: *Scientific and Statistical Database Systems, 1996. Proceedings., Eighth International Conference on*. [S.l.: s.n.], 1996. p. 234–239.
- SMITH, B. *Introducing the Dropbox Sync API for mobile developers*. 2013. Disponível em: <<https://blogs.dropbox.com/developers/2013/02/introducing-the-dropbox-sync-api-for-mobile-developers/>>.
- SOWAH, R. A.; FIAWOO, S. Y. Design and development of a web service for android applications for extensive data processing. In: *International Journal of Engineering Research Technology*. [S.l.: s.n.], 2013. ISSN 2278-0181.
- SQLITE. *About SQLite*. 2015. Disponível em: <<https://www.sqlite.org/about.html>>.
- STAGE, A. Synchronization and replication in the context of mobile applications. *May*, v. 30, p. 1–16, 2005.
- TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. *Introdução à engenharia de software experimental*. [S.l.]: UFRJ, 2002.
- WOHLIN, C. et al. *Experimentation in software engineering: an introduction*. 2000. [S.l.]: Kluwer Academic Publishers, 2000.

WU, Y.; LUO, J.; LUO, L. Porting mobile web application engine to the android platform. In: *CIT*. IEEE Computer Society, 2010. p. 2157–2161. ISBN 978-0-7695-4108-2. Disponível em: <<http://dblp.uni-trier.de/db/conf/IEEEcit/IEEEcit2010.htmlWuLL10>>.