

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

**PROTÓTIPO DE SOFTWARE DE GERENCIAMENTO E
CONTROLE DE CHAMADOS TÉCNICOS PARA
DISPOSITIVOS MÓVEIS BASEADOS EM ANDROID**

ANDERSON NICOLAU HASKEL

BLUMENAU
2013

2013/1-06

ANDERSON NICOLAU HASKEL

**PROTÓTIPO DE SOFTWARE DE GERENCIAMENTO E
CONTROLE DE CHAMADOS TÉCNICOS PARA
DISPOSITIVOS MÓVEIS BASEADOS EM ANDROID**

Trabalho de Conclusão de Curso submetido à
Universidade Regional de Blumenau para a
obtenção dos créditos na disciplina Trabalho
de Conclusão de Curso II do curso de Ciências
da Computação — Bacharelado.

Prof. Francisco Adell Péricas - Orientador

**BLUMENAU
2013**

2013/1-06

PROTÓTIPO DE SOFTWARE DE GERENCIAMENTO E CONTROLE DE CHAMADOS TÉCNICOS PARA DISPOSITIVOS MÓVEIS BASEADOS EM ANDROID

Por

ANDERSON NICOLAU HASKEL

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

Presidente: Prof. Francisco Adell Péricas – Orientador, FURB

Membro: Prof. Mauro Marcelo Mattos, Doutor – FURB

Membro: Prof. Paulo Fernando Da Silva, Mestre – FURB

Blumenau, 08 de Julho de 2013

Dedico este trabalho a todos os amigos,
especialmente aqueles que me ajudaram
diretamente na realização deste.

AGRADECIMENTOS

Agradeço primeiramente Deus, pela vida, por estar sempre no meu caminho, iluminando e guiando meus passos.

Aos meus pais Eunildo Haskell e Bernadete Iaginski Haskell, que com carinho, apoio e incentivo, não mediram esforços para que eu chegasse até esta etapa de minha vida, apoiando-me e ensinando-me a persistir nos meus objetivos e ajudando a superar os obstáculos encontrados na vida pessoal e acadêmica.

Aos Professores, especialmente ao Prof. Francisco Adell Péricas, meu orientador, que com tanta presteza colaborou na realização e conclusão deste trabalho.

Aos amigos, e a todos aquele que participaram da minha caminhada dentro da universidade.

O sucesso torna as pessoas modestas,
amigáveis e tolerantes; é o fracasso que as faz
ásperas e ruins.

William Hazlitt

RESUMO

Este trabalho apresenta a especificação e desenvolvimento de um protótipo para as empresas de suporte técnico de informática, através de uso de dispositivos móveis. Na implementação do protótipo foram desenvolvidos: o aplicativo para dispositivo móvel sob a plataforma Android e um software em ambiente *web*. O grande diferencial deste trabalho é que permite que as informações sejam sincronizadas entre as aplicações. Possui também a capacidade efetuar a atualização de maneira *offline*.

Palavras-chave: Android. Dispositivo móvel. *Web service*.

ABSTRACT

This work presents the specification and development of a prototype to support companies computer technician, through the use of mobile devices. In the prototype implementation have been developed: the application to mobile Android platform and under the software in a web environment. The great advantage of this work is that it allows information to be synchronized between the applications. It also has the ability to perform the update offline way.

Keywords: Android. Mobile device. Web service.

LISTA DE ILUSTRAÇÕES

Figura 1 – recursos do software de gestão de chamados	16
Figura 2 - Exemplos de dispositivos móveis	17
Figura 3 - Smartphone Motorola com o Sistema Operacional Android.....	18
Figura 4 - Divisão do Sistema Operacional Android em Camadas	20
Figura 5 - Representação do funcionamento de uma Activity	23
Figura 6 - Ciclo de vida de um <i>Service</i> controlado via Android	25
Figura 7 - Principais características apresentadas pelo Web Service	27
Figura 8 - Estrutura de um web service REST	28
Figura 9 – Cenário do protótipo	32
Figura 10 – Diagrama de casos de uso da ferramenta smartphone executada pelo usuário.....	34
Quadro 1 – Caso de uso 01	35
Quadro 2 – Caso de uso 02	35
Quadro 3 – Caso de uso 03	35
Quadro 4 – Caso de uso 04	35
Quadro 5 – Caso de uso 05	36
Quadro 6 – Caso de uso 06	36
Figura 11 – Diagrama de casos de uso da ferramenta servidor executados pelo usuário.....	36
Quadro 7 – Caso de uso 01	37
Quadro 8 – Caso de uso 02	37
Quadro 9 – Caso de uso 03	37
Quadro 10 – Caso de uso 04	37
Figura 12 – Estrutura de pacotes do aplicativo Android	38
Figura 13 – Classes do pacote br.com.mcr.Activity	39
Figura 14 – Classes do pacote br.com.mcr.service.....	40
Figura 15 – Classes do pacote br.com.mcr.provider	41
Figura 16 – Classes do pacote br.com.mcr	42
Figura 17 – Classes do pacote br.com.mcr.helper	43
Figura 18 – Classes do pacote br.com.mcr.rest	43
Figura 19 – Classes do pacote br.com.mcr.rest.resource.....	44
Figura 20 – Classes do pacote br.com.mcr.rest.routes	45

Figura 21 – Classes do pacote br.com.mcr.util.....	46
Figura 22 – Estrutura de pacotes aplicação servidor	47
Figura 23 – MER do aplicativo Android	49
Figura 24 – MER do aplicativo servidor	50
Figura 25 – Diagrama de atividades do processo de Chamado Técnico	51
Figura 26 – Sincronismo com o aplicativo servidor.....	52
Figura 27 - Ambiente de desenvolvimento eclipse IDE - Android	53
Figura 28 – Smartphone Samsung Galaxy Note	54
Figura 29 – Ambiente de desenvolvimento eclipse IDE - Scala	55
Quadro 11 – método atualizar() da classe ChamadosListActivity	55
Quadro 12 – método atualizar() da classe McrServiceHelper.....	56
Quadro 13 – método onHandleIntent() da classe Service.....	57
Quadro 14 – método atualizar() da classe ChamadoProcessor	58
Figura 30 – Atividades do sincronismo	59
Figura 31 – Tela de login do servidor	60
Figura 32 – Tela de boas vindas	61
Figura 33 – Tela de lista de chamados	61
Figura 34 – Incluir chamado.....	62
Figura 35 – Listagem e inclusão de peças	63
Figura 36 – Editar um chamado	64
Figura 37 – Tela de login do smartphone	65
Figura 38 – Lista de chamados do smartphone	66
Figura 39 – Descrição do chamado do smartphone.....	67
Figura 40 – Alteração de status do smartphone.....	68
Figura 41 – Alteração de Chamado não sincronizado do smartphone	69
Figura 42 – Opção de sincronismo do smartphone	70
Quadro 15 – Comparação com trabalhos correlatos.....	71

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS DO TRABALHO	13
1.2 ESTRUTURA DO TRABALHO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 VISÃO GERAL DE CHAMADOS TÉCNICOS.....	15
2.2 DISPOSITIVOS MÓVEIS	16
2.2.1 SMARTPHONES	18
2.2.2 VANTAGENS	18
2.2.3 DESVANTAGENS.....	19
2.3 ANDROID.....	19
2.3.1 DEFINIÇÃO	20
2.3.2 PLATAFORMA ANDROID	20
2.3.3 ARQUITETURA DO APLICATIVO	21
2.3.4 CICLO DE VIDA DE UMA <i>ACTIVITY</i>	22
2.3.5 CICLO DE VIDA DE UM <i>SERVICE</i>	24
2.3.6 SQLite	26
2.4 WEB SERVICES	27
2.4.1 REST.....	27
2.5 SCALA	29
2.6 TRABALHOS CORRELATOS.....	30
2.6.1 WebAssist	30
2.6.2 Qtux controle e gestão de chamados técnicos.....	30
3 DESENVOLVIMENTO.....	32
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	32
3.2 ESPECIFICAÇÃO	33
3.2.1 Diagramas de casos de uso.....	34
3.2.1.1 Aplicativo smartphone.....	34
3.2.1.2 Aplicativo servidor	36
3.2.2 Diagrama de classes	38
3.2.2.1 Diagrama de classes do aplicativo smartphone	38
3.2.2.1.1 Pacote <code>br.com.mcr.Activity</code>	39

3.2.2.1.2	Pacote Classes do pacote <code>br.com.mcr.Service</code>	40
3.2.2.1.3	Pacote <code>br.com.mcr.Provider</code>	41
3.2.2.1.4	Pacote <code>br.com.mcr</code>	42
3.2.2.1.5	Pacote <code>br.com.mcr.helper</code>	43
3.2.2.1.6	Pacote <code>br.com.mcr.rest</code>	43
3.2.2.1.7	Pacote <code>br.com.mcr.rest.resource</code>	44
3.2.2.1.8	Pacote <code>br.com.mcr.rest.routes</code>	45
3.2.2.1.9	Pacote <code>br.com.mcr.util</code>	46
3.2.2.2	Aplicativo servidor	47
3.2.2.2.1	Pacote <code>Controllers</code>	47
3.2.2.2.2	Pacote <code>Models</code> e <code>Models.dto</code>	48
3.2.2.2.3	Pacote <code>Services</code>	48
3.2.2.2.4	Pacote <code>Exceptions</code>	48
3.2.2.2.5	Pacote <code>Views</code>	48
3.2.3	Banco de dados	49
3.2.4	Diagramas de atividade	51
3.3	IMPLEMENTAÇÃO	53
3.3.1	Técnicas e ferramentas utilizadas	53
3.3.1.1	Sincronismo	55
3.3.2	Operacionalidade da implementação	59
3.3.2.1	Aplicação servidor	59
3.3.2.1.1	Efetuar <i>login</i>	59
3.3.2.1.2	Listar chamados	60
3.3.2.1.3	Incluir chamado	61
3.3.2.1.4	Listar peças	63
3.3.2.1.5	Editar chamado	63
3.3.2.2	Aplicação smartphone	64
3.3.2.2.1	Efetuar <i>login</i>	64
3.3.2.2.2	Consultar chamados	65
3.3.2.2.3	Excluir chamados	66
3.3.2.2.4	Alterar status	67
3.3.2.2.5	Sincronizar	69
3.4	RESULTADOS E DISCUSSÃO	71

4 CONCLUSÕES.....	72
4.1 EXTENSÕES	72
REFERÊNCIAS BIBLIOGRÁFICAS	73

1 INTRODUÇÃO

A área de computação móvel está gerando uma nova e ampla frente de pesquisa e desenvolvimento, não só em Ciência da Computação, mas em outras áreas de conhecimento. A crescente proximidade da computação móvel com a telefonia celular e com a internet abriu ainda mais a possibilidade para a criação de novas tecnologias e, em consequência, de novos serviços e produtos.

As empresas de suporte técnico em informática têm dificuldade em acompanhar a atividade de seus técnicos, ou perceber problemas em reincidências de chamados. Em muitas empresas os atendimentos não têm sido bem avaliados quando solicitados, às vezes por ser um atendimento demorado. Em muitos casos o técnico não sabe nem o que vai atender. Tudo isso poderia ser melhorado com o uso de um software de controle de chamados.

Diante disso, as empresas têm investido em aplicações, utilizando-se de ferramentas *desktops* para esta finalidade, visando à verificação de sua produtividade, mas os técnicos passam maior parte do seu tempo fora da empresa, atendendo clientes, onde necessitam constantemente de informações e contatos com a empresa para atendê-los.

Buscando solucionar este tipo de problema em empresas de suporte técnico de informática, tanto fora como dentro da empresa, surgiu a ideia de desenvolver um software para o controle de chamados técnicos e gerenciamento em *smartphones*, para que as informações não estejam somente na empresa, mas também junto aos técnicos. Não basta somente o técnico ter acesso ao software, necessita-se também de um software para que as informações sejam sincronizadas e controladas com o software em tempo real, entre as bases de dados das aplicações.

Diante do exposto, pretende-se desenvolver um software para plataforma Android, e um software em ambiente web para gerenciamento e controle de chamados técnicos.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo de controle de chamados técnicos e gerenciamento para dispositivos móveis baseado no Sistema Operacional Android.

Os objetivos específicos do trabalho são:

- a) disponibilizar informações de chamados para usuários em tempo real;

- b) disponibilizar um software em ambiente web para controle dos chamados;
- c) disponibilizar um software para *smartphone* Android, cuja base de dados possa ser sincronizada com o software em ambiente *web*;
- d) permitir alterações dos dados no dispositivo móvel;
- e) permitir que o software do *smartphone* seja monitorado pelo software *desktop*.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos. O capítulo dois contém a fundamentação teórica necessária para o desenvolvimento do trabalho. Nele são abordados assuntos relacionados a uma visão geral de chamados técnicos, dispositivos móveis, arquitetura Android, *web services* e os trabalhos correlatos. O terceiro capítulo trata sobre o desenvolvimento desta ferramenta. O quarto capítulo refere-se às conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é realizada uma visão geral de chamados técnicos. A seção 2.2 aborda o conceito de dispositivos móveis e *smartphone*. A seção 2.3 apresenta a arquitetura Android incluindo os componentes de desenvolvimento da plataforma. A seção 2.4 contém uma introdução ao conceito de *web services* com a arquitetura *Representational State Transfer* (REST). A seção 2.5 apresenta a linguagem Scala. A seção 2.6 apresenta trabalhos correlatos ao trabalho proposto.

2.1 VISÃO GERAL DE CHAMADOS TÉCNICOS

Atualmente pode-se perceber que uma das características das empresas que buscam sucesso é a agilidade de solucionar rapidamente problemas de seus clientes e funcionários. Sendo assim, a adoção de um software para o registro e controle de chamados se faz necessário para um controle efetivo dos atendimentos, gerando um menor tempo de resposta nos chamados e um melhor atendimento aos clientes interno externos.

Se não existe um sistema central de controle de chamados, a qualidade dos serviços prestados começam a cair. As falhas acontecem, pois esses chamados técnicos podem ser perdidos ou esquecidos, podem ficar sob responsabilidade de uma única pessoa que não repassa para o setor responsável, ou algumas vezes é desconhecida a forma de se abrir um chamado de suporte técnico.

Segundo Aurélio (2008), a funcionalidade típica de um software de *help desk* inclui administração de telefonemas, acompanhamento de chamados, administração de base de conhecimento, resolução de problema, capacidades de auto-atendimento e consulta a banco de dados com as resoluções anteriores de problemas.

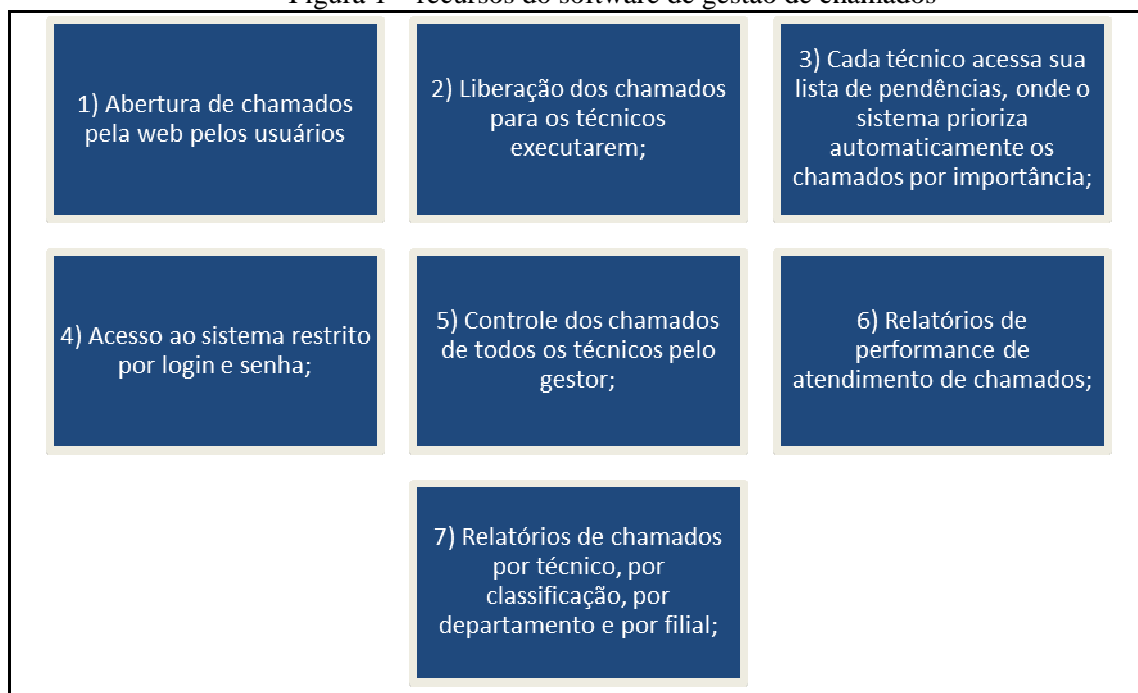
Ocasionalmente quando o pessoal da área técnica é acionado, através de telefone, o técnico marca o local da ocorrência e a hora. Este atendimento é feito por telefone ou desloca-se até o local da ocorrência. Após resolvido o problema, anota-se o motivo e a hora que foi finalizado e se foi ou não resolvido. Para isso não ocorrer, é interessante criar um canal para que os clientes ou usuários possam facilmente cadastrar seus problemas ou solicitações, e assim que for realizado, a área responsável seja comunicada o mais rápido possível.

De uma forma mais específica, o controle de chamados busca promover a gestão com foco no cliente e na qualidade dos serviços de tecnologia da informação (TI). Tendo como objetivo o endereçamento de estruturas de processos para a gestão de uma organização de TI

apresentando um conjunto abrangente de processos e procedimentos gerenciais, organizados em disciplinas, com os quais uma organização pode fazer sua gestão tática e operacional visando alcançar o alinhamento estratégico com os negócios. O controle dá uma descrição detalhada sobre importantes práticas de TI com *checklists*, tarefas e procedimentos que uma organização de TI pode customizar para suas necessidades (SECURE.NET, 2013).

Ainda de acordo com a Secure.net (2013) a instalação do software de interface *web* para a gestão do departamento de TI, pode incluir os seguintes recursos representados na figura 1.

Figura 1 – recursos do software de gestão de chamados



Fonte: adaptado de Secure. Net (2013).

2.2 DISPOSITIVOS MÓVEIS

De acordo com Posser (2006) um dispositivo móvel deve ter algumas características básicas, tais como: possuir um grau de mobilidade, tamanho reduzido, realizar processamento, trocar informações com alguma rede, independência de rede fixa.

Conforme Lecheta (2009, p. 3), o mercado de dispositivos móveis está crescendo cada vez mais. Estudos mostram que mais de 3 bilhões de pessoas possuem um aparelho celular, o que corresponda quase metade da população mundial.

O mercado corporativo também está crescendo muito e diversas empresas estão buscando incorporar aplicações móveis ao seu dia-a-dia para agilizar seus negócios e integrar as aplicações móveis com seus sistemas de *back-end*. Os celulares e *smartphones* podem ocupar um importante espaço em um mundo onde a palavra mobilidade está cada vez mais presente e mais conhecida (LECHETA, 2009, p. 19).

Segundo Fox (2003, p. 5), os dispositivos móveis frequentemente utilizados em processos de computação móvel tornaram-se pequenos computadores que facilmente são levados a qualquer lugar. Para as empresas, os dispositivos móveis são ótimos geradores de informações, podendo ser utilizados na automação de tarefas até a simples coleta de informações. A figura 2 apresenta exemplos de dispositivos móveis atuais.

Figura 2 - Exemplos de dispositivos móveis



2.2.1 SMARTPHONES

Segundo Führ (2007), *Smartphone* é um telefone celular com funcionalidades estendidas através de programas que podem ser carregados para rodarem no seu Sistema Usualmente um *Smartphone* possui características mínimas de hardware, sendo as principais: conexão por infra-vermelho e *bluetooth*, capacidade de sincronização dos dados do organizador com um computador pessoal e câmera para fotos e vídeos.

Ainda segundo Führ (2007) os *smartphones* são a combinação de duas classes de dispositivos móveis: os celulares e os *Personal Digital Assistants* (PDA), que além da função básica de telefonia, tem recursos que facilmente encontram-se em um computador, mas de forma mais compacta. A figura 3 representa um exemplo de Smartphone.

Figura 3 - Smartphone Motorola com o Sistema Operacional Android



Fonte: Android Developers (2013).

2.2.2 VANTAGENS

Segundo Schaefer (2004, p. 21), para aqueles que consomem grande parte do seu tempo trabalhando remotamente, estes equipamentos são versáteis, dedicados, multifuncionais e de uso genérico, e do ponto de vista empresarial, os dispositivos móveis são ótimos geradores de informação, podendo ser utilizados desde a automatização do processo até na coleta de informações estratégicas, pois com suas reduzidas dimensões podem estar sendo

transportados e estar presentes em todas as situações que um profissional dessa área pode atuar.

Os dispositivos móveis apresentam benefícios em comparação a dispositivos fixos. Pontos de ganho e benefícios são:

- a) baixo custo;
- b) portabilidade;
- c) flexibilidade de uso;
- d) disponibilidade de aplicativos de interação com outros sistemas.

De acordo com Pekus Consultoria e Desenvolvimento (2010), a cada dia as empresas estão recorrendo ao uso de tecnologias móveis em seus negócios para se tornarem mais ágeis, eficazes e competitivas. Seja dentro do próprio local de trabalho, seja através de redes de longa distância, os benefícios gerados pela mobilidade oferecida por dispositivos móveis são cada vez mais evidentes.

2.2.3 DESVANTAGENS

Segundo Martinelli (2011), os dispositivos móveis passaram a fazer parte do dia-a-dia das empresas com mais intensidade, e com isto existe a possibilidade de perder arquivos e dispositivos é muito maior.

A troca de dados por meio de dispositivos móveis ainda é feita com insegurança, haja visto que os aparelhos são suscetíveis a ataques maliciosos principalmente quando trata-se de um dispositivo de uso também pessoal, sem o controle da empresa e, consequentemente, torna-se muito mais vulnerável.

De acordo com Easy Binary Option (2013), a falta de rede é mais uma desvantagem, pois podem existir locais onde não há cobertura de rede, exemplo se você estiver em um trem ou em um elevador.

2.3 ANDROID

Desenvolvido pela *Open Handset Alliance*, Android é, segundo Android Developers (2013), um pacote de software para dispositivos móveis que inclui um sistema operacional, um *middleware* e aplicativos de características essenciais, dentre os quais se encontram o emulador e um *debugger*, juntamente com toda a biblioteca de desenvolvimento. Além disso,

é considerada uma plataforma móvel completa, livre e aberta (ANDROID DEVELOPERS, 2013), apresentando características incorporadas de outros sistemas, além de inúmeros conceitos inovadores para o segmento móvel.

2.3.1 DEFINIÇÃO

O Android é uma pilha de softwares ou software de arquitetura em pilha para dispositivos móveis que permite aos desenvolvedores criar aplicações para a plataforma usando o *Android Software Development Kit* (SDK) (ANDROID DEVELOPERS, 2013). As aplicações para esta plataforma são escritas usando a linguagem de programação Java e executam sobre o *Dalvik Virtual Machine* (DVM), uma máquina virtual customizada para dispositivos móveis com algumas restrições de recursos.

2.3.2 PLATAFORMA ANDROID

De acordo com a especificação do Android Developers (2013), a arquitetura do sistema operacional Android é dividida em quatro camadas, conforme mostra a figura 4.

Figura 4 - Divisão do Sistema Operacional Android em Camadas



Fonte: Aquino (2007, p.05).

Seguindo a divisão de camadas representada na figura 4, tem-se:

- a) primeira camada: Linux Kernel, funciona como uma camada de abstração entre o hardware e as demais camadas de software. É responsável por gerenciar a memória, processos, threads, segurança dos arquivos, pastas e drivers;
- b) segunda camada: Libraries contém as bibliotecas, escrita na linguagem de programação C/C++ usada por vários componentes do sistema e acessadas apenas através da camada de application framework. Ainda na segunda camada em paralelo com as bibliotecas encontra-se o Android runtime que inclui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java e máquina virtual Dalvik (DVM);
- c) terceira camada: *Application Framework* permite que o desenvolvedor tenha acesso completo as mesmas APIs que são usadas pelas aplicações da plataforma. O *Application Framework* foi desenvolvido para abstrair a complexidade e simplificar o reuso de componentes;
- d) quarta camada: *Application* é formada por todos os aplicativos do Android, como navegador web, contatos, cliente de e-mail e as aplicações desenvolvidas.

2.3.3 ARQUITETURA DO APLICATIVO

A *Dalvik Virtual Machine* (DVM) foi escrita de forma que um dispositivo possa executar múltiplas máquinas virtuais concorrentemente de maneira eficiente. Ela usa o *kernel* do Linux para prover a funcionalidade de múltiplas *threads* e gerenciamento de memória de baixo nível. Cada aplicação Android roda em seu próprio processo, com sua própria instância da máquina virtual. Isso garante que caso a aplicação apresente erros, ela possa ser isolada e removida da memória sem comprometer o resto do sistema (ANDROID DEVELOPERS, 2013).

Segundo Aquino (2007, p. 5) ao desenvolver as aplicações em Java, a DVM compila o *bytecode* (.class) e converte para o formato *Dalvik Executable* (.dex), que representa a aplicação do Android compilada. Depois disso, os arquivos .dex e outros recursos do projeto são compactados em um único arquivo com a extensão *Android Package File* (.apk), que representa a aplicação final.

De acordo com a definição forçada pela Android Developers (2013) um aplicativo Android consiste em uma ou mais das componentes a seguir:

- a) componente *activity* (atividade): geralmente representa um tela da aplicação.

Segundo esse contexto um aplicativo que possui uma tela visível é implementado

com uma *activity*. Quando um usuário seleciona um aplicativo da tela inicial ou de um ativador de aplicativo, uma *activity* é iniciada;

- b) componente *broadcast receivers* (receptores de difusão): representa uma mensagem da aplicação para o sistema operacional, ou seja a intenção da aplicação realizar determinada tarefa. Segundo esse contexto um aplicativo Android para processar um dado ou para responder a um evento como exemplo uma chamada;
- c) componente *service* (serviço): não apresenta interface com o usuário e especifica uma ação, no entanto, é capaz de continuar a ser executado mesmo quando em segundo plano. Um serviço pode ser usado por um aplicativo que precise persistir por um período de tempo, como exemplo um aplicativo de verificação de atualização;
- d) componente *content provider* (provedor de conteúdo): um provedor de conteúdo compõe um conjunto específico dos dados do aplicativo disponíveis a outros aplicativos, como exemplo os dados podem ser armazenados em um banco de dados SQLite ou qualquer outra maneira.

Segundo Aquino (2007, p. 121) nem toda aplicação necessita ter esses quatro blocos, mas uma aplicação Android deverá ser escrita com um ou mais desses blocos de construção. Uma vez decidido quais componentes são necessários à aplicação, é necessário listá-los em um arquivo chamado *AndroidManifest.xml*. Este arquivo deve conter os componentes da aplicação, além das capacidades e requerimentos para tais.

2.3.4 CICLO DE VIDA DE UMA *ACTIVITY*

Conforme Lecheta (2009, p.79), uma *activity* tem um ciclo de vida bem definido. Cada *activity* que é iniciada é inserida no topo de uma pilha, chamada de *activity stack*. Conforme o conceito de pilha a cada nova *activity* criada será inserida no topo da pilha, a *activity* que estava em execução fica abaixo da nova *activity*.

Uma *activity* essencialmente possui três estados possíveis. Uma *activity* que está no topo da pilha é a *activity* que esta em execução no momento encontra-se no estado execução, e as demais podem estar executando em segundo plano, estar no estado pausado ou totalmente parada, a seguir descreve-se cada estado:

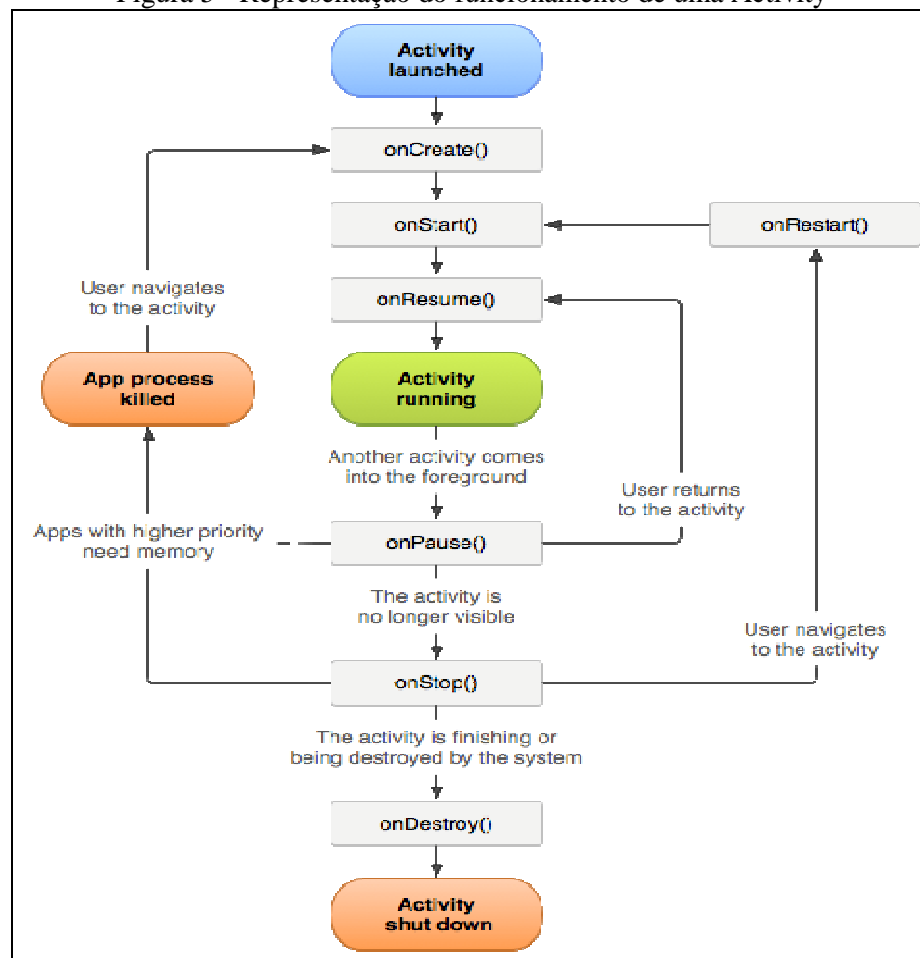
- a) estado execução: é o estado que *activity* está em primeiro plano na tela e tem o foco do usuário;

- b) estado pausado: é quando outra *activity* está em primeiro plano na tela e tem o foco do usuário. A *activity* que estava presente faz uma pausa, e essa *activity* que entrou em pausa continua em atividade sendo mantida pela memória todas as informações do estado pausado e permanecem ligados ao gerenciador de janelas;
- c) estado parado: é o estado que *activity* é completamente obscurecida por outra *activity*. Conforme o estado pausado a *activity* continua em atividade porem não permanece ligada ao gerenciador de janelas. Contudo, já não é mais visível para o usuário e pode ser morta pelo sistema quando memória for necessária.

Ainda segundo Lecheta (2009, p.79) sempre que uma *activity* está pausada o sistema operacional pode decidir encerrar o processo, por exemplo, para liberar recursos e memória para outros aplicativos.

A figura 5 ilustra os caminhos importantes de uma *activity*. Os retângulos representam chamados a métodos que podem implementar para realizar operações quando a *activity* muda entre os estados.

Figura 5 - Representação do funcionamento de uma Activity



Fonte: Android Developers (2013).

Seguindo o ciclo de vida de uma *activity* apresentado na figura 4, temos a descrição de cada método conforme Android Developers (2013):

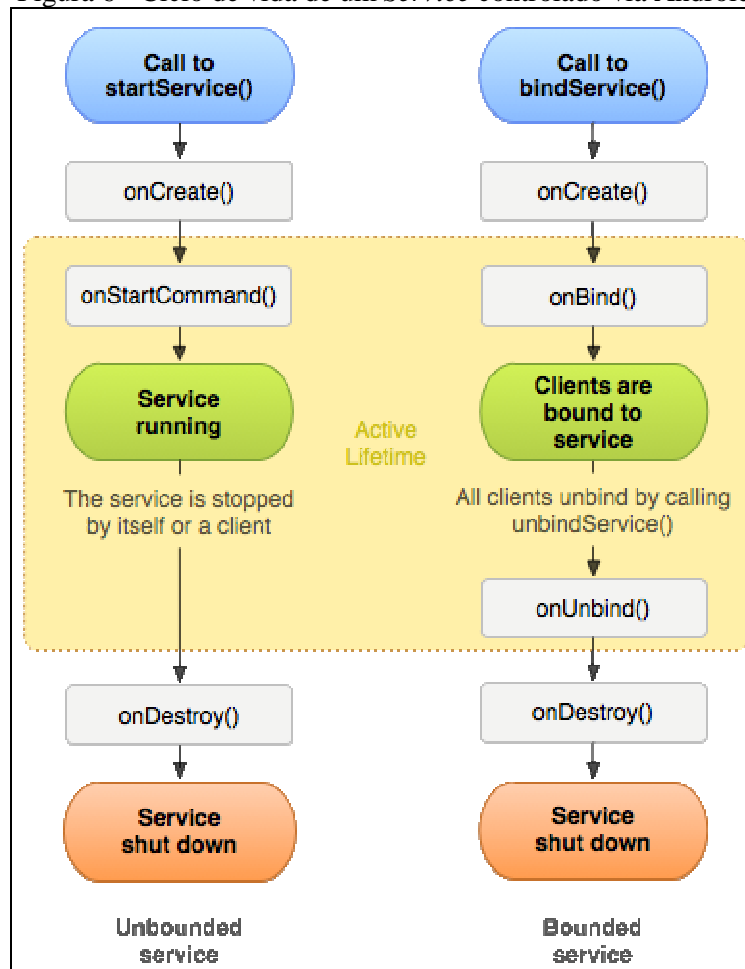
- a) método onCreate(): chamado quando a atividade é criada pela primeira vez. Este é o lugar onde se deve fazer todo o seu conjunto estático normal, onde criam *views*, vinculam dados a listas, e assim por diante;
- b) método onRestart(): chamado após a *activity* ser interrompida e antes de ser reiniciada;
- c) método onStart(): chamado quando a *activity* se torna visível ao usuário;
- d) método onResume(): chamado quando a *activity* vai iniciar a interação com o usuário, nesse momento a *activity* encontra-se no topo da pilha de atividades;
- e) método onPause(): chamado quando o sistema for resumir a *activity* anterior. Este método é geralmente usado para confirmar as alterações não salvas dados persistentes, parada de animações e outras coisas que podem estar consumindo CPU, e assim por diante;
- f) método onStop(): chamado quando a *activity* não estiver mais visível ao usuário, pois outra *activity* foi resumida e está na frente desta;
- g) método onDestroy(): chamado antes da *activity* ser destruída. Está é a ultima chamada que a *activity* irá receber.

2.3.5 CICLO DE VIDA DE UM SERVICE

De acordo com Android Developers (2013), um *service* é um componente da aplicação que pode realizar operações de longa duração em modo *background* e não provê interface de usuário. Outro componente de aplicação pode iniciar o *service* e ele vai continuar rodando em *background* mesmo que o usuário mude para outra aplicação.

A classe *Service* faz parte do ciclo de vida dos processos controlados pelo sistema operacional do Android, e seu processo não será finalizado enquanto estiver em execução, a não ser que as condições de memória do celular realmente baixas, situação na qual o Android pode decidir encerrar alguns processos para liberar recurso (LECHETA, 2009, p. 291).

Para exemplificar, a figura 6 apresenta, o ciclo de vida de um *service*.

Figura 6 - Ciclo de vida de um *Service* controlado via Android

Fonte: Android Developers (2013).

Um *service* essencialmente tem duas formas de ser executado:

- Started:** quando é chamado `startService()`. Uma vez iniciado, o serviço pode rodar em modo *background* indefinidamente, mesmo que o componente que o iniciou seja destruído. Usualmente, um serviço iniciado realiza uma tarefa única e não retorna o resultado para quem o chamou. Por exemplo, pode-se fazer um *download* ou *upload* de arquivos em uma rede. Quando a operação for terminada, o serviço termina sem a necessidade de interação do usuário;
- Bound:** chamado `bindService()`. Um serviço *bound* oferece uma interface cliente servidor que permite ao componente interagir com o serviço, enviar requisições, obter resultados e mesmo fazer alguns processos com comunicação entre processos. Um serviço *bound* roda apenas enquanto o componente da aplicação estiver ligado a ele.

Os demais métodos apresentados são:

- a) método `onStartCommand()`: chamado quando outro componente, como por exemplo uma *activity*, faz a requisição para que o serviço seja iniciado chamando `startService()`. Uma vez que esse método é executado, o serviço é iniciado e pode rodar em *background* indefinidamente. Se implementar isso é da sua responsabilidade parar o serviço uma vez que o trabalho está feito, chamando `stopSelf()` ou `stopService()`;
- b) método `onBind()`: chamado quando outro componente quer fazer o *bind* com o serviço (para realizar um *Remote Procedure Call* - RPC) chamando `bindService()`. Em sua implementação desse método deve prover uma interface que os clientes usarão para comunicar com o serviço, retornando um `IBinder`;
- c) método `onCreate()`: chamado quando o serviço é criado pela primeira vez, para realizar procedimentos de configuração iniciais (antes mesmo de chamar ou o `onStartCommand()` ou `onBind()`). Se o serviço já está rodando, esse método não é chamado;
- d) método `onDestroy()`: chamado quando o serviço não está mais sendo usado e está sendo destruído. Seu serviço deve implementar esse método para limpar os recursos como *threads*, *listeners* registrados, *receivers*, etc. Essa é a última chamada que o serviço recebe.

2.3.6 SQLite

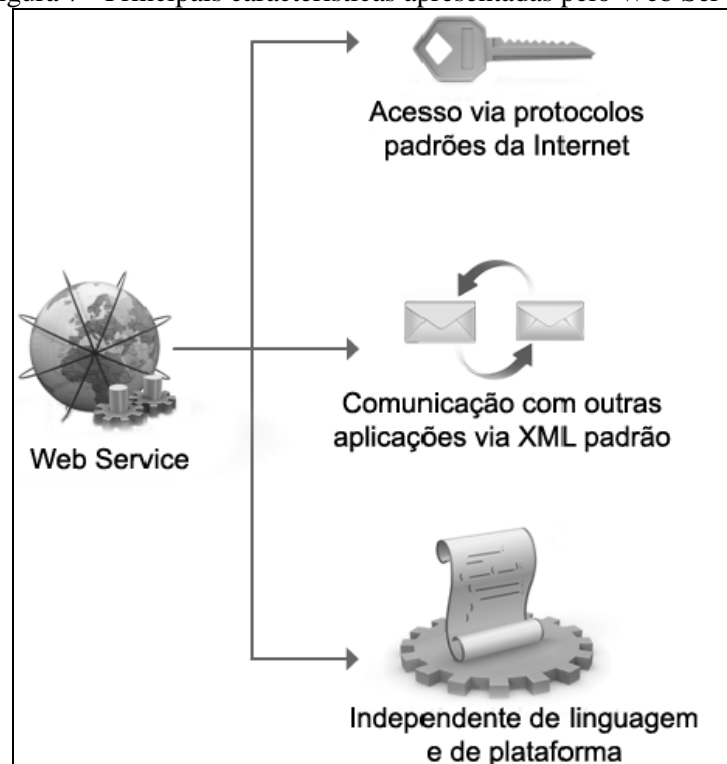
De acordo com SQLite (2013), o SQLite é uma biblioteca que implementa mecanismo transacional de dados do SQL. Ele funciona como um mecanismo de banco de dados embutido, e diferentemente da maioria dos outros bancos de dados SQL, não possui um servidor separado, não requer instalação ou administração e não há arquivos de configuração. Ainda segundo SQLite (2013), o SQLite lê e escreve diretamente para arquivos do disco e o formato do banco de dados é multi-plataforma.

2.4 WEB SERVICES

Segundo Menéndez (2002), um *web service* é um sistema de software desenvolvido para suportar interoperabilidade entre máquinas sobre uma rede.

O uso de um web service possibilita que aplicações diferentes interajam entre si em que sistemas desenvolvidos em plataformas diferentes se tornem compatíveis, ou seja os *Web services* são componentes que permitem que aplicações enviem e recebam dados de formatos variados. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal. A figura 7 mostra uma visão geral das principais características de um web service.

Figura 7 - Principais características apresentadas pelo Web Service



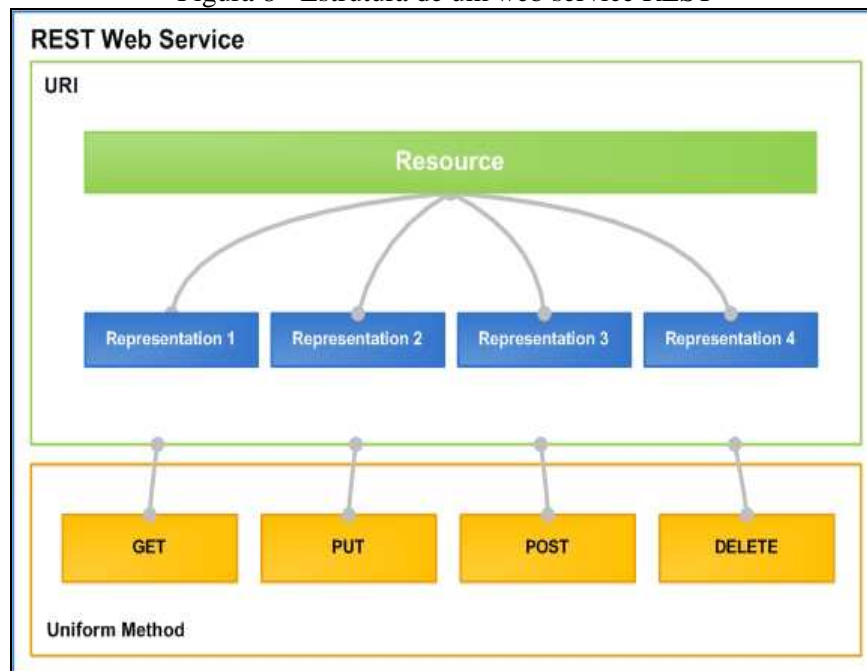
Fonte: Damasceno (2009).

2.4.1 REST

O REST é um estilo de arquitetura para comunicação baseada na web que permite aos clientes conversar com os servidores de maneira única. Em particular, a REST representa recursos dentro de um dado servidor, como *Uniform Resource Identifier* (URI), onde os recursos são manipulados através dos métodos do *Hypertext Transport Protocol* (HTTP), que são representados por um formato padrão (RICHARDSON; RUBY, 2007).

A figura 8 tem como objetivo representar a estrutura do *web service* REST.

Figura 8 - Estrutura de um web service REST



Fonte: Apach (2009).

De acordo com a representação da figura 8, tem-se:

- a) *Interface Uniforme*: é a característica central que diferencia o estilo arquitetural REST de outros estilos baseados em rede, e está baseada em uma *interface uniforme* entre os componentes (cliente, servidor). O HTTP é o protocolo utilizado pelo REST, para prover uma *interface uniforme* com quatro métodos básicos para as quatro operações mais comuns. GET para recuperar uma representação de um recurso, PUT para criar um novo recurso ou modificar um existente, DELETE para deletar um recurso e POST comumente utilizado para criação de um novo recurso. O protocolo HTTP é ainda muito útil na troca de mensagens nos serviços REST, retornando códigos de status na resposta a uma requisição;
- b) URI: este tem como objetivo identificar o recurso a ser utilizado e a sua representação identificada é apresentada em formato legível para que a aplicação seja de fácil entendimento, como exemplo um documento em XML ou JSON.

2.5 SCALA

A linguagem Scala foi desenvolvida pelo pesquisador Martin Odersky na Escola Politécnica Federal de Lausana (EPFL), na Suíça por volta de 2001. Foi liberada publicamente na plataforma Java em janeiro de 2004, uma segunda versão da linguagem foi liberada em março de 2006.

Scala é similar a linguagem Java, e roda na Java Virtual Machine, permitindo ser compatível com programas java existentes. Assim como a linguagem Java é puramente orientada a objetos no sentido de que todo valor é um objeto. Tipos e comportamentos de objetos são definidos por classes.

De acordo Scala (2013), é uma linguagem funcional no sentido de que cada função é também um valor. Tem uma sintaxe limpa e fácil para definir funções anônimas, que suporta *higher-order functions*, permite funções encadeadas e suporta *currying*.

Ainda conforme Scala (2013), a linguagem Scala é equipado com um sistema de tipos expressivo, que reforça estáticamente que abstrações sejam usadas de maneira segura e coerente. Em particular, o sistema de tipos suporta:

- a) programação genérica;
- b) anotações de variância;
- c) superior e inferior tipo acoplado;
- d) classes e enumeração (tipo de dado);
- e) tipo composto;
- f) auto referencia;
- g) métodos polimórficos.

O Scala foi projetado para interoperar com ambientes de programação, e com linguagens orientadas a objeto, exemplos Java e C#. Permitindo ainda fazer uso de todas as bibliotecas disponíveis para Java e C#.

2.6 TRABALHOS CORRELATOS

Algumas ferramentas desempenham papel semelhante ao proposto. Dentre elas foram selecionadas: Web Assist (UTECS, 2010) e Qtux controle e gestão de chamados técnicos (QTUX, 2013).

2.6.1 WebAssist

O WebAssist (UTECS, 2010) é um software para gerenciamento de chamados e ordens de serviços desenvolvido pela União de Tecnologia e Serviços. O sistema segue o modelo *Software as a Service* (SaaS), sem a necessidade de instalação de algum programa em sua estrutura, pois é baseado na web, onde o acompanhamento e o gerenciamento da equipe técnica e o controle dos serviços são realizados através do navegador de internet com as principais funções de gerenciamento de serviços técnicos e monitoramento de chamados. As funcionalidades do *software* são:

- a) controle de chamados de campo;
- b) controle de equipamentos em contrato;
- c) controle de produtividade técnica;
- d) controle de SLA;
- e) controle por usuários/regiões/unidades;
- f) controle de parque com inventário de equipamentos;
- g) controle por usuário
- h) controle de chamados de laboratório;
- i) controle individual por tipo de chamados;
- j) alerta de prazos de atendimento;
- k) estatísticas de chamados;
- l) atualização dinâmica e automática de chamados.

2.6.2 Qtux controle e gestão de chamados técnicos

O Qtux (QTUX, 2013), é um *software* desenvolvido pela Qtux Tecnologia da Informação, cujo propósito é ajudar as empresas de prestação de serviços técnicos a gerenciar e controlar chamados técnicos afim de aumentar a qualidade dos serviços prestados. O

sistema é desenvolvido sobre a plataforma web 2.0, que de acordo com a Qtux (2013), o sistema possibilita ao cliente as seguintes funcionalidades:

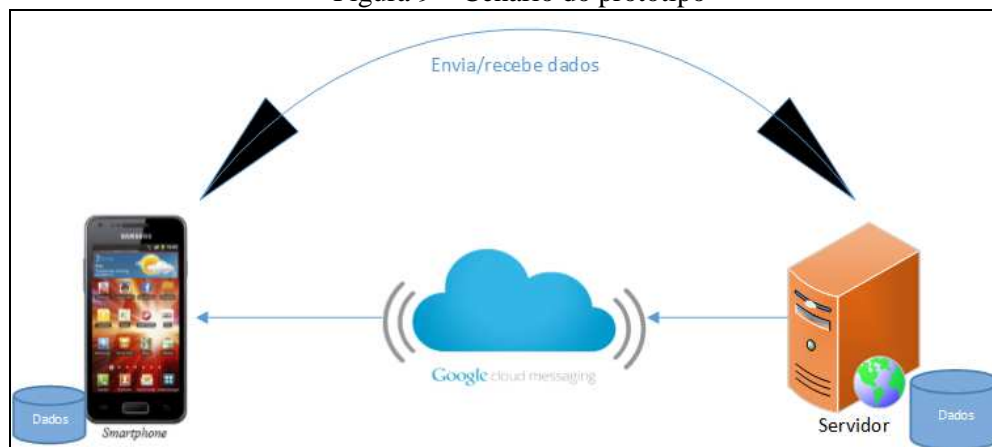
- a) solicitar uma visita técnica selecionando o período do dia que quer ser atendido;
- b) acompanhar o andamento do chamado;
- c) avaliar o atendimento;
- d) checar os chamados fechados;
- e) avaliação real do serviço;
- f) mostrar todos os chamados abertos, em andamento, realizados;
- g) alocar técnicos a chamados;
- h) acompanhar a execução dos serviços.

3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas do desenvolvimento da ferramenta. São apresentados os requisitos principais, a especificação, a implementação e por fim são listados os resultados e discussão.

Para um melhor entendimento do protótipo é apresentado o cenário do protótipo na figura 9. Este é composto por dois componentes, o servidor este uma aplicação em ambiente web, e um *smartphone* este uma aplicação cliente rodando no sistema operacional Android. A comunicação entre as aplicações é utilizado a arquitetura REST e o serviço Google *cloud messaging* para comunicar a aplicação cliente de novos recursos no servidor.

Figura 9 – Cenário do protótipo



3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O sistema é composto por duas partes: um aplicativo para *smartphone* e um software em ambiente web (servidor).

Smartphone, os Requisitos Funcionais (RFs) do aplicativo *smartphone* são:

- a) RF01: permitir consultar a lista de chamados;
- b) RF02: permitir consultar a lista de peças solicitada;
- c) RF03: permitir gerar registros no banco de dados;
- d) RF04: permitir sincronizar com o software em ambiente *web*;
- e) RF05: permitir atualizar os chamados (*status*);
- f) RF06: permitir excluir chamados;
- g) RF07: permitir trabalhar *off-line*;
- h) RF08: permitir efetuar *login*.

Os requisitos Não Funcionais (RNFs) do aplicativo *smartphone* são:

- a) RNF01: utilizar banco de dados SQLite;
- b) RNF02: ser implementa utilizando o ambiente de desenvolvimento no Android SDK.

Quanto ao software servidor, os requisitos funcionais são:

- a) RF09: permitir enviar chamados para o software *smartphone*;
- b) RF10: permitir receber informações do software *smartphone*;
- c) RF11: permitir alterar informações do chamados;
- d) RF12: permitir incluir chamados;
- e) RF13: permitir excluir chamados;
- f) RF14: permitir efetuar *login*;
- g) RF15: permitir consultar a lista de peças.

Os requisitos não funcionais do software em ambiente *web* são:

- a) RNF03: utilizar banco de dados MySQL.

3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida utilizando diagramas da *Unified Modeling Language* (UML) e de Modelos de Entidades e Relacionamentos (MER). Os casos de uso são apresentados através de diagrama de casos de usos, seguindo da descrição do cenário de cada um. Em seguida, as classes da aplicação *smartphone* e servidor são apresentadas através de diagramas de classes. As entidades de banco de dados do *smartphone* e do servidor são demonstradas através de diagramas MER. Como forma de facilitar a implementação e entendimento, dois processos importantes do sistema foram representados através de diagrama de atividades.

A ferramenta utilizada para gerar os diagramas da UML foi o *Enterprise Architect* versão 8.0 para Windows. O diagrama MER do servidor foi desenvolvido utilizando a ferramenta MySQL Workbench 5.2 CE e o MER do *smartphone* foi utilizado o SQLite Maestro para Windows.

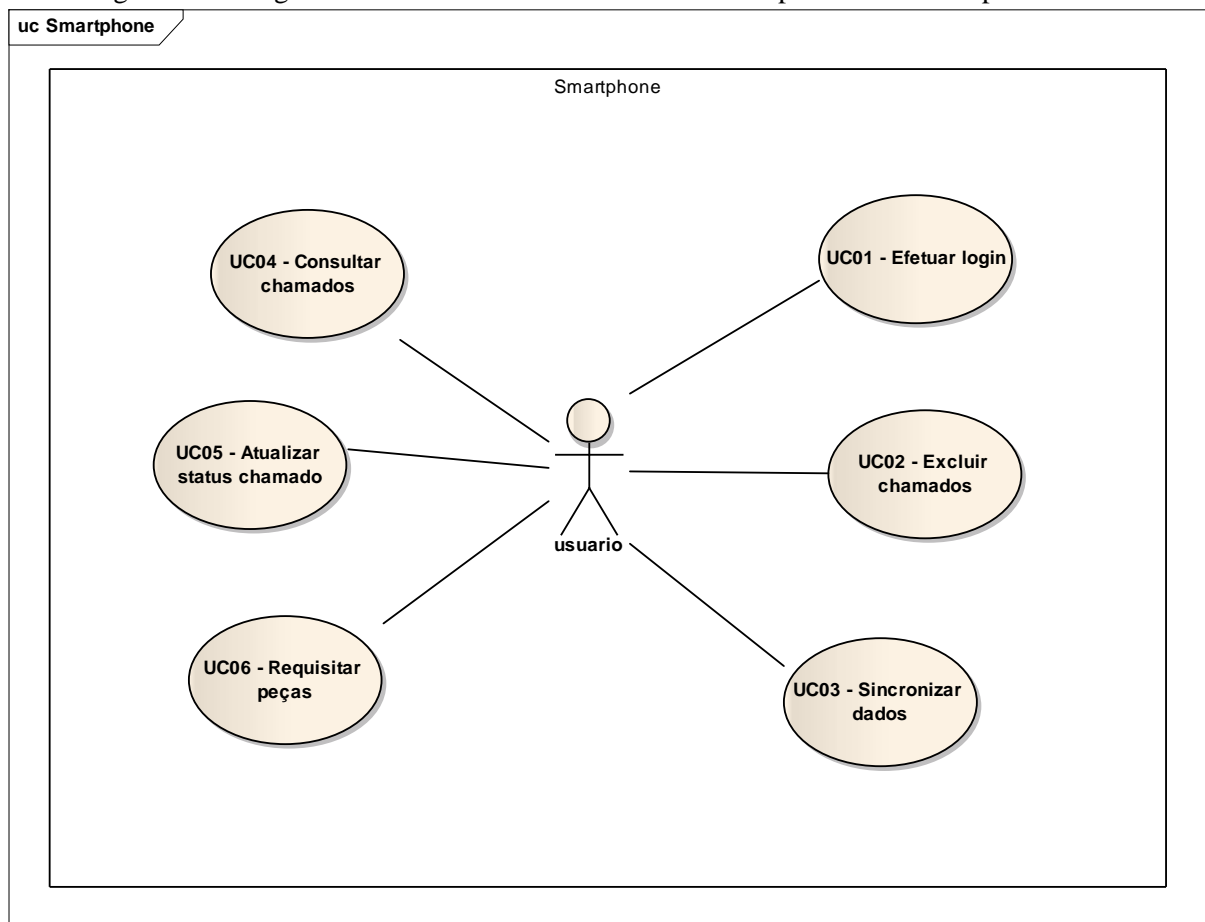
3.2.1 Diagramas de casos de uso

Os diagramas de casos de uso estão divididos em dois cenários: *smartphone* e servidor. A seguir são apresentados os diagramas de caso de uso do *smartphone* e do servidor, detalhando os principais.

3.2.1.1 Aplicativo smartphone

A figura 10 apresenta os casos de uso do *smartphone*, que são executados pelo usuário da ferramenta.

Figura 10 – Diagrama de casos de uso da ferramenta smartphone executada pelo usuário



São detalhados os seguintes casos de uso: Efetuar login (quadro 01), Excluir chamado (quadro 02), Sincronizar dados (quadro 03), Consultar chamado (quadro 04), Atualizar status chamado (quadro 05), Requisitar peças (quadro 06).

Quadro 1 – Caso de uso 01

UC01 – Efetuar login	
Requisitos atendidos	RF08
Pré-condições	O usuário possuir cadastro na base de dados do servidor.
Cenário Principal	01) O usuário informa endereço de e-mail e a senha. 02) Clica no botão login.
Fluxo alternativo	Caso não tenha conexão com a internet e o usuário já tenha efetuado o login, o acesso é efetuado de modo off-line através do banco de dados do smarphphone.
Exceção 01	No passo 01, se for inserido um usuário invalido ou senha, é gerado uma mensagem de erro.
Pós-Condição	O banco de dados é sincronizado com a aplicação servidor.

Quadro 2 – Caso de uso 02

UC02 – Excluir chamado	
Requisitos atendidos	RF06
Pré-condições	O usuário deve ter efetuado o caso de uso 01.
Cenário Principal	01) O usuário selecionada um chamado da lista de chamados. 02) O Usuário clica na opção excluir.
Pós-Condição	O chamado é excluído da base de dados do smartphphone.

Quadro 3 – Caso de uso 03

UC03 – Sincronizar dados	
Requisitos atendidos	RF04
Pré-condições	O usuário deve ter efetuado o caso de uso 01.
Cenário Principal	01) O usuário seleciona a opção sincronizar.
Exceção 01	Caso não tenha conexão com a internet o sincronismo ficara esperando uma conexão estabelecer e efetuar o processo
Exceção 02	O caso o servidor não esteja em execução, é gerando uma mensagem de erro.
Pós-Condição	É gerado uma mensagem informando que foi efetuado com sucesso.

Quadro 4 – Caso de uso 04

UC04 – Consultar chamado	
Requisitos atendidos	RF01
Pré-condições	O usuário deve ter efetuado o caso de uso 01.
Cenário Principal	01) O usuário visualiza os chamados em uma lista de chamados. 02) O usuário seleciona um chamado da lista.
Pós-Condição	É exibido as informações referente ao chamado selecionado.

Quadro 5 – Caso de uso 05

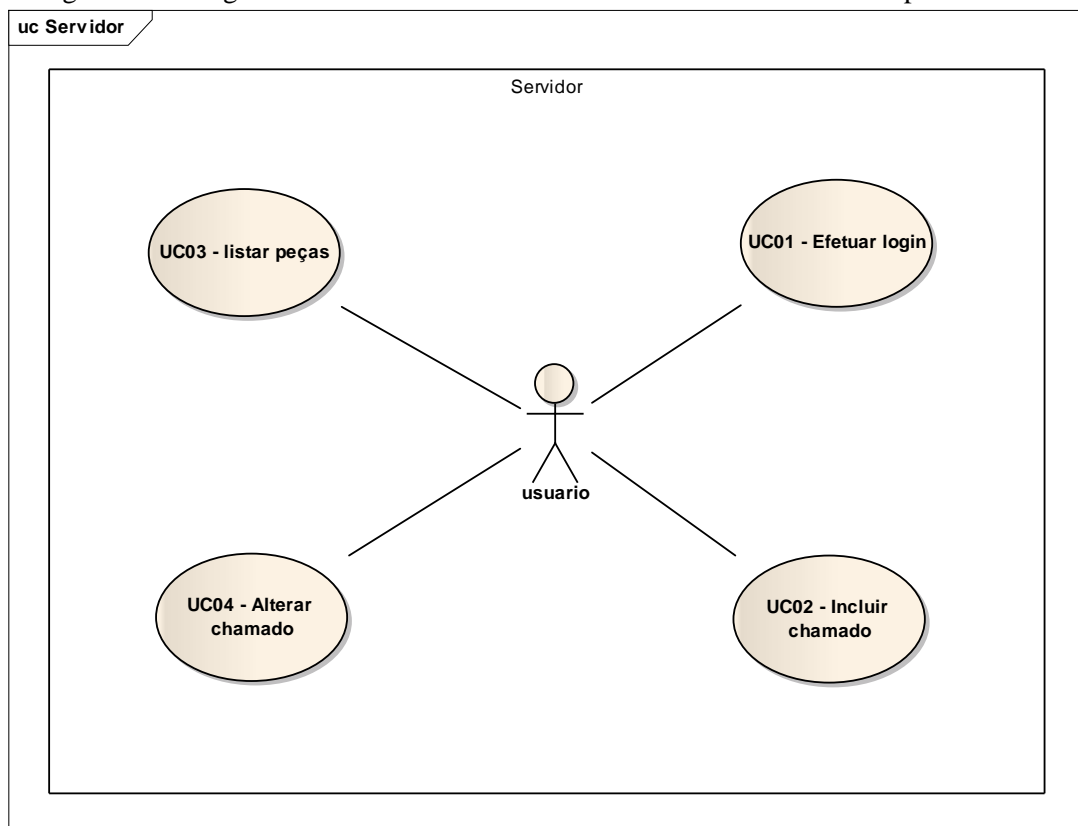
UC05 – Alterar status do chamado	
Requisitos atendidos	RF05
Pré-condições	O usuário deve ter efetuado o caso de uso 01.
Cenário Principal	01) O usuário seleciona um chamado da lista de chamados. 02) O usuário altera o campo status (Iniciado, em execução, pendente, finalizado).
Fluxo alternativo	
Pós-Condição	A informação é alterada e sincronizada.

Quadro 6 – Caso de uso 06

UC05 – Alterar status do chamado	
Pré-condições	O usuário deve ter efetuado o caso de uso 01.
Cenário Principal	01) O usuário seleciona um chamado da lista de chamados. 02) O usuário seleciona a peça e a quantidade solicitada.
Fluxo alternativo	
Pós-Condição	A informação é alterada e sincronizada.

3.2.1.2 Aplicativo servidor

Figura 11 – Diagrama de casos de uso da ferramenta servidor executados pelo usuário.



Os caso de uso que são detalhados: Efetuar login (quadro 07), Incluir chamado (quadro 08), Listar peças (quadro 09), Alterar chamado (quadro 10).

Quadro 7 – Caso de uso 01

UC01 – Efetuar login	
Requisitos atendidos	RF14
Pré-condições	O usuário possuir cadastro na base de dados do servidor.
Cenário Principal	01) O usuário informa endereço de e-mail e a senha. 02) Seleciona o botão login.
Exceção 01	No passo 01, se for inserido um usuário inválido ou senha, é gerado uma mensagem de erro.
Pós-Condição	Visualização da tela inicial.

Quadro 8 – Caso de uso 02

UC02 – Incluir chamado	
Requisitos atendidos	RF12
Pré-condições	O usuário deve ter efetuado o caso de uso 01
Cenário Principal	01) O usuário seleciona o botão chamados. 02) O usuário seleciona o botão novo. 03) O usuário preenche as informações obrigatórias (descrição, cliente, status, endereço, prioridade, técnico). 04) O usuário seleciona o botão salvar.
Exceção 01	No passo 04, se não for preenchido os campos obrigatórios é gerado uma mensagem de erro.
Pós-Condição	É exibida a lista de chamados.

Quadro 9 – Caso de uso 03

UC03 – Listar peças	
Requisitos atendidos	RF15
Pré-condições	O usuário deve ter efetuado o caso de uso 01
Cenário Principal	01) O usuário seleciona um chamado da lista de chamados. 02) O usuário seleciona o botão listar.
Pós-Condição	É exibida a lista de peças do chamado.

Quadro 10 – Caso de uso 04

UC03 – Alterar chamado	
Requisitos atendidos	RF11
Pré-condições	O usuário deve ter efetuado o caso de uso 01
Cenário Principal	01) O usuário seleciona um chamado da lista de chamados. 02) O usuário seleciona o botão editar.
Pós-Condição	É exibida a lista de chamados.

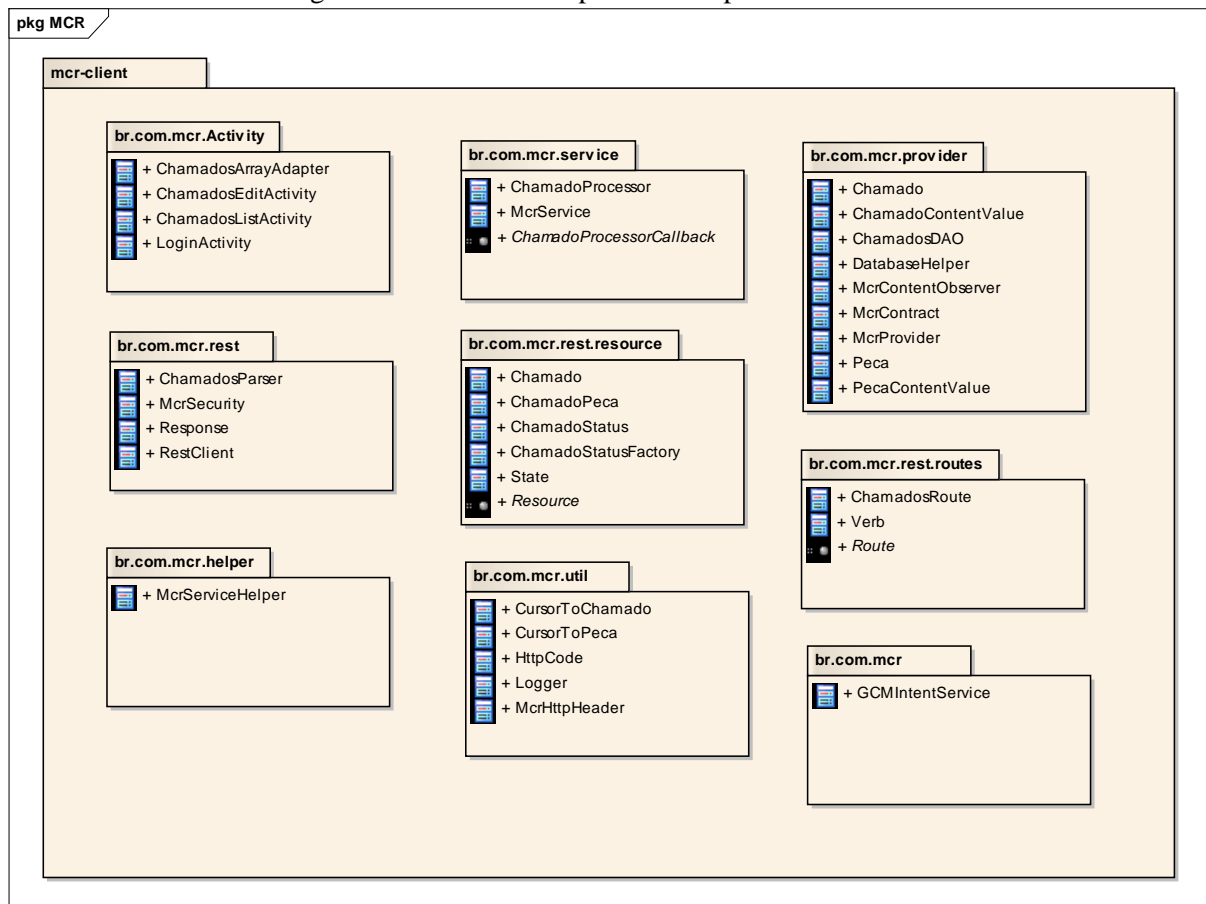
3.2.2 Diagrama de classes

O diagrama de classes facilita uma visão de como as classes estão estruturadas e relacionadas. De forma a facilitar a estruturação e a relação entre as classes, são apresentado diagrama de pacotes do aplicativo smartphone e servidor e descritas as classes necessárias para o desenvolvimento das aplicações.

3.2.2.1 Diagrama de classes do aplicativo smartphone

Nesta seção são descritas as classes necessárias para o desenvolvimento do aplicativo cliente. Para facilitar a visualização e que favoreça o melhor entendimento, a figura 12 exibe os pacotes que compõem a aplicação servidor.

Figura 12 – Estrutura de pacotes do aplicativo Android

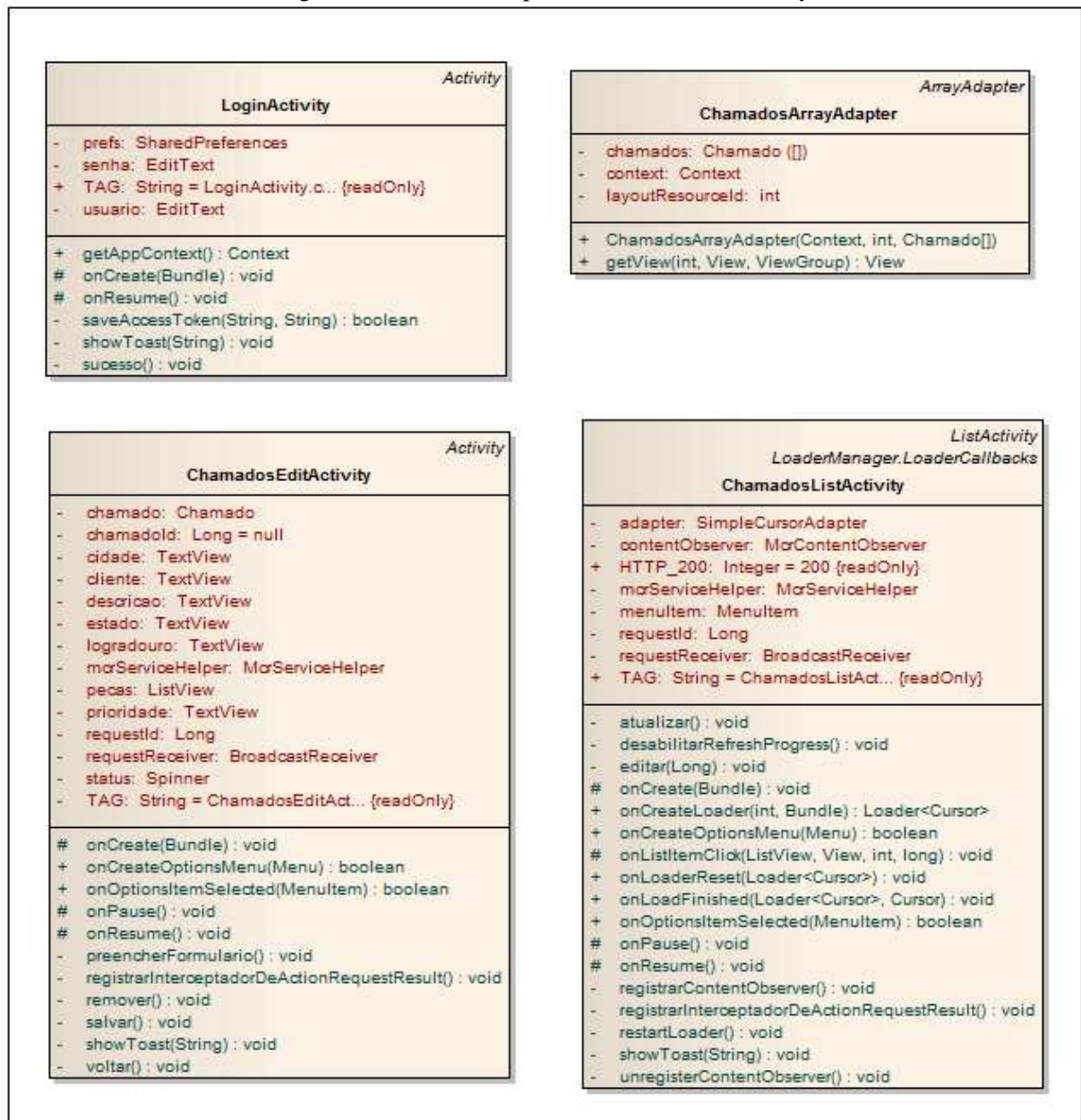


Nas seções seguintes são explanados cada pacote, bem como as classes que os compõem com suas respectivas responsabilidades.

3.2.2.1.1 Pacote `br.com.mcr.Activity`

O pacote `br.com.mcr.Activity` é composto pelas classes responsáveis por controlar os eventos e definir as telas exibidas ao usuário. A figura 13 ilustra as classes do pacote.

Figura 13 – Classes do pacote `br.com.mcr.Activity`



A classe `LoginActivity` é responsável por permitir o usuário efetuar o *login* com as informações de e-mail e senha e validar as informações enviando para o servidor e recebendo a validação pelo servidor através de um *token* e o *usuarioId*. A classe

ChamadosListActivity é responsável por atualizar os chamados bem como solicitar a sincronização.

A classe ChamadosEditActivity é responsável por apresentar as informações do chamados, assim como alterar, salvar e excluir os chamados, já a classe ChamadosArrayAdapter é responsável por mostrar os chamados na tela.

3.2.2.1.2 Pacote Classes do pacote br.com.mcr.Service

O pacote br.com.mcr.Service é composto pelas classes que executam tarefas em segundo plano na aplicação. A figura 14 ilustra as classes do pacote Service.

Figura 14 – Classes do pacote br.com.mcr.service



A classe ChamadoProcessor é responsável por analisar e tratar as requisições solicitadas pelo usuário referente aos chamados.

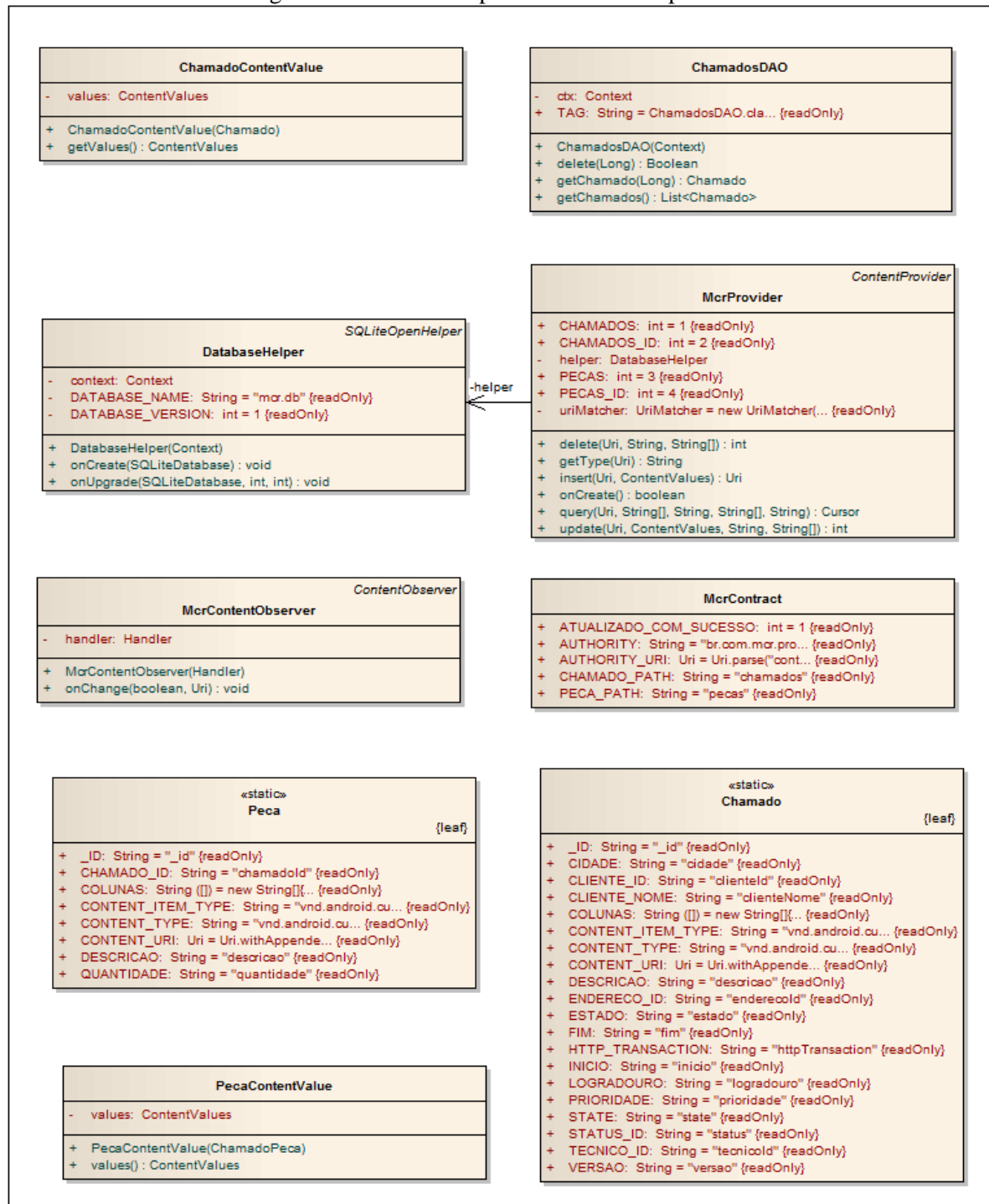
A classe ChamadoProcessorCallback é responsável por enviar uma resposta a classe McrService avisando do fim do processo.

A classe McrService é responsável por verificar que tipo de requisição está sendo desejada e enviar para a classe ChamadoProcessor tratar a solicitação.

3.2.2.1.3 Pacote br.com.mcr.Provider

O pacote Provider compoem as classes que são responsáveis pelo banco de dados local do Android.

Figura 15 – Classes do pacote br.com.mcr.provider



As classes `ChamadosContentValue` e `PecaContentValue` são classes helper responsáveis por criar um objeto `ContentValues` que é usado para inserir dados no banco de dados.

A classe `ChamadosDAO` é uma classe DAO responsável por buscar dados do banco de dados e inserir dados no banco.

A classe `DatabaseHelper` é responsável por criar o banco de dados caso não esteja criado.

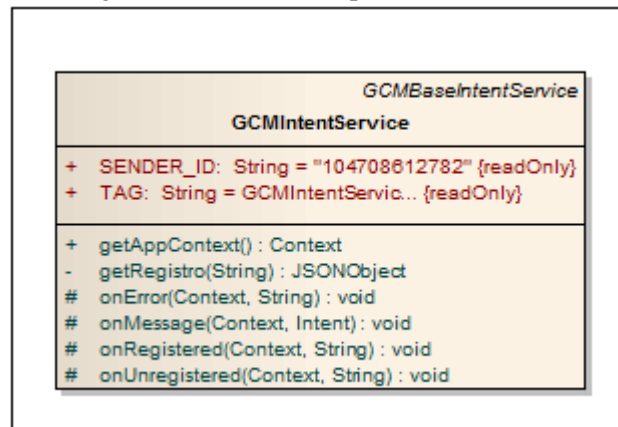
A classe `McrContentObserver` é responsável por observar um evento de mudança no banco de dados, para informar a classe `activity` que foi realizado o *uptade* no banco de dados com sucesso.

A classe `McrContract` é um utilitário para definir o contrato do banco de dados local.

A classe `McrProvider` é responsável por realizar as queries do banco de dados.

3.2.2.1.4 Pacote `br.com.mcr`

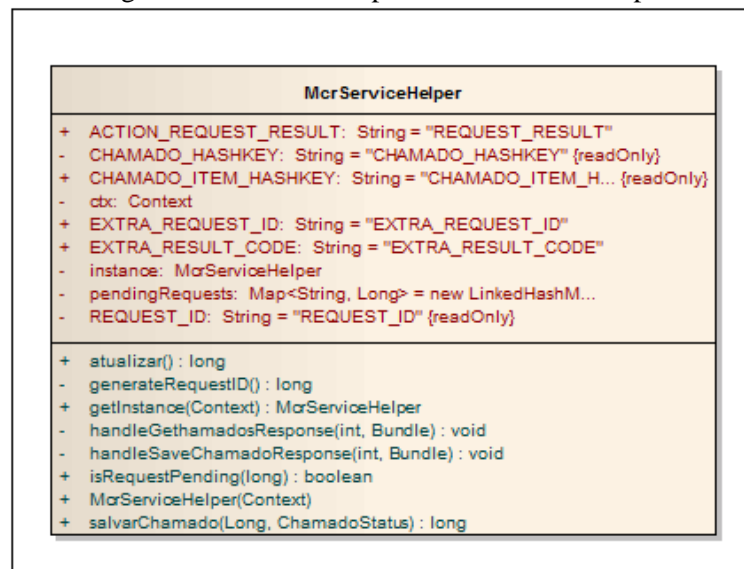
Figura 16 – Classes do pacote `br.com.mcr`



A classe `GCMIntentService` é responsável por registrar o dispositivo *smartphone* no GCM e por tratar as mensagens recebidas do GCM, assim como enviar um requisição ao servidor para registrar o aparelho.

3.2.2.1.5 Pacote `br.com.mcr.helper`

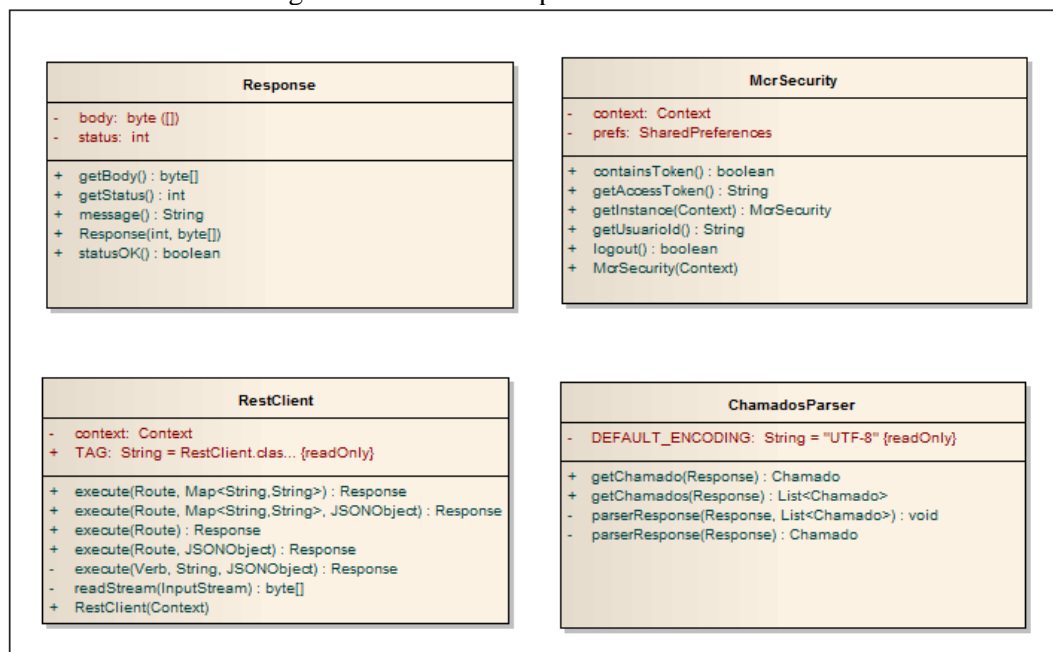
Figura 17 – Classes do pacote `br.com.mcr.helper`



A classe `McrServiceHelper` responsável por identificar requisições que já estão sendo realizadas para o recurso.

3.2.2.1.6 Pacote `br.com.mcr.rest`

Figura 18 – Classes do pacote `br.com.mcr.rest`

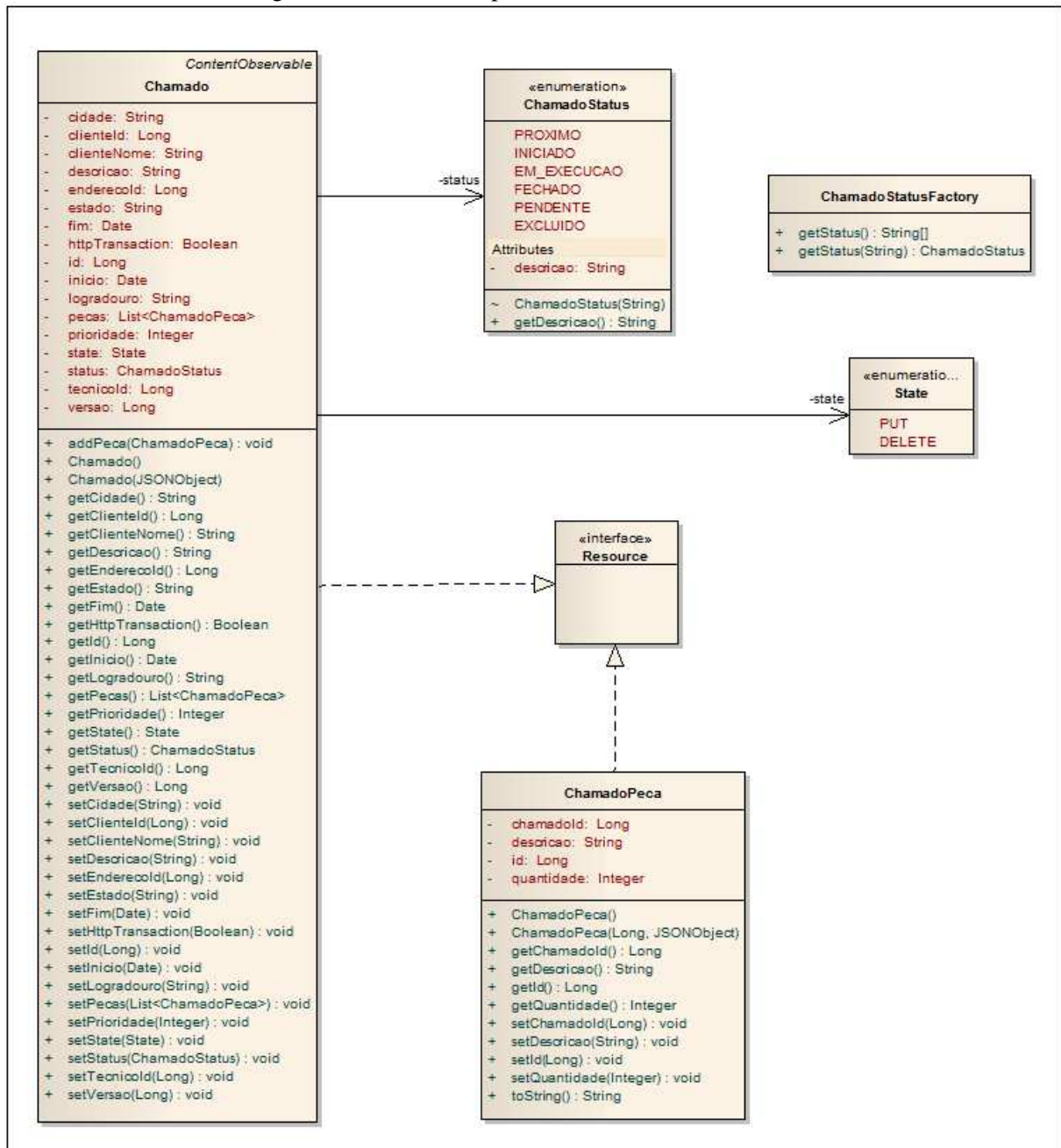


A classe `ChamadosParser` é responsável por converter um objeto JSON para Java. A classe `McrSecurity` é responsável por guardar as informações de *login* no dispositivo podendo realizar o *logout* da aplicação.

A classe `Response` é responsável por representar o response do HTTP, ou seja a resposta do cliente, já a classe `RestClient` é responsável por executar a requisição ao servidor.

3.2.2.1.7 Pacote `br.com.mcr.rest.resource`

Figura 19 – Classes do pacote `br.com.mcr.rest.resource`



A classe `Chamado` é responsável por identificar a classe de negócio `Chamado` ou seja representa nosso objeto no servidor.

A classe `ChamadoPeca` é responsável por identificar as peças que o chamado possui.

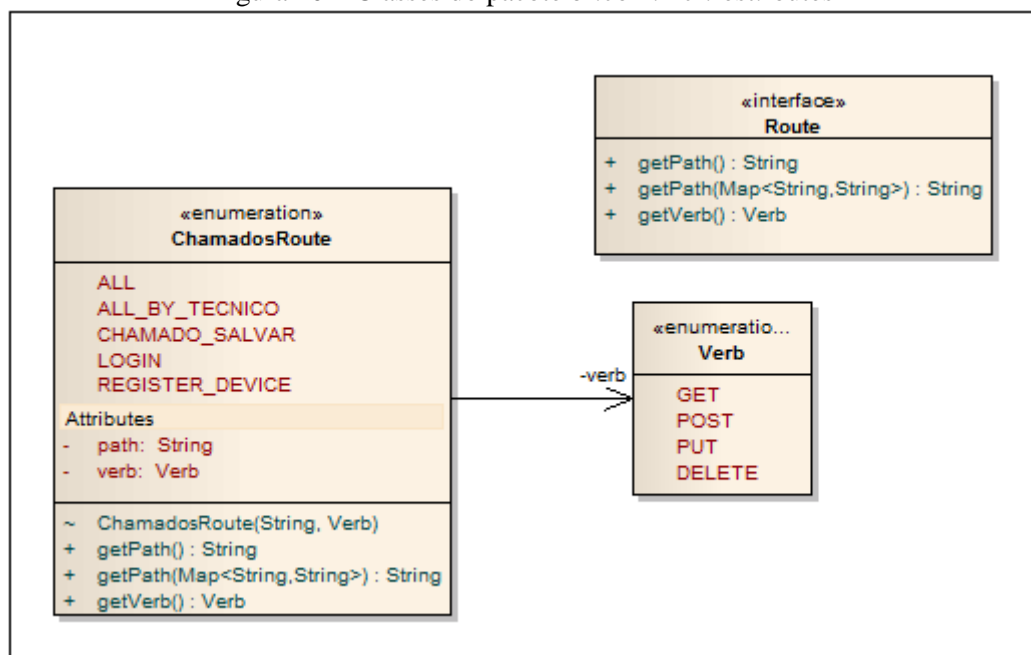
A classe `ChamadoStatus` é responsável por representa o status do chamado.

A classe `ChamadoStatusFactor` é um classe utilitária que irá instanciar um `ChamadoStatus`, já a classe `Resource` é um interface para identificar que as classes do Modelo são do tipo *Resource*.

A classe `State` é responsável por identificar o que foi realizado com Chamado no lado cliente, ou seja, se ele foi deletado ou feito um *update*. Caso não seja atualizado com sucesso no servidor, é possível identificar que tipo de solicitação foi feita com o chamado para posteriores tentativas de atualização no servidor.

3.2.2.1.8 Pacote `br.com.mcr.rest.routes`

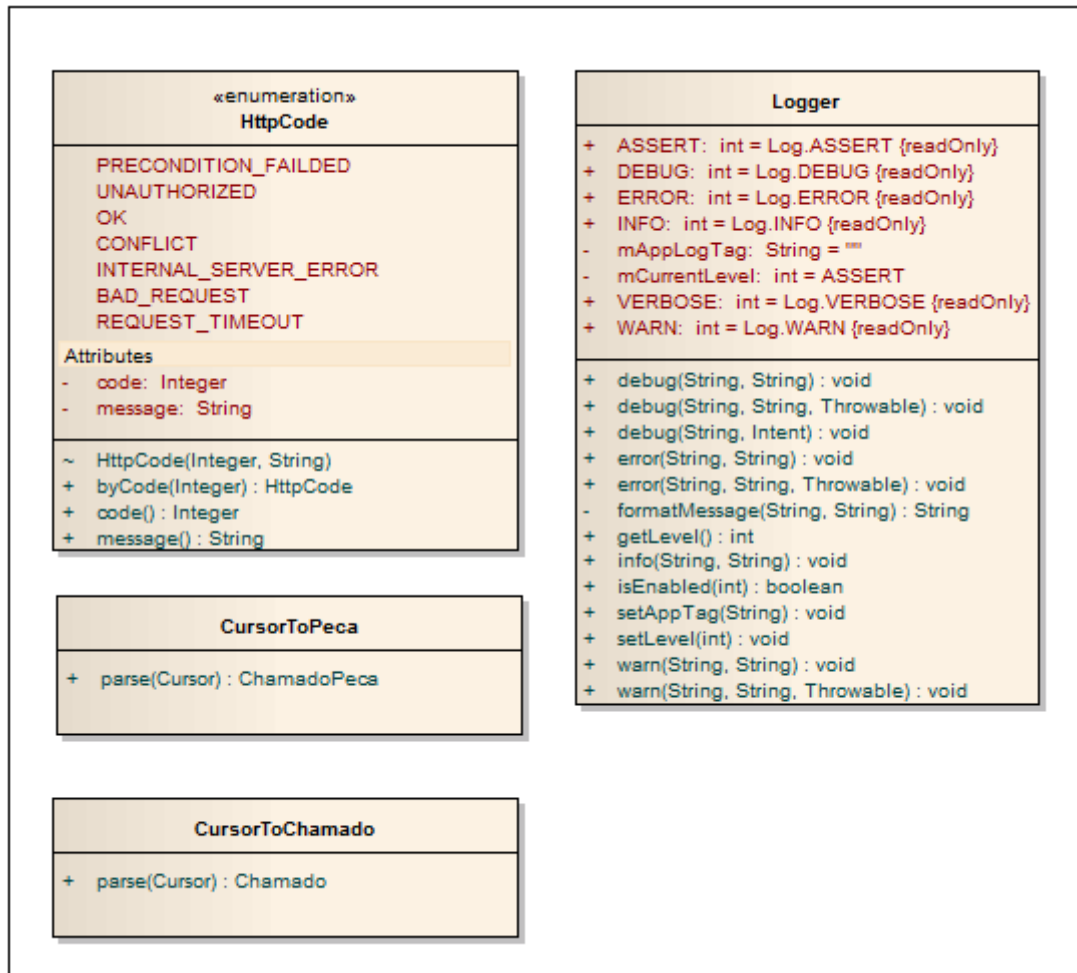
Figura 20 – Classes do pacote `br.com.mcr.rest.routes`



A classe `ChamadosRoute` é responsável por identificar as URL que existem no servidor.

A classe `Route` contrato implementado por `ChamadoRoute`.

A classe `Verb` é responsável por identificar os verbos do HTTP.

3.2.2.1.9 Pacote `br.com.mcr.util`Figura 21 – Classes do pacote `br.com.mcr.util`

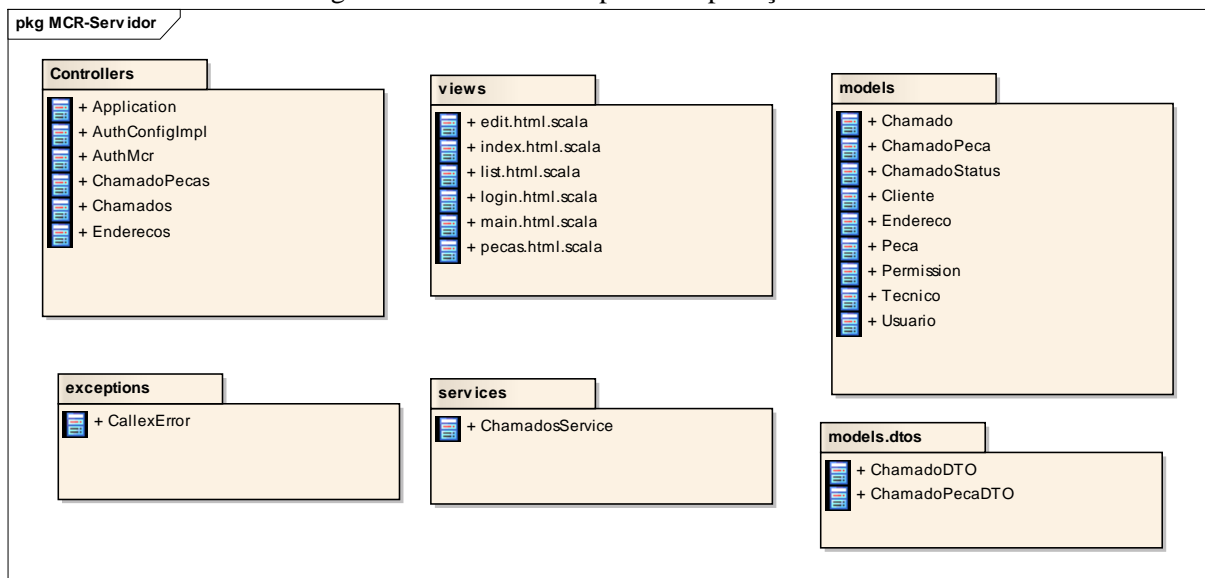
A Classe `HttpCode` é responsável por identificar os códigos HTTP.

A Classe `Logger` é um classe auxiliar para realizar logs na aplicação, `CursorToPeca` é uma classe auxiliar para fazer a conversão de um cursor do banco de dados para `chamadoPeca`, já `CursorToChamado` é uma classe auxiliar para fazer a conversão de um cursor do banco de dados para `chamado`

3.2.2.2 Aplicativo servidor

Nesta seção são descritas as classes necessárias para o desenvolvimento do aplicativo servidor. Para facilitar a visualização e que favoreça o melhor entendimento, a figura 22 exhibe os pacotes que compõem a aplicação servidor.

Figura 22 – Estrutura de pacotes aplicação servidor



Nas seções seguintes são explanados cada pacote, bem como as classes que os compõem com suas respectivas responsabilidades.

3.2.2.2.1 Pacote Controllers

O pacote `controllers` é composto por classes da camada de visão e é responsável pela ligação entre o pacote `model` e a camada de visualização, interagindo com a interface usuário.

A classe `AuthConfigImpl` é responsável por definir a segurança da aplicação. A classe `authMCR` está é responsável por logar e deslogar o usuário do sistema.

A classe `Chamados` é responsável pela tela de chamados. A classe `ChamadosPecas` é responsável da adição de peças aos chamados, e a classe `endereco` é responsável por fornecer o endereço por cliente.

3.2.2.2.2 Pacote `Models` e `Models.dto`

O pacote `Model` é responsável por armazenar e persistir os dados da aplicação.

3.2.2.2.3 Pacote `Services`

O pacote `services` é composto pelas classes responsável por executar os serviços auxiliares a aplicação cliente.

A classe `ChamadoService` é responsável por validar o que o usuário do dispositivo móvel envia e fornecer e retornar um objeto JSON com as informações requisitadas.

A classe `McrGCM` é utilizada para enviar ao GCM a mensagem que houve uma inserção de chamado. A classe `ServiceSecurity` é responsável pela autenticação e autorização do usuário, já a classe `TecnicoService` é responsável por registrar o *smartphone* do técnico.

3.2.2.2.4 Pacote `Exceptions`

O pacote `Exceptions` é responsável por simbolizar erros e a classe `MCRerror` é responsável por simbolizar erros de execução do servidor.

3.2.2.2.5 Pacote `Views`

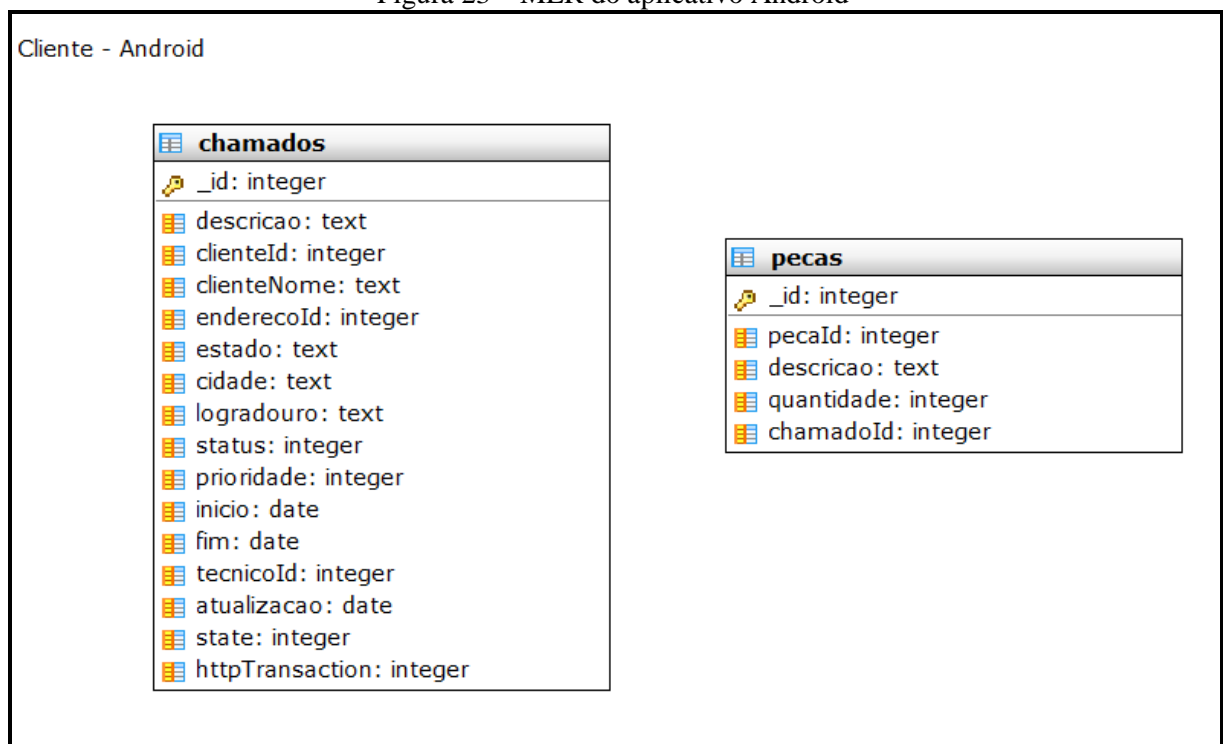
O pacote `views` contém os `htmls`.

3.2.3 Banco de dados

O MER tem como objetivo representar de forma visual as estruturas de dados de um banco de dados, o modelo é utilizado para representar a aplicação *smartphone* e servidor.

As informações do sistema foram persistidas no aplicativo cliente e no aplicativo servidor. O aplicativo cliente que está na plataforma Android utiliza biblioteca do framework Android para criar e gerenciar o banco de dados SQLite. A figura 23 apresenta o MER do banco de dados do aplicativo cliente.

Figura 23 – MER do aplicativo Android



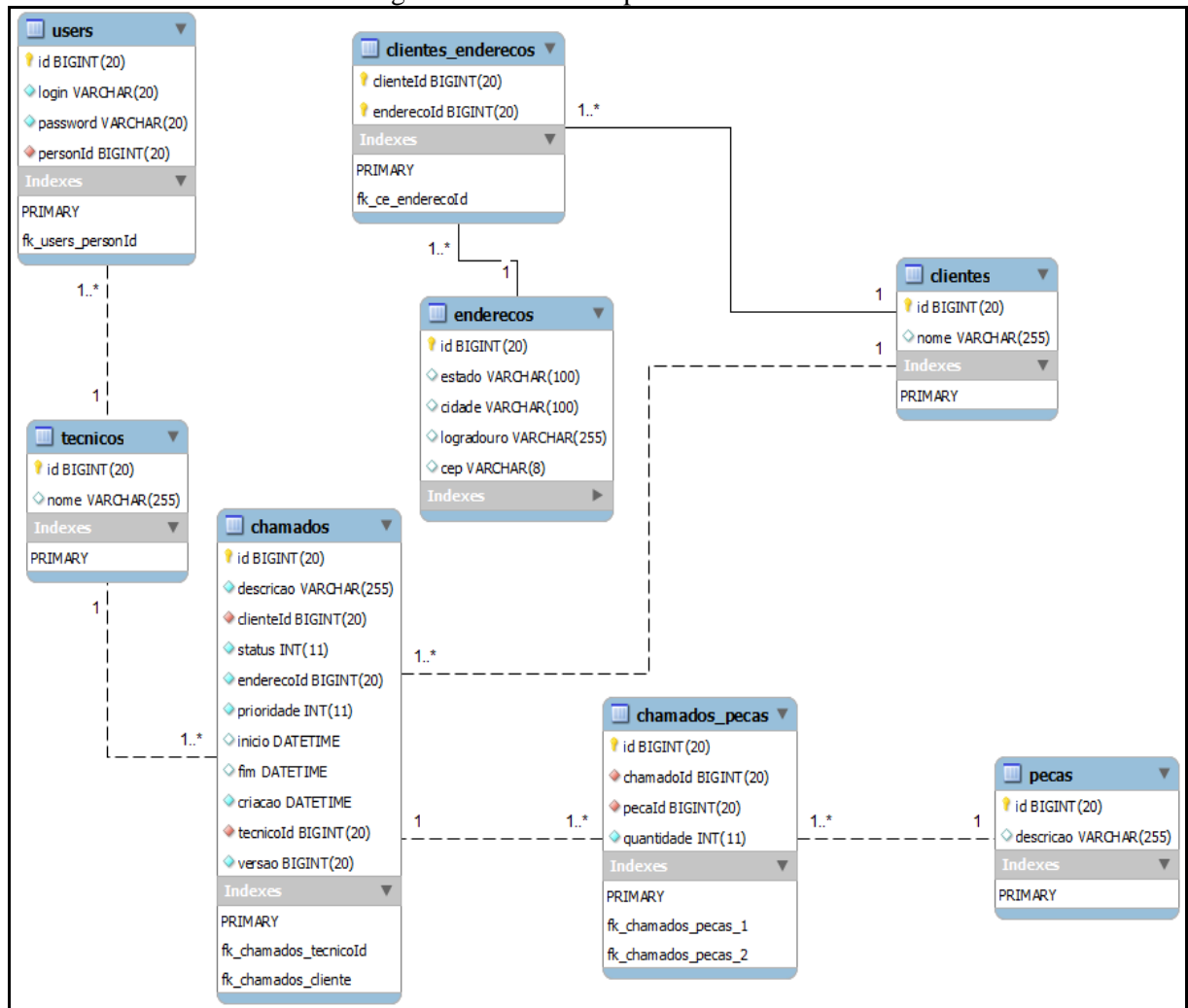
A tabela *chamados* contém as informações dos chamados. Esta tabela é atualizada toda vez que aplicativo sincroniza os dados com o servidor.

A tabela *peças* contém as informações das peças. Esta tabela é atualizada toda vez que aplicativo cliente sincroniza os dados com o servidor.

O aplicativo servidor utiliza banco de dados MySQL para persistir e recuperar informações relativas aos registros dos chamados e gerenciamento dos chamados pelo usuário do *smartphone*.

A figura 24 apresenta o MER do banco de dados do aplicativo servidor.

Figura 24 – MER do aplicativo servidor

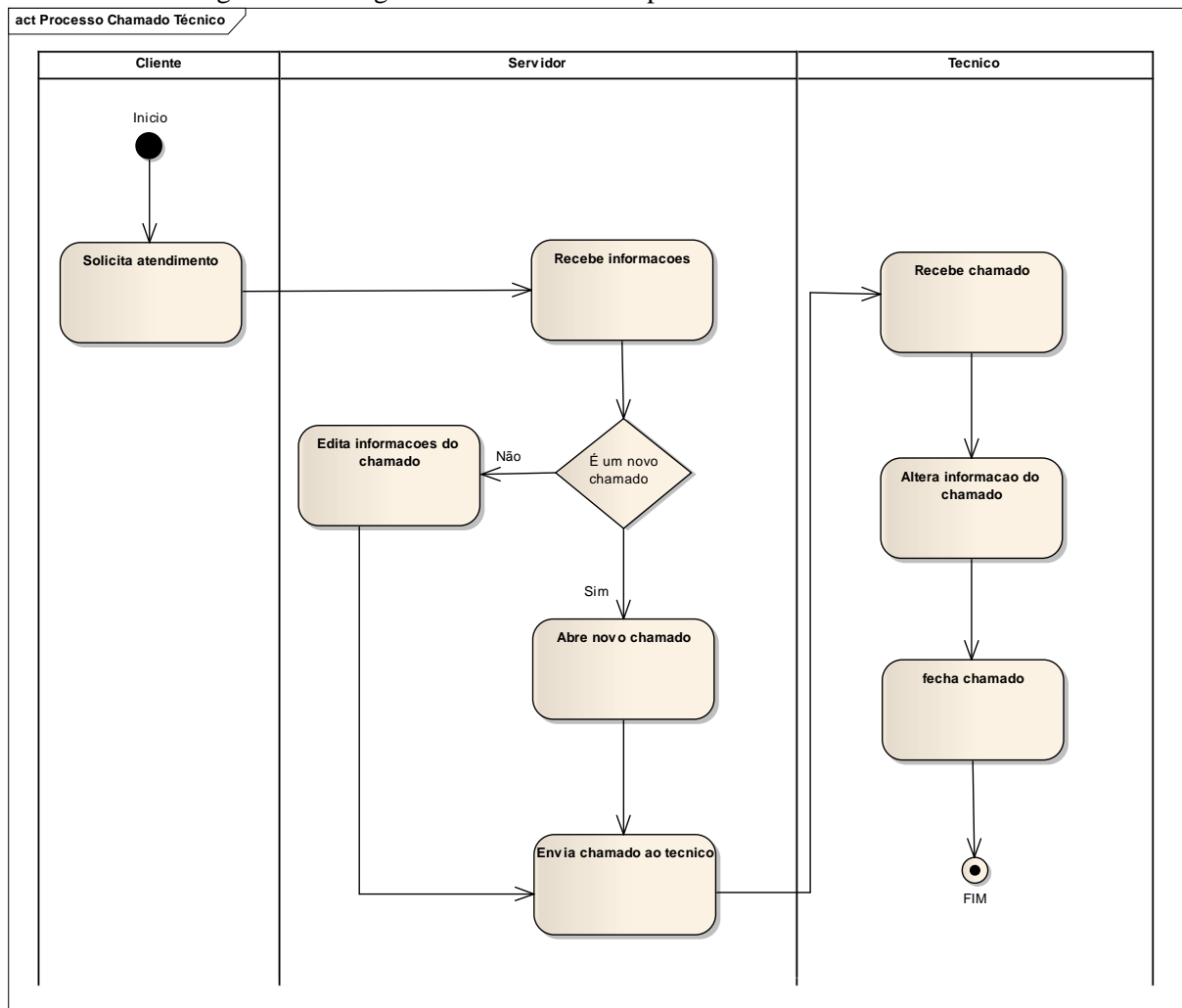


A tabela `users` é responsável por armazenar os usuários cadastrados para acessar o sistema. A tabela `tecnicos` armazena os técnicos cadastrados. A tabela `chamados` armazena as informações do chamados, tais como técnico responsável, endereço, status, prioridade, descrição. A tabela `clientes` é responsável por armazenar os clientes cadastrados. A tabela `endereco` é responsável por armazenar os endereços. A tabela `clientes_enderecos` é responsável por armazenar e vincular um endereço a um cliente. A tabela `pecas` é responsável por armazenar as peças cadastradas. Na tabela `chamados_pecas` são armazenada as peças vinculadas a um chamado.

3.2.4 Diagramas de atividade

O diagrama de atividades foi utilizado para facilitar o entendimento de determinada situações do sistema. A figura 25 apresenta as atividades do processo Chamado Técnico.

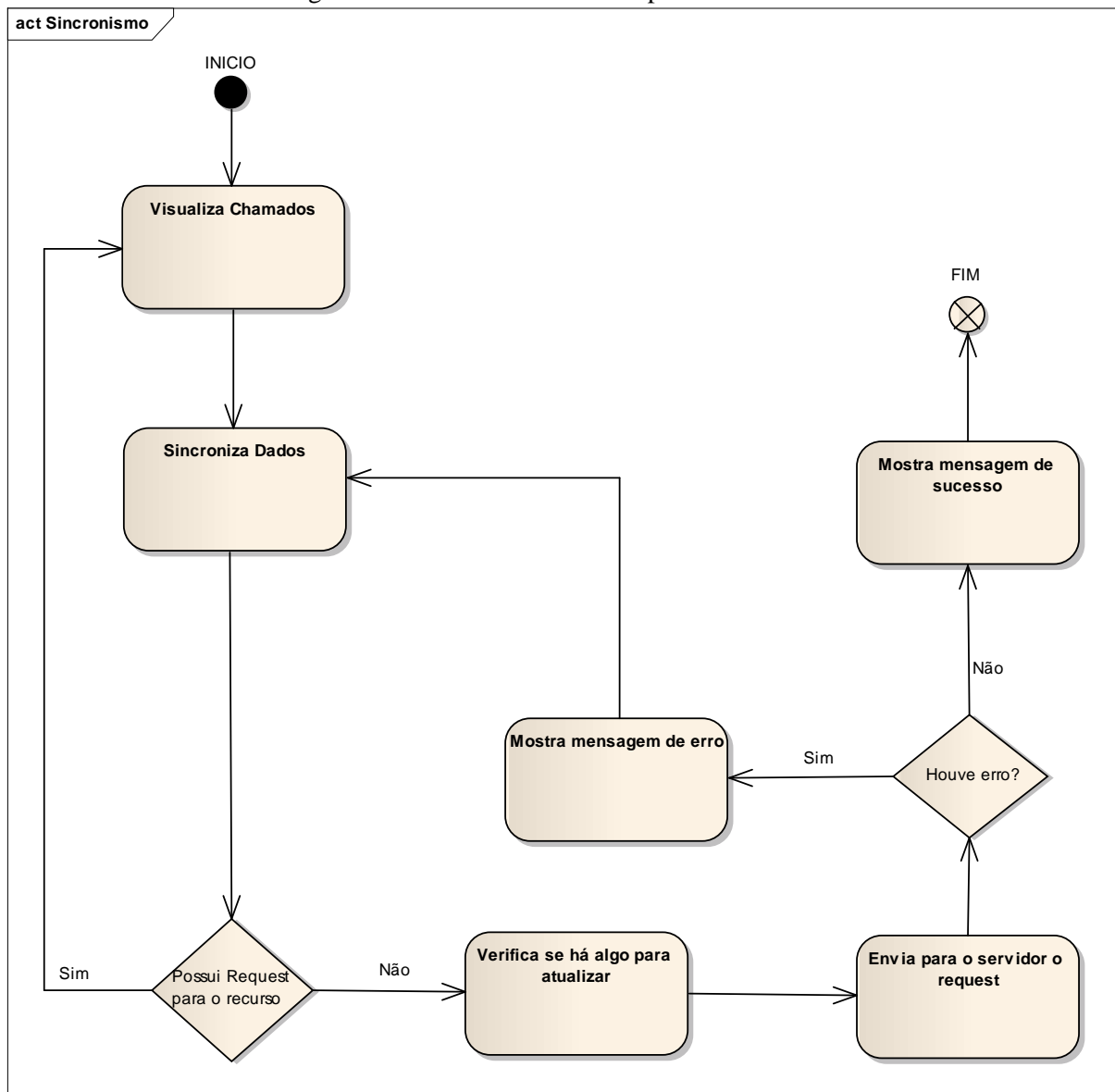
Figura 25 – Diagrama de atividades do processo de Chamado Técnico



No diagrama apresentado, o cliente solicita atendimento. O usuário do servidor recebe as informações do cliente e o verifica se é um chamado novo ou alguma informação do cliente relacionado ao chamado em aberto. O usuário preenche os campos grava o registro no banco de dados e envia o chamado para o técnico. O técnico recebe o chamado no *smartphone*, visualiza o chamado, altera a informações de um chamado e fecha o chamado.

O diagrama representado na figura 26 apresenta o processo de sincronismo do dispositivo móvel com o servidor.

Figura 26 – Sincronismo com o aplicativo servidor



O técnico visualiza a lista de chamados e solicita para sincronizar. Nesse processo, se o técnico clicar para sincronizar mais de uma vez, o sistema verifica se a requisição de sincronismo não é do mesmo recurso. Se for então volta para lista de chamados, senão o sistema verifica o que tem para atualizar e envia um *request* para o servidor. Se a atualização for efetiva então exibe uma mensagem de sucesso de que o sincronismo ocorreu. Caso contrário, é gerada uma mensagem de erro e volta a lista de chamados.

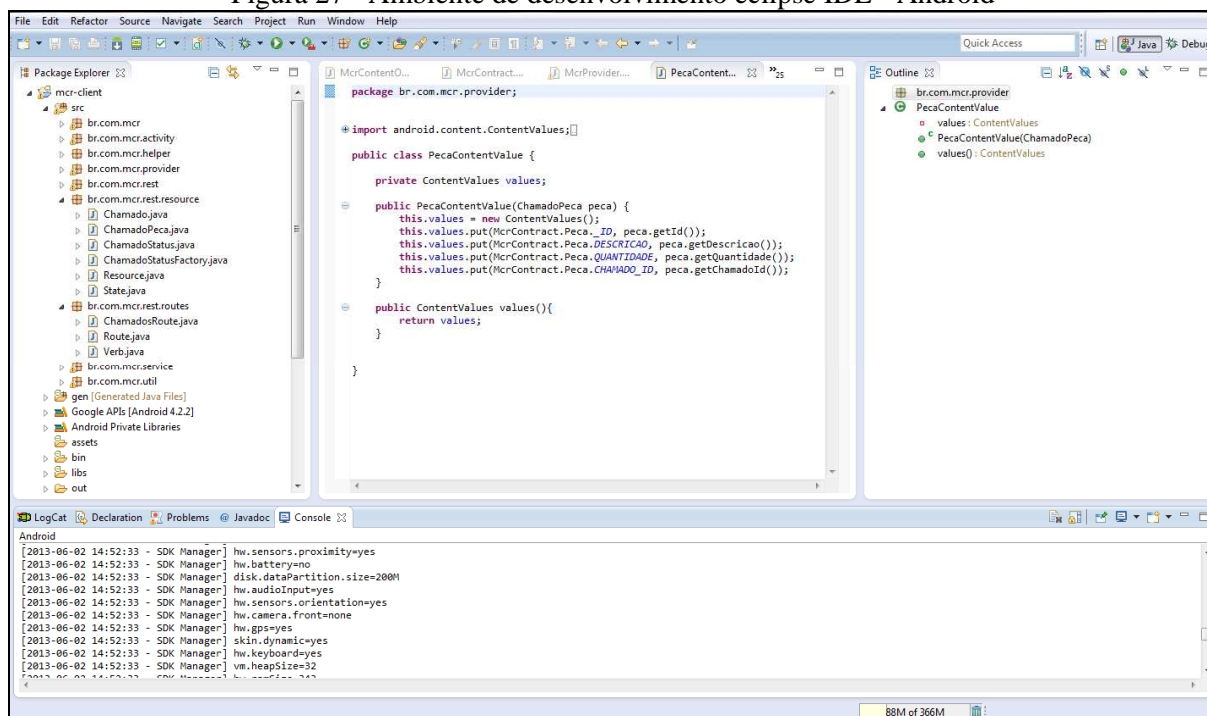
3.3 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas na implementação, bem como trechos de códigos relevantes implementados durante o desenvolvimento do trabalho. Por fim, é apresentada a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento do sistema proposto foi efetuado em duas partes. Para o dispositivo móvel, foi utilizada a linguagem de programação Java com a API de desenvolvimento Android versão 4.0.3, também conhecida como *Ice Cream Sandwich*. O ambiente de desenvolvimento utilizado foi o Eclipse IDE juntamente com o conjunto de *plugins* do *Android Development Tools* (ADT) disponibilizado pela Google, que provém os recursos necessários para criação, execução e depuração de aplicações Android de forma a facilitar o desenvolvimento (figura 27).

Figura 27 - Ambiente de desenvolvimento eclipse IDE - Android



Para a execução e a depuração da aplicação, em um primeiro momento foi utilizado o emulador Android disponibilizado pelo *Software Development Kit* (SDK), e num segundo momento um *smartphone* modelo GT-N7000 da fabricante Samsung chamado de Galaxy Note.

O dispositivo Galaxy Note possui processador dual core 1.4 Giga Hertz (GHz), com memória de 1024 Mega Bytes (MB) de *Random Access Memory* (RAM), 16 Giga Bytes (GB) de memória interna e resolução de 1280x800.

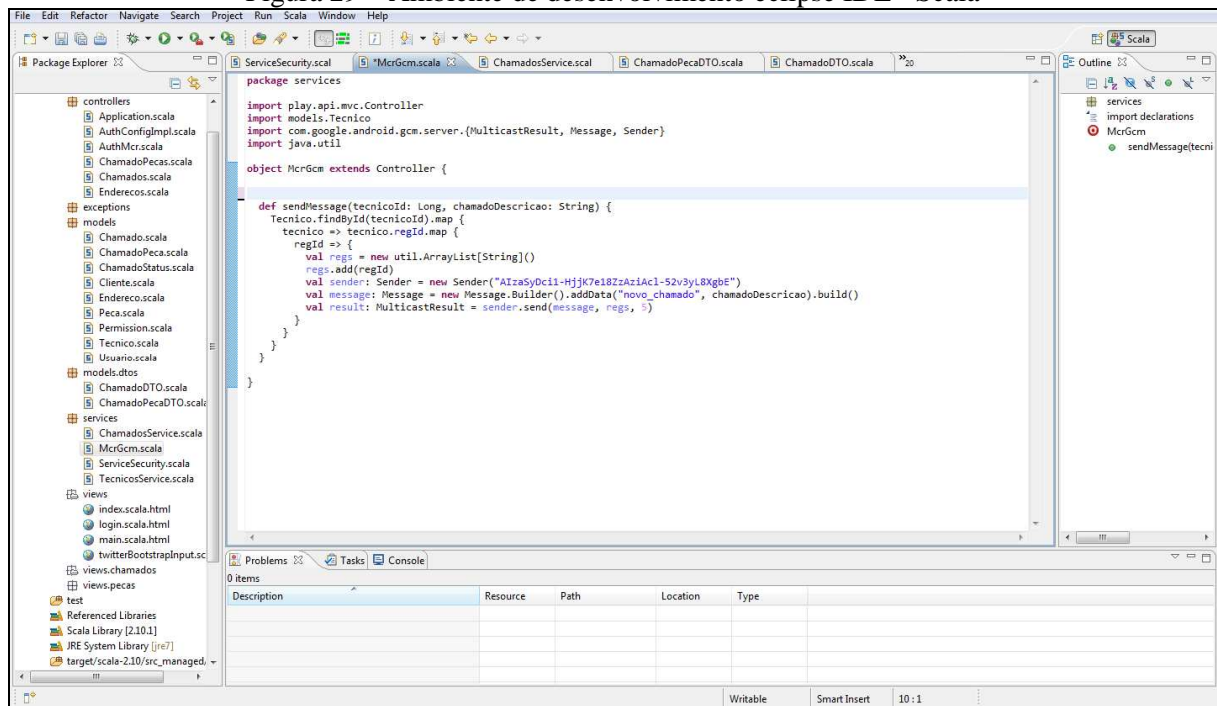
Figura 28 – Smartphone Samsung Galaxy Note



A aplicação web (servidor), foi implementada sob o paradigma da orientação a objetos, utilizando a linguagem de programação Scala com API de desenvolvimento 2.10.1 (figura 29), juntamente com o *framework* Play que facilitou o desenvolvimento da aplicação *web*. O ambiente de desenvolvimento utilizado foi o Eclipse IDE juntamente com o conjunto de *plugins* necessários com os recursos necessários para criação, execução e depuração.

Como banco de dados foi utilizado o MySQL.

Figura 29 – Ambiente de desenvolvimento eclipse IDE - Scala



3.3.1.1 Sincronismo

Uma das principais funcionalidades da aplicação é a possibilidade de sincronizar os dados do dispositivo móvel com os da aplicação servidor.

A seguir são descritas as etapas envolvidas no processo de sincronização, quando o usuário opta por sincronizar a lista de chamados. O método `atualizar()` da classe `ChamadosListActivity`, é responsável por solicitar a sincronização da lista de chamados. Esse método, invoca o método `atualizar()` da classe `McrServiceHelper` que verifica se já não está executando o sincronismo e retorna um id para a requisição (Quadro 11).

Quadro 11 – método `atualizar()` da classe `ChamadosListActivity`

	<pre> private void atualizar() { if (requestId == null) { requestId = mcrServiceHelper.atualizar(); } else if (mcrServiceHelper.isRequestPending(requestId)) { Logger.info(TAG, "Requisicao sendo processada."); } else { requestId = null; Logger.info(TAG, "Nossa requisicao ja foi processada."); } } </pre>
--	---

A Classe `McrServiceHelper` ao ser invocada, o primeiro passo é verificar se já existe uma chamada para este recurso. Se existir é retornado o id do recurso para classe a `ChamadoListAcitivity`. Caso não esteja executando a sincronização de dados é gerado um id para o recurso e é iniciado um serviço em *background* pela classe `McrService` passando como parâmetros as intenções, identificando o tipo de recurso que se deseja obter assim como um callback (`ResultReceicer` que é uma classe do Android) que irá tratar a resposta do que vier no `McrService`. No quadro 12 pode ser visto o código.

Quadro 12 – método `atualizar()` da classe `McrServiceHelper`

	<pre> public long atualizar(){ if (pendingRequests.containsKey(CHAMADO_HASHKEY)) { return pendingRequests.get(CHAMADO_HASHKEY); } long requestId = generateRequestID(); pendingRequests.put(CHAMADO_HASHKEY, requestId); ResultReceiver serviceCallback = new ResultReceiver(null) { protected void onReceiveResult(int resultCode, Bundle resultData) { handleGethamadosResponse(resultCode, resultData); } }; Intent intent = new Intent(this.ctx, McrService.class); intent.putExtra(McrService.METHOD_EXTRA, McrService.METHOD_GET); intent.putExtra(McrService.RESOURCE_TYPE_EXTRA, McrService.RESOURCE_TYPE_CHAMADO); intent.putExtra(McrService.SERVICE_CALLBACK, serviceCallback); intent.putExtra(REQUEST_ID, requestId); this.ctx.startService(intent); return requestId; } </pre>
--	---

O método `onHandleIntent()` da classe `McrService`, é responsável por identificar a requisição de intenção que foi passado pelo `McrServiceHelper`, identificado é invocado o método `atualizar()` passando o contexto da aplicação e também um callback que irá tratar a resposta do método `atualizar()` da classe `ChamadoProcessor`.

Quadro 13 – método `onHandleIntent()` da classe `Service`

```

protected void onHandleIntent(Intent requestIntent) {

    mOriginalRequestIntent = requestIntent;

    String method =
requestIntent.getStringExtra(McrService.METHOD_EXTRA);

    int resourceType =
requestIntent.getIntExtra(McrService.RESOURCE_TYPE_EXTRA, -1);

    mCallback =
requestIntent.getParcelableExtra(McrService.SERVICE_CALLBACK);

    switch (resourceType) {
        case RESOURCE_TYPE_CHAMADO: {
            ChamadoProcessor processor = new
ChamadoProcessor(getApplicationContext());
            processor.atualizar(makeChamadoProcessorCallback());
            break;
        }
        case RESOURCE_TYPE_CHAMADO_ITEM: {
            Long chamadoId =
requestIntent.getLongExtra(McrService.CHAMADO_ID, -1);
            Integer status =
requestIntent.getIntExtra(McrService.CHAMADO_STATUS, -1);
            ChamadoProcessor processor = new
ChamadoProcessor(getApplicationContext());
            processor.saveChamado(makeChamadoProcessorCallback(),
chamadoId, status);
            break;
        }
    }
}
}

```

O método `atualizar()` da classe `ChamadoProcessor` irá criar um `RestClient` que é um cliente `Rest` que executa as requisições que foram requisitadas. Antes de executar qualquer requisição é verificado se existe algum chamado no banco de dados que não foi atualizado. Se existir algum, é pego um por um e feito uma requisição ao servidor, realizando a atualização dos chamados após a atualização de cada um, é feito uma nova requisição identificando chamados novos que precisam ser sincronizados (Quadro 14).

Quadro 14 – método atualizar() da classe ChamadoProcessor

```

public void atualizar(ChamadoProcessorCallback callback) {

    boolean timeOut = false;

    RestClient restClient = new RestClient(mContext);

    Chamado chamado = chamadosPrecisamSerAtualizados();

    while (chamado != null) {

        updateTransacao(chamado.getId(), chamado.getStatus(),
chamado.getState(), true);

        try {
            JSONObject json = new JSONObject();
            json.put("chamadoId", chamado.getId());
            json.put("status", chamado.getStatus().ordinal());
            json.put("versao", chamado.getVersao());
            Response response =
restClient.execute(ChamadosRoute.CHAMADO_SALVAR, json);

            if (response.getStatus() == HttpStatusCode.OK.code()) {
                ChamadosParser parser = new ChamadosParser();
                updateChamado(parser.getChamado(response));
            }

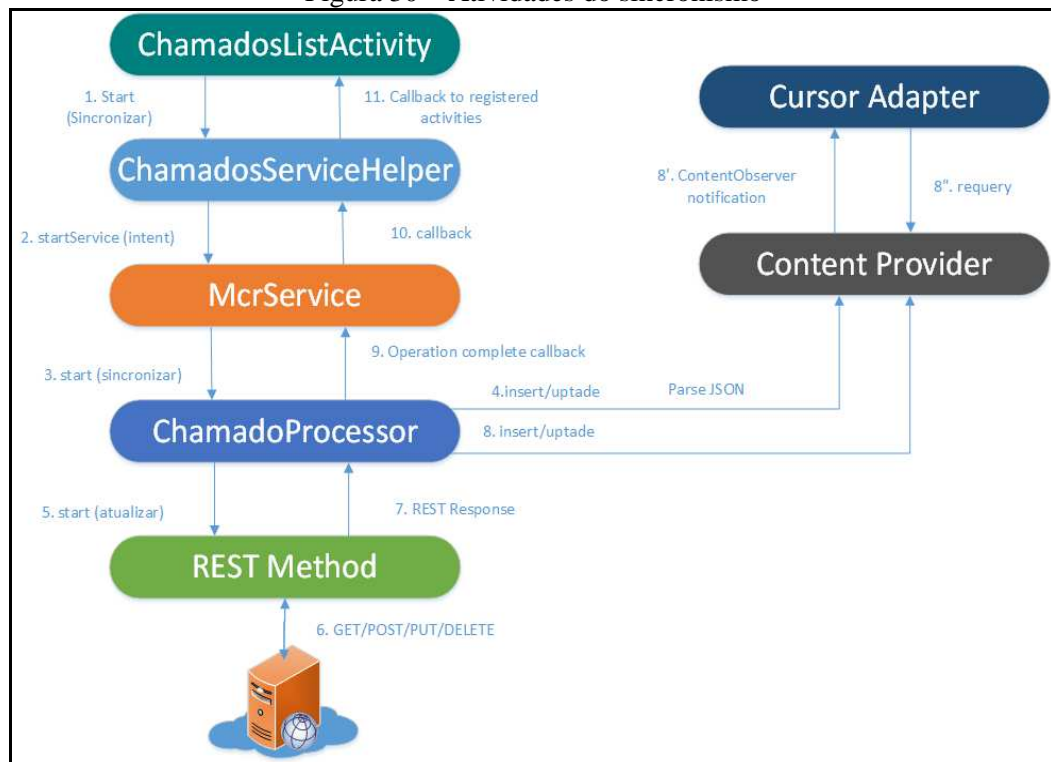
mContext.getContentResolver().notifyChange(McrContract.Chamado.CONTENT_URI,
null);
        }
    }
}

```

Após concluir todo o processo no chamadoProcessor, é utilizado o callback para enviar uma resposta ao McrServiceHelper, avisando que terminou o trabalho. Já o McrServiceHelper analisa a resposta, retirar o id da fila de requisições, e avisa a todos que estão escutando o evento de atualização, que uma atualização foi realizada.

A figura 30 exibe o diagrama de atividades que representa todas as etapas do processo de sincronismo.

Figura 30 – Atividades do sincronismo



3.3.2 Operacionalidade da implementação

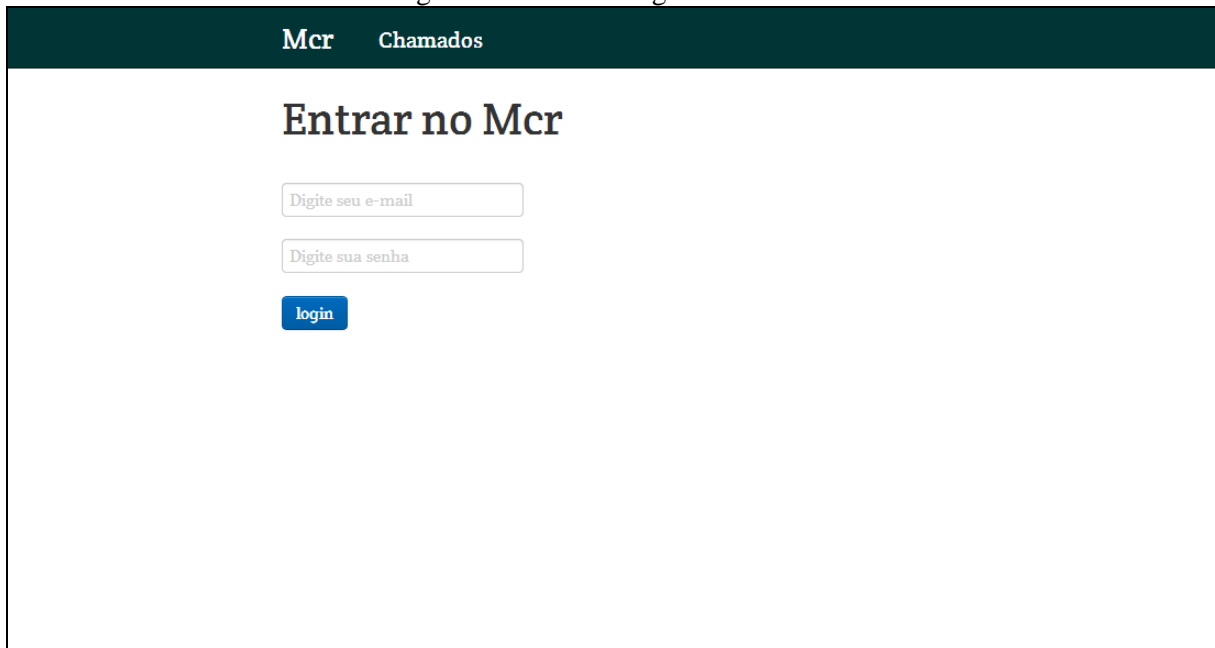
Nesta seção é apresentado a operacionalidade da aplicação em função de casos de uso da aplicação cliente Android e aplicação servidor, utilizando de imagens para facilitar o entendimento de cada uma das funcionalidades disponibilizadas.

3.3.2.1 Aplicação servidor

3.3.2.1.1 Efetuar *login*

A tela inicial do aplicativo serve para realizar o *login* do usuário, conforme demonstrado na figura 31.

Figura 31 – Tela de login do servidor



A interface de login do servidor apresenta uma barra superior escura com os links "Mcr" e "Chamados". O título principal da seção é "Entrar no Mcr". Abaixo dele, há dois campos de entrada: "Digite seu e-mail" e "Digite sua senha". Um botão azul com o texto "login" está posicionado abaixo dos campos.

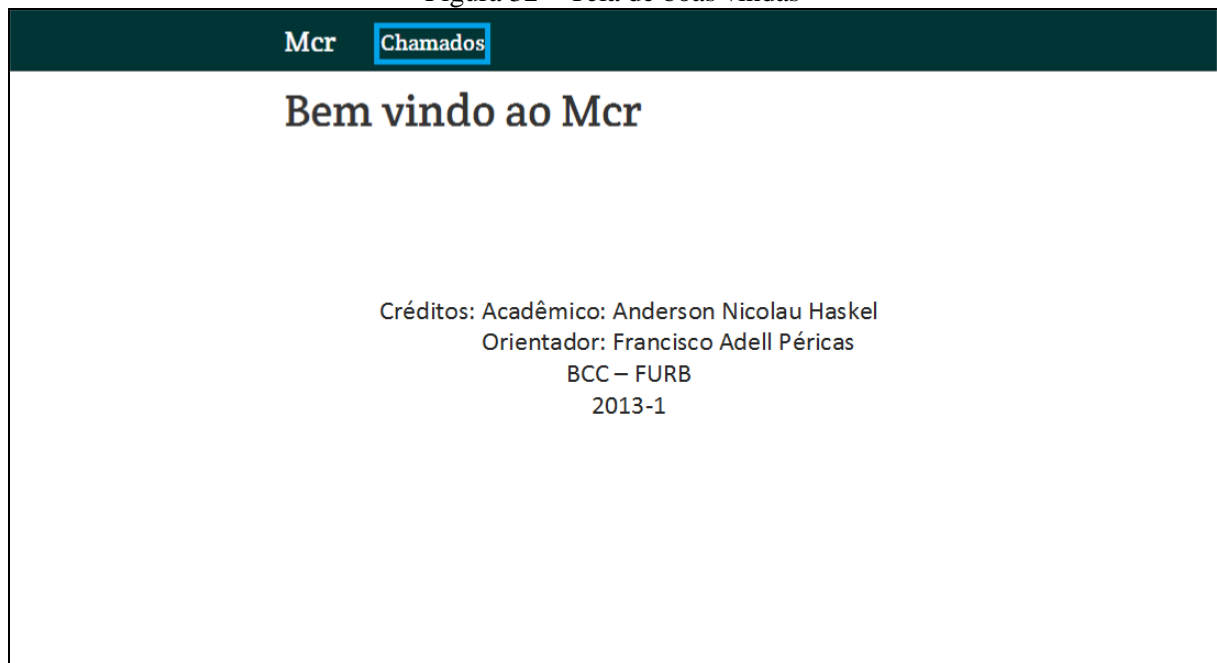
Para efetuar o *login*, o usuário deve seguir os seguintes passos:

- a) preencher o campo usuário;
- b) preencher o campo senha;
- c) clicar no botão *login*.

3.3.2.1.2 Listar chamados

Esta tela é exibida logo após a execução do *login* do usuário. É exibida uma mensagem de boas-vindas, e um botão para listar os chamados, conforme demonstra a figura 32.

Figura 32 – Tela de boas vindas



Após o usuário clicar na opção Chamados, é exibido a lista com todos os chamados (figura 33), bem como as demais funções disponíveis na tela.

Figura 33 – Tela de lista de chamados

Mcr Chamados										
Chamados										Novo Chamado
Descrição	Cliente	Status	Endereco	Prioridade	Início	Fim	Peças	Editar	Excluir	
MONITOR NAO LIGA	FURB	Próximo	RUA: ANTONIO DA VEIGA, 140	1			Listar	Editar	Excluir	
TECLADO COM DEFEITO	MCRTEC	Próximo	RUA: DOIS DE SETEMBRO	1			Listar	Editar	Excluir	
CPU NAO LIGA	MOBILE TEC	Próximo	RUA: ALMIRANTE BARROSO	3			Listar	Editar	Excluir	
SISTEMA OPERACIONAL NAO CARREGA	MCR	Próximo	RUA: SETE DE SETEMBRO	5			Listar	Editar	Excluir	

3.3.2.1.3 Incluir chamado

Esta funcionalidade permite ao usuário criar um novo chamado. Para tanto deve-se clicar no botão Novo Chamado conforme visto na figura 33.

Figura 34 – Incluir chamado

Mcr Chamados

Descrição *

Cliente *

Status *

Endereço *

Prioridade *

Início

Fim

Técnico *

Salvar

Após clicar no botão novo chamado, o usuário é remetido à outra tela, onde deverão ser preenchidos os campos relacionados ao cadastro de um novo chamado, como demonstrado na figura 34. Após o preenchimento destes campos o usuário deve clicar o botão salvar.

Para incluir um novo chamado, o usuário deverá preencher no mínimo os seguintes campos obrigatórios, identificados pelo sinal gráfico asterisco (*):

- descrição: neste campo o usuário deverá fazer uma breve descrição de sua problemática (chamado);
- cliente: neste campo o usuário seleciona o cliente;
- status: neste campo o usuário seleciona o status do chamado;
- endereço: neste campo o usuário seleciona o endereço;
- prioridade: neste campo o usuário define a prioridade;
- início: neste campo o usuário preenche a data de abertura do chamado;
- fim: este campo deverá ser preenchido após a finalização do chamado;
- técnico: neste campo o usuário seleciona o técnico para o atendimento do chamado.

3.3.2.1.4 Listar peças

Esta funcionalidade permite ao usuário listar e/ou incluir uma peça ao chamado. Desta forma deve-se clicar na opção listar conforme visto na figura 33.

Figura 35 – Listagem e inclusão de peças

The screenshot shows a web interface titled "Mcr Chamados". It features a form with a "Peça" dropdown menu (currently showing "Selecione"), a "Quantidade *" input field, and an "Inserir" button. Below the form is a table with the following data:

Descrição	Quantidade	Excluir
MONITOR LCD	1	Excluir

3.3.2.1.5 Editar chamado

Esta funcionalidade permite ao usuário em qualquer momento editar todas as informações referentes ao chamado uma peça ao chamado. Conforme representado na figura 35.

Figura 36 – Editar um chamado

The image shows a web form titled "Mcr Chamados" (Calls) for editing a call ticket. The form is enclosed in a dark green header bar. The fields are as follows:

- Descrição ***: A text input field containing "MONITOR NAO LIGA".
- Cliente ***: A dropdown menu with "FURB" selected.
- Status ***: A dropdown menu with "Próximo" selected.
- Endereco ***: A dropdown menu with "RUA: ANTONIO DA VEIG" selected.
- Prioridade ***: A dropdown menu with "1" selected.
- Início**: An empty text input field.
- Fim**: An empty text input field.
- Técnico ***: A dropdown menu with "Anderson" selected.

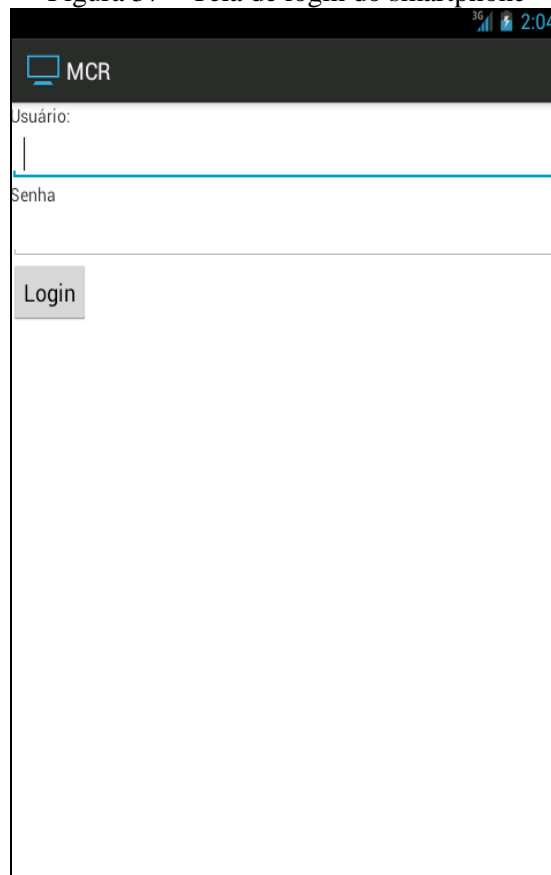
At the bottom of the form is a blue button labeled "Salvar" (Save).

3.3.2.2 Aplicação smartphone

3.3.2.2.1 Efetuar *login*

Esta funcionalidade permite ao usuário em seu primeiro acesso ao sistema, efetuar o *login*.

Figura 37 – Tela de login do smartphone

A screenshot of a smartphone screen displaying a login interface. At the top, there is a status bar with a 3G signal icon, a battery icon, and the time 2:04. Below the status bar is a dark header bar with a blue icon of a computer monitor and the text 'MCR'. The main area of the screen contains two input fields: the first is labeled 'Usuário:' and the second is labeled 'Senha'. Below these fields is a grey button with the text 'Login'.

Para efetuar o *login*, o usuário deve seguir os seguintes passos:

- a) preencher o campo usuário com seu respectivo e-mail, sendo este utilizado no servidor;
- b) preencher o campo senha;
- c) clicar no botão *login*.

3.3.2.2.2 Consultar chamados

Esta tela é exibida logo após a execução do *login* do usuário, e permite visualizar a lista de chamados. Nesta opção, o usuário poderá escolher o chamado e visualizar suas respectivas informações.

Figura 38 – Lista de chamados do smartphone



3.3.2.2.3 Excluir chamados

Esta funcionalidade permite ao usuário excluir qualquer chamado desejado da lista (Figura 39).

Figura 39 – Descrição do chamado do smartphone

The screenshot shows a mobile application interface for a call record. At the top, there is a status bar with signal strength, battery level, and the time 2:06. Below this is a header bar with the text 'MCR' and two buttons: 'SALVAR' (Save) and 'REMOVER' (Remove). The main content area is a form with the following fields:

- Descricao: MONITOR NAO LIGA
- Cliente: FURB
- Estado: SC
- Cidade: BLUMENAU
- Logradouro: RUA: ANTONIO DA VEIGA, 140
- Status: Iniciado (with a dropdown arrow)
- Prioridade: 1
- Peças: MONITOR LCD - 1

Para excluir um chamado deve seguir os seguintes passos:

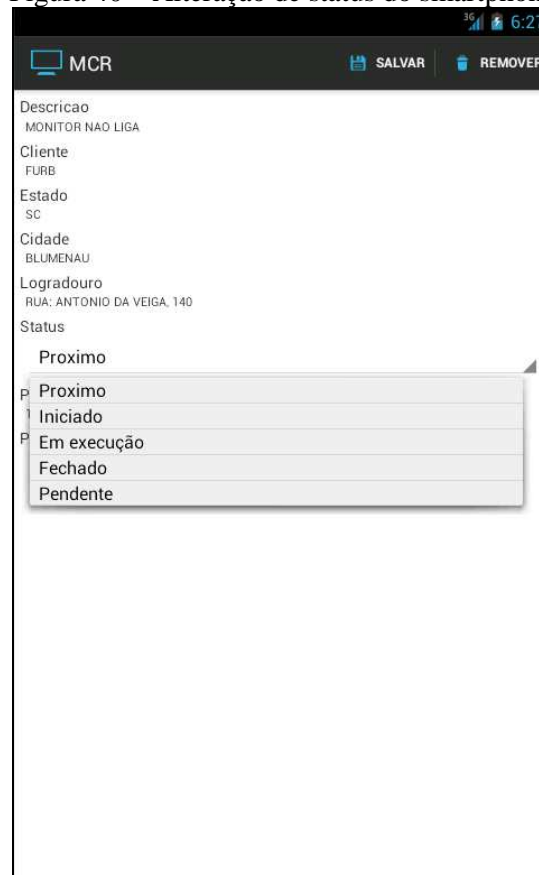
- primeiramente o usuário deverá clicar no chamado desejado;
- após a seleção do chamado, o usuário deve clicar no botão remover.

Se a exclusão do chamado for concluída com sucesso, o usuário será remetido a tela inicial.

3.3.2.2.4 Alterar status

Esta funcionalidade permite ao usuário a atualização do *status* do chamado. Tendo como opções: *Próximo* (chamado disponível para atendimento), *Iniciado* (técnico está em deslocamento para atendimento do chamado), *Em execução* (técnico está em atendimento), *Fechado* (chamado finalizado), *Pendente* (chamado requer uma solicitação de peça ou outro equipamento).

Figura 40 – Alteração de status do smartphone



Para alterar o status do chamado o usuário deve seguir os seguintes passos:

- a) o usuário deve clicar no campos *status* e escolher a opção desejada;
- b) após a seleção do *status*, o usuário deve clicar no botão salvar.

Se a atualização do *status* do chamado for concluída com sucesso, é exibido ao usuário uma notificação informando ao mesmo que a operação ocorreu com sucesso. Caso contrário é exibido ao usuário uma notificação de erro e o chamado ficará em vermelho indicando que o chamado não foi sincronizado com o servidor, conforme representado na figura 41.

Figura 41 – Alteração de Chamado não sincronizado do smartphone



3.3.2.2.5 Sincronizar

Esta funcionalidade permite ao usuário sincronizar os dados da lista de chamados com o servidor. Para ocorrer a sincronização de dados, o usuário deverá clicar no botão sincronizar como representado na figura 42.

Figura 42 – Opção de sincronismo do smartphone



Se o sincronismo for efetuado com sucesso, é exibido ao usuário uma notificação de sucesso.

3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o desenvolvimento de um software para gerenciamento e controle de chamados técnicos em dispositivos móveis baseados em Android, permitindo através do dispositivo, sincronizar a base de dados com o servidor, receber chamados, consultar, alterar status e trabalhar *offline*. Também foi desenvolvido um software em ambiente web para ser acessado através de um desktop, para gerenciamento e controle de chamados técnicos.

Foram encontradas algumas dificuldades durante a implementação do aplicativo Android, dentre as quais foi o pouco conhecimento da plataforma Android, que demandaram muita pesquisa, e através de pesquisas de exemplos de codificação de rotinas não inerentes ao contexto do trabalho é que foi possível encontrar as classes necessárias para a implementação das funcionalidades propostas. A parte mais complexa foi a parte da sincronização, que demandou muito estudo e pesquisa para encontrar uma maneira de sincronizar os dados, sem perder a consistência dos mesmos e conseguir deixar a aplicação cliente saber o que estava acontecendo. Vários erros foram encontrados durante todo o processo de desenvolvimento, sendo os mais comuns relacionados aos dados que ficavam inconsistentes. Não houveram dificuldades relacionadas ao desenvolvimento da aplicação do lado servidor.

Quanto aos trabalhos correlatos, é demonstrado no quadro 15 uma comparação entre os trabalhos correlatos e o protótipo desenvolvido neste trabalho. Para fazer esta comparação foram utilizados as funcionalidades que o protótipo construído possui.

Quadro 15 – Comparação com trabalhos correlatos

Funcionalidades	Este trabalho	Web Assist	Qtux
aplicativo <i>smartphone</i>	sim	não	não
aplicação <i>web</i>	sim	sim	sim
possibilidade sincronizar	sim	não	não
consultar lista de chamados	sim	sim	sim
consultar lista de pecas	sim	sim	sim
atualizar chamados (<i>status</i>)	sim	sim	sim

4 CONCLUSÕES

A concepção de um protótipo de software para controle de chamados técnicos em dispositivos móveis consolida a união de duas necessidades nas empresas de suporte técnico em informática, vinculando assim informações de grande relevância, com controle e agilidade das mesmas.

A partir do desenvolvimento do projeto constatou-se que as SDK de desenvolvimento do Android e do Scala, em conjunto com o ambiente de desenvolvimento Eclipse IDE, possibilitaram de forma adequada o desenvolvimento dos requisitos propostos neste trabalho. O emulador Android e o *smartphone* foram necessários para desenvolver e realizar os testes. Quanto ao servidor web a utilização do *framework* Play permitiu que as funcionalidades fossem rapidamente testadas.

Em relação aos trabalhos correlatos apresentados, verifica-se que o trabalho proposto atende parte das funcionalidades existentes nas ferramentas disponíveis no mercado, mas de uma forma diferente das encontradas, uma vez que as informações estão no aplicativo móvel e sincronizando em tempo real com o software em ambiente web acessado por um *desktop*, e esta torna-se a principal diferença e característica deste software.

Por fim, obteve-se um sistema funcional que possibilitou a utilização de uma variedade de conceitos, e recursos do ambiente de desenvolvimento da plataforma Android.

4.1 EXTENSÕES

Sugerem-se as seguintes extensões para trabalhos futuros:

- a) Adicionar ao aplicativo do *smartphone* geolocalização, permitindo saber a localização do técnico;
- b) adicionar ao aplicativo do *smartphone* inteligência artificial para informar a melhor rota;
- c) adicionar ao aplicativo servidor a emissão de relatórios de acompanhamento de chamados pendentes, finalizados, e concluídos por técnico;
- d) implementar ao aplicativo servidor telas para cadastro de clientes;
- e) implementar segurança de forma que garanta a integridade do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID DEVELOPERS. **Android:** an open handset alliance project. [S.l.], 2013.

Disponível em: <<http://code.google.com/android>>. Acesso em: 02 mar. 2013.

APACHE. **Apache:** REST architecture. [S.l.], 2009. Disponível em:

<<http://wink.apache.org/documentation/1.0/html/1%20Introduction%20to%20Apache%20Wink.html>>. Acesso em: 04 mar. 2013.

AQUINO, Juliana F. S. **Plataformas de desenvolvimento para dispositivos móveis.** 2007.

14 f. Monografia (Pós Graduação em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <<http://www-di.inf.puc-rio.br/~endler/courses/Mobile/Monografias/07/Android-Juliana-Mono.pdf>>.

Acesso em: 06 mar. 2013

AURÉLIO, Marco. **O que você sabe sobre software de helpDesk.** [S.l.], 2008. Disponível

em: <[HTTP://www.malima.com.br/article_read.asp?id=137](http://www.malima.com.br/article_read.asp?id=137)>. Acesso em: 16 out. 2011.

DAMASCENO, J. C. **Introdução à composição de serviços web.** [S.l.], 2009. Disponível

em:<<http://www.ufpi.br/ercemapi/pesquisa/?q=DAMASCENO%2C+J.C.Introdu%27%E3o+%E0+Composi%27%E3o+de+Servi%27o+&image.x=-1164&image.y=-266>>. Acesso em: 12 fev. 2013.

EASY BINARY OPTION. **Transação de opções binárias via dispositivos móveis.** [S.l.],

2010. Disponível em: < <http://easybinaryoption.com/pt/trading/mobile-binary-trading>>.

Acesso em: 20 mar. 2013.

FÜHR, Daniel. **Análise e modelagem de um Sistema de CRM para dispositivos móveis.**

2007. 127 f. Trabalho de Conclusão (Bacharelado em Sistemas de Informação) – Instituto de Ciências Exatas e Tecnológicas, Centro Universitário Feevale, Novo Hamburgo. Disponível em: <tconline.feevale.br/tc/files/0002_1274.doc>. Acesso em: 02 mar. 2013.

FOX, DAN. **Building solutions with the Microsoft .net Compact Framework**. Boston: Pearson Education, 2003.

LECHETA, Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos móveis com Android SDK**. São Paulo: Novatec, 2009.

POSSER, Lucas S. **Computação móvel e mlearning: estudo e construção de um protótipo para Smartphone**. 2006. 83 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas da Informação) – Centro Universitário Ritter dos Reis, Porto Alegre. Disponível em: <http://www.uniritter.edu.br/graduacao/informatica/sistemas/downloads/Computacao_Movel_e_M-Learning.pdf>. Acesso em: 02 mar. 2013.

MARTINELLI, Rafael. **Dispositivos móveis: uma nova realidade**. [S.l.], 2011. Disponível em: <<http://www.dclickholmes.com/blog/dispositivos-moveis-uma-nova-realidade/>>. Acesso em: 16 mar. 2013

MENÉDEZ, Andrés I. M. **Plataformas de desenvolvimento para dispositivos móveis**. 2002. 93 f. Monografia (Pós Graduação em Informática) – Centro de Ciências e Tecnologia, Universidade Federal de Campina Grande, Campina Grande. Disponível em: <http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2002/Dissertacao_AndresIgnacioMartinezMenendez.pdf>. Acesso em: 06 mar. 2013

PEKUS CONSULTORIA E DESENVOLVIMENTO. **Dispositivos móveis**. [S.l.], 2010. Disponível em: <<http://www.pekus.com.br>>. Acesso em: 20 mar. 2011.

QTUX. **Controle e gestão de chamados técnicos**. Jundiaí, 2010. Disponível em: <<http://www.qtux.com.br/institucional/chamados>>. Acesso em: 27 mar. 2013.

RICHARDSON, Leonard; RUBY, Sam. **RESTful Web Services**. USA: O' Reilly, 2007.

SCALA. **Scala: a tour of scala**. [S.l.], 2013. Disponível em: <<http://www.scala-lang.org/node/104>>. Acesso em: 23 mar. 2013.

SCHAEFER, Carine. **Protótipo de aplicativo para transmissão de dados a partir de dispositivos móveis aplicado a uma empresa de transporte**. 2004. 51 f. Trabalho de Conclusão de Curso II (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2004/279049_1_1.pdf>. Acesso em: 02 mar. 2013.

SECURE.NET. **Gestão de chamados**. [S.l.]: Secure.net, 2010. Disponível em: <<http://blog.securenet.inf.br/secoes/gest%C3%A3o-de-chamados>>. Acesso em: 20 fev. 2013

SQLITE. **SQLite**. [S.l.], [2011]. Disponível em: <<http://www.sqlite.org/docs.html>>. Acesso em: 20 fev. 2013.

UTECS. **WebAssist**. Cascavel, 2010. Disponível em: <[http:// http://www.webassist.com.br](http://http://www.webassist.com.br)>. Acesso em: 27 mar. 2011.