UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

JOÃO PAULO CATTANI

APLICATIVO PARA GERENCIAMENTO DE NÃO CONFORMIDADES EM OCORRÊNCIA DE TRANSPORTE DE MERCADORIAS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

JOÃO PAULO CATTANI

APLICATIVO PARA GERENCIAMENTO DE NÃO CONFORMIDADES EM OCORRÊNCIA DE TRANSPORTE DE MERCADORIAS

Trabalho de Conclusão de Curso, apresentado ao III Curso de Especialização em Tecnologia Java, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2015

APLICATIVO PARA GERENCIAMENTO DE NÃO CONFORMIDADES EM OCORRÊNCIA DE TRANSPORTE DE MERCADORIAS

Por

João Paulo Cattani

Esta monografia foi apresentada ás 14h30 do dia 3 de novembro de 2015 como requisito parcial para a obtenção do titulo de ESPECIALISTA, no III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profa. Dra. Beatriz Borsoi
Orientadora
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Vinicius Pegorini
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

RESUMO

CATTANI, João Paulo. Aplicativo para gerenciamento de não conformidades em ocorrência de transporte de mercadorias. 63 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

A melhoria da agilidade na resolução de não conformidades em uma empresa bem como a adequada documentação dessas não conformidades melhora a qualidade dos serviços prestados pela empresa. Nesse sentido, o presente trabalho relata o desenvolvimento um aplicativo para dispositivos móveis para o gerenciamento de não conformidades em ocorrências de transporte de mercadorias. Foram desenvolvidas uma aplicação cliente (Java Android) e uma aplicação servidor (Java). O lado servidor faz o controle de login e roteamento das informações enviadas pelos clientes. O lado cliente cria, atualiza e deleta ocorrências além de efetuar a persistencia dos dados. Na comunicação entre cliente e servidor foram utilizados os serviços de envio e recepção de mensagens disponibilizados gratuitamente pelos servidores do *Google Cloud Messaging* (GCM), para isso foram utilizadas as bibliotecas Smack (lado servidor) e API do Google Services (cliente). A troca de mensagens entre cliente e servidor foi efetuada por meio de objetos serializados em strings *JavaScript Object Notation* (JSON) utilizando a biblioteca Gson. Para o banco de dados no lado cliente foi utilizado o SQLite e no lado servidor por meio de arquivo *eXtensible Markup Language* (XML) manipulado pela biblioteca Smack. O desenvolvimento reportado neste trabalho trata desde a etapa de concepção da ideia até o desenvolvimento final. Como resultado tem-se um aplicativo que proporciona um ambiente simples focado na resolução de não conformidades em ocorrências de transporte de mercadorias, onde é possivel acompanhar, visualizar, relatar, atualizar ocorrências bem como enviar mensagens de colaboração.

Palavras-chave: Android. Java. Transporte de cargas. Não conformidades. GCM.

ABSTRACT

CATTANI, João Paulo. Software for management of nonconformities in cargo transport ocurrences. 2015. 63 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

The Improvement of agility in non conformities's resolution in a company as well as its documentation improves the overall quality of the company. In this sense the present paper describes the development an application for managing nonconformities in cargo transport occurrences. A client application (Java Android) and a server application (Java) have been developed. The server side does the login control and information routing sent by client aplications. The client side creates, update, and delete nonconformities events, besides making the data persistence. The communication between client and server have used the services for sending and receiving messages available for free by Google Cloud Messaging (GCM) servers, for this we used the Smack library (server side) and Google Services API (client). The messaging between client and server was made through serialized objects by JavaScript Object Notation (JSON) strings using the Gson library. The database on the client side was made through the SQLite and side server by eXtensible Markup Language (XML) file manipulated by the Smack library. The development reported in this work comes from the idea's conception phase until the final development. As a result, there is an application that provides a simple environment focused on solving nonconformities in cargo transport where it is possible to monitor, visualize, report, update nonconformities events and send collaboration´s messages.

Keywords: Android. Java. Cargo Transport. Nonconformities. GCM.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
CTRC	Conhecimento de Transporte Rodoviário de Carga
GCM	*Google Cloud Message*
CCS	Cloud Conection Server
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
MDF-e	Manifesto Eletrônico de Documentos Fiscais
XML	*EXtensible Markup Language*
XMPP	*Extensible Messaging and Presence Protocol*
XML	*eXtensible Markup Language*
ROO	Resposta Rápida a Ocorrências

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações inicias, os objetivos e a justificativa da realização do trabalho. No final do capítulo é apresentada a organização do texto por meio de uma sucinta descrição dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

O processo de transporte e entrega de mercadorias a clientes, que é um dos integrantes da logística, é composto por uma série de etapas e com diversas pessoas envolvidas. Quando um mesmo veículo transporta mercadorias para clientes distintos, esse transporte é acompanhado por um documento denominado manifesto.

O documento de manifesto de cargas atualmente possui uma versão eletrônica, chamado de Manifesto Eletrônico de Documentos Fiscais (MDF-e), mas o processo tradicional de acompanhamento de entregas não é alterado em decorrência da existência desse documento eletrônico. Na empresa para a qual o sistema foi desenvolvido uma via impressa do MDF-e acompanha a entrega para conferência, como ocorre com o manifesto de cargas tradicional.

Visando resposta rápida à identificação de não conformidades no processo de distribuição de mercadorias acompanhadas por manifesto, neste trabalho é apresentado um aplicativo computacional que foi desenvolvido com o objetivo de agilizar o encaminhamento dessas não conformidades. Não conformidades estão relacionadas a ocorrências que impedem ou dificultam a chegada da mercadoria correta ao seu respectivo destinatário, na quantidade certa, no tempo previsto e sem avarias. Mercadorias que são entregues em outro endereço que não o do destinatário e mercadorias danificadas durante o transporte são exemplos de não conformidades para o escopo deste trabalho. Esse aplicativo permitirá o gerenciamento das ações imediatas tomadas nos casos de ocorrência de não conformidades com encomendas. O sistema criará um ambiente de colaboração entre os envolvidos no processo.

O sistema foi desenvolvido para ambiente Android que possuam *Application Programming Interface* (API) 15 ou superiores pois englobam a maioria dos aparelhos do mercado. A aplicação servidor será desenvolvida em java, e a persistencia dos objetos nessa versão será feita através de arquivos *eXtensible Markup Language* (XML), porém a forma de implementação permitirá uma facil migração para utilização de um banco de dados.

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho. O sistema desenvolvido como resultado da realização deste trabalho é um aplicativo que visa atender esses objetivos.

1.2.1 Objetivo Geral

Desenvolver um aplicativo para agilizar o encaminhamento e a resolução de não conformidades em transporte de mercadorias.

1.2.2 Objetivos Específicos

- Oferecer uma maneira de enviar informes sobre eventos não planejados aos envolvidos no transporte de mercadorias agilizando a identificação do problema e possível solução.
- Prover uma forma de gerenciamento de ocorrências em transporte de mercadorias.
- Permitir o registro das atividades realizadas na resolução de uma não conformidade, facilitando consultas posteriores.

1.3 JUSTIFICATIVA

A justificativa de desenvolvimento do aplicativo é pautada na agilidade que o mesmo pode trazer para a identificação e o tratamento de não conformidades no transporte de mercadorias. Uma não conformidade pode estar relacionada à não existência de mercadoria para entregar ao cliente como consta no manifesto de transporte; à sobra de mercadorias para entrega ao chegar no final do trajeto de entrega; à discrepância entre quantidade registrada e existente para entrega para determinado cliente, entre outros. Por meio do sistema, assim que uma não conformidade é identificada uma ocorrência é aberta por quem a identificou. E todos os envolvidos no processo são comunicados.

Com o uso do aplicativo, o informe da ocorrência entre os envolvidos no processo é imediato, agilizando a localização do problema e o encaminhamento da solução possível. Com o aplicativo uma espécie de *broadcast* é gerada entre todas as pessoas vinculadas ao

transporte das mercadorias. Assim, todos ficam imediatamente informadados da ocorrência e a solução pode, em muitos casos, também, ser imediata. O aplicativo facilitará a rastrear as ocorrências durante o trajeto das mercadorias e, assim, mais facilmente identificar a solução do problema envolvido na ocorrência.

Optou-se por um aplicativo para dispositivos móveis pelo fato de esse tipo de aplicativo facilitar o acesso pelos envolvidos no processo e agilizar a comunicação entre eles.

1.4 ESTRUTURA DO TRABALHO

A sequência deste texto está organizada em capítulos. O Capítulo 2 apresenta o referencial teórico que é sobre logística e transporte de mercadorias. No capítulo 3 estão os materiais e o método utilizados para modelar e desenvolver o aplicativo. O resultado da realização do trabalho está no capítulo 4. Por fim estão as considerações, seguidas das referências utilizadas no texto.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho, centrado na distribuição de produtos e logística por ser a fundamentação conceitual do aplicativo desenvolvido.

2.1 CONTEXTO

A distribuição de produtos pode ser vista de diversas perspectivas, decorrentes dos interesses dos envolvidos (MACHADO, 2006). Para os profissionais de logística a distribuição de produtos é, em geral, denominada por distribuição física e envolve os processos relativos ao deslocamento de produtos de um ponto de inicio a um ponto final, podendo passar por pontos intermediários. Em termos de ciclo completo de produção de um produto, o ponto inicial é o produtor da matéria-prima e o ponto final é o consumidor. Contudo, vários processos menores podem ocorrer nesse ciclo de vida. Por exemplo, se é transporte de produto que é matéria-prima, o ponto inicial é o produtor e o ponto final o consumidor, mas passa por uma série de intermediários (responsável pela transformação industrial, atacadista, revendedor, entre outros); se são produtos para venda ao consumidor *in natura* a origem é o produtor e o destino é o consumidor final e pode não haver intermediários.

Independentemente da abrangência do processo ou das etapas do ciclo de transporte, a ótica dos profissionais envolvidos no transporte de mercadorias está voltada para questões de natureza material e inclui assuntos como depósitos, veículos de transporte, estoques e/ou armazenamento, equipamentos de carga e descarga, pessoas responsáveis pelas diversas etapas envolvidos, documentos manipulados, entre outros (MACHADO, 2006). O transporte de mercadorias, independentemente da sua natureza ou destino, faz parte de um processo denominado logística.

2.2 LOGÍSTICA

Logística é definida como um processo de planejamento, implementação e controle do fluxo desde a origem até o destino e do armazenamento e de controle de estoque de produtos ou de matérias-primas (BALLOU, 2001). A logística é a área da *supply chain* (cadeia de

suprimentos) que planeja e controla o fluxo e o estoque de bens, serviços e informações do ponto de origem até o ponto final (MARQUES, 2010).

Logística é uma indústria de serviços abrangente, integrada com tráfego e transporte, entrega e armazenamento, embalagem e outros recursos serviços (WU; JU; SU, 2010). Para esses autores o desenvolvimento da logística está fortemente relacionado ao desenvolvimento da economia. E, ainda, que a logística desempenha um papel chave de com outras indústrias e no transporte de bens e serviços nos mercados doméstico e internacional.

Paura (2012) define transportes, manutenção de estoque e processamento de pedidos como as atividades primárias da logística. E esse autor considera essas atividades como bases da logística porque contribuem com a maior parcela do custo total da logística ou porque elas são essenciais para a coordenação e o cumprimento da tarefa de transporte de materiais de um ponto de origem a um ponto de destino. A logística provê serviços de valor agregado por meio de uma combinação ótima de tempo e espaço (PIAO; LI; WANG, 2009).

O profissional de logística empresarial estuda como prover com eficiência, a lucratividade nos serviços de distribuição ao cliente, o fluxo de materiais dentro da empresa, o planejamento de compra, o controle e a organização de estoques, o planejamento de controle da produção e o controle de transporte de embalagens (PAURA, 2012, p. 23).

3 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e as tecnologias utilizadas na implementação do sistema. O capítulo também apresenta a sequência de atividades, denominada método, realizadas.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do aplicativo.

Ferramenta / Tecnologia	Versão	Referência	Finalidade
JAVA SE Development Kit	1.8.0_11	http://www.oracle.com/	Kit para desenvolvimento Java
Android Studio	1.4	https://developer.android.com	*Integrated Development Environment* (IDE) para desenvolvimento java. Utilizado na implementação da aplicação servidor.
Eclipse Luna	4.4.0	https://eclipse.org/	IDE do Google para desenvolvimento de aplicativos Android.
Android SDK Plataform-Tools	23.0.1	http://developer.android.com/	Ferramentas de desenvolvimento para aplicações mobile em plataforma Androir
Smack	3.2.1	https://www.igniterealtime.org/projects/smack/	Biblioteca para criação de clientes *Extensible Messaging and Presence Protocol* (XMPP);
XStream	1.4.7	https://github.com/codehaus/xstream	Biblioteca de serialização que converte objetos Java em XML e *JavaScript Object Notation* (JSON) e vice-versa.
Gson	2.2.4	https://github.com/google/gson	Biblioteca de serialização que converte Objetos Java em JSON e vice-versa.
Google Play Services	8.1.0	https://developers.google.com	API de Serviços do Google.

Quadro 1 – Tecnologias e ferramentas para modelagem e implementação do sistema

3.2 MÉTODO

O manifesto de carga que vinha sendo utilizado pelas empresas de transporte de mercadorias foi substituído pelo Manifesto Eletrônico de Documentos Fiscais. O MDF-e é o documento emitido e armazenado eletronicamente para vincular os documentos fiscais transportados na unidade de carga utilizada (MANIFESTO…, 2015). A validade jurídica desse documento é assegurada pela assinatura digital do emitente e autorização de uso pelo ambiente autorizador (MANIFESTO…, 2015). A finalidade do MDF-e é agilizar o registro em lote de documentos fiscais em trânsito e identificar a unidade de carga utilizada e as demais características do transporte. A autorização de uso do MDF-e implica o registro posterior dos eventos nos documentos fiscais eletrônicos a ele relacionados.

A única diferença para os operadores durante o processo é que o MDF-e é homologado na receita por meio de um arquivo (feito de forma transparente pelo sistema), o restante do processo continua igual ao manifesto tradicional. Na empresa para a qual o sistema foi desenvolvido uma via impressa do NDF-e acompanha os volumes para as conferências, como o manifesto de cargas tradicional.

Os processos envolvidos no envio de encomendas da empresa para a qual o sistema foi desenvolvido são:

1. recepção de volumes para envio;
2. emissão de Conhecimento de Transporte Rodoviário de Carga (CTRC);
3. carregamento e emissão de manifestos;
4. chegada de veículos x manifestos;
5. entrega ao cliente.

A fase de recepção dos volumes para envio é representada na Figura 1.

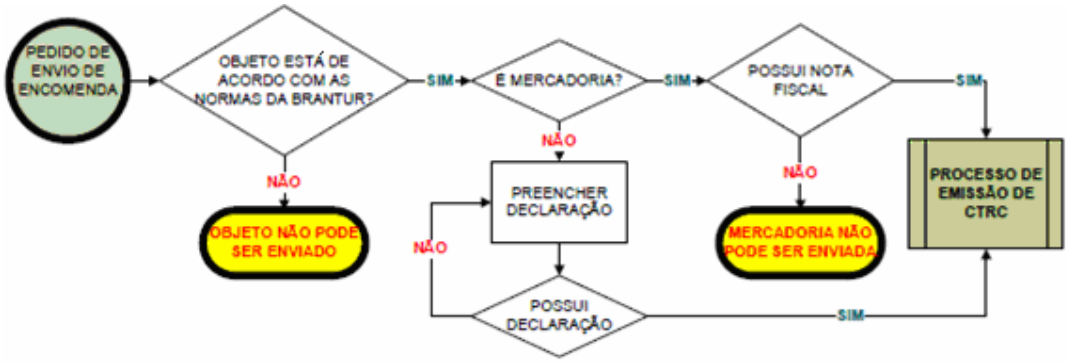


Figura 1 – Recepção de volume para envio

O CTRC é o documento fiscal emitido pela transportadora de carga para acompanhar o transporte da mercadoria entre a origem e o destino da carga e para definir receita e recolhimento de impostos e taxas. O processo de emissão de CTRCs é representado na Figura 2.

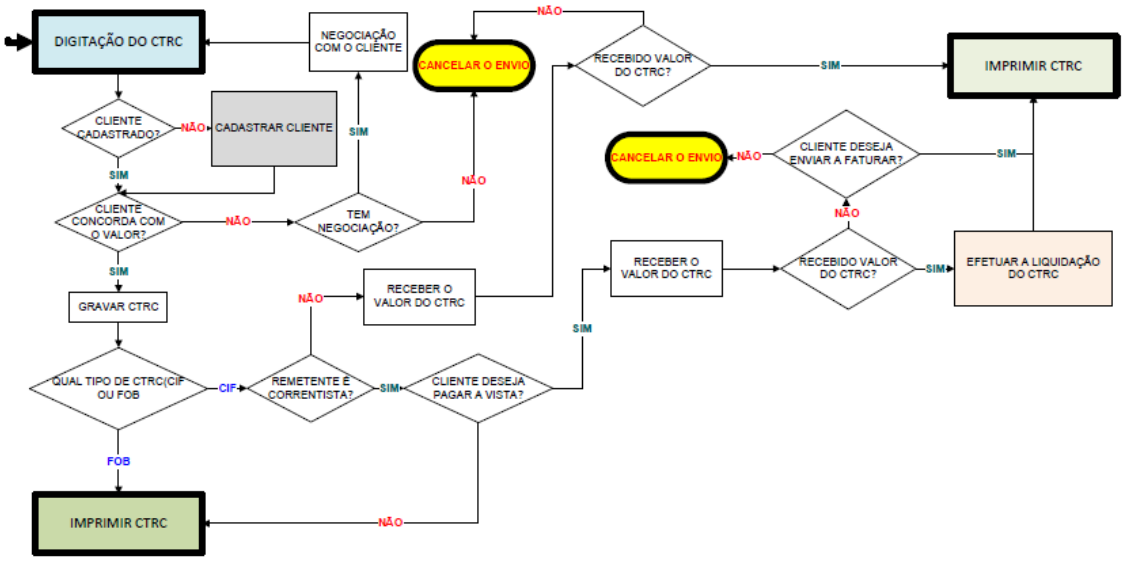


Figura 2 – Emissão de CTRCs

O manifesto de carga é um documento utilizado pelas transportadoras de cargas que contém relacionados todos os conhecimentos de transporte quando a carga é fracionada, ou seja, um veículo transporta mercadorias para destinatários distintos. A Figura 3 apresenta o processo para carregamento do veículo e emissão do manifesto.

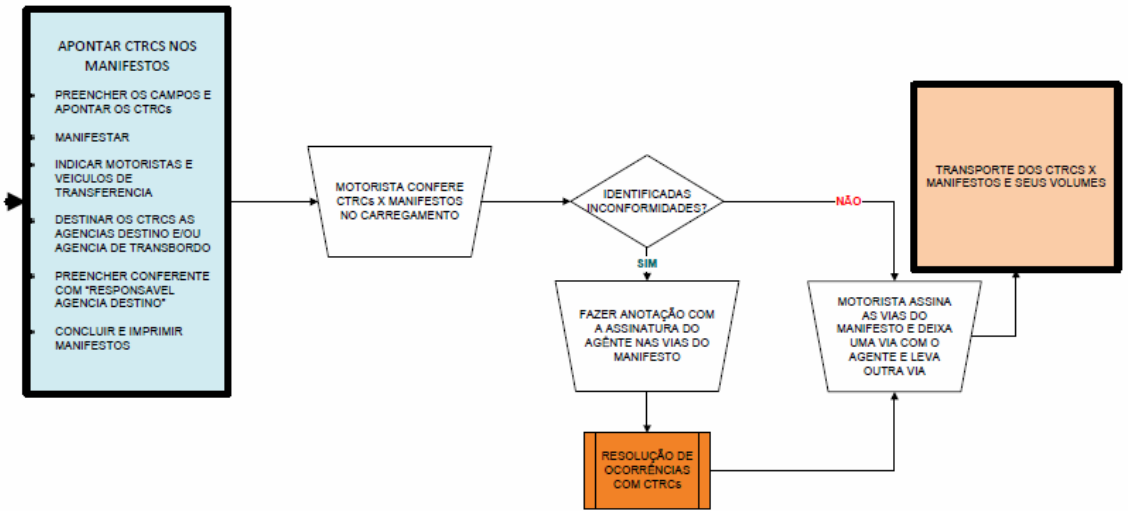


Figura 3 – Carregamento e emissão de manifesto

A chegada dos veículos aos seus destinos e dos respectivos manifestos é representada na Figura 4.

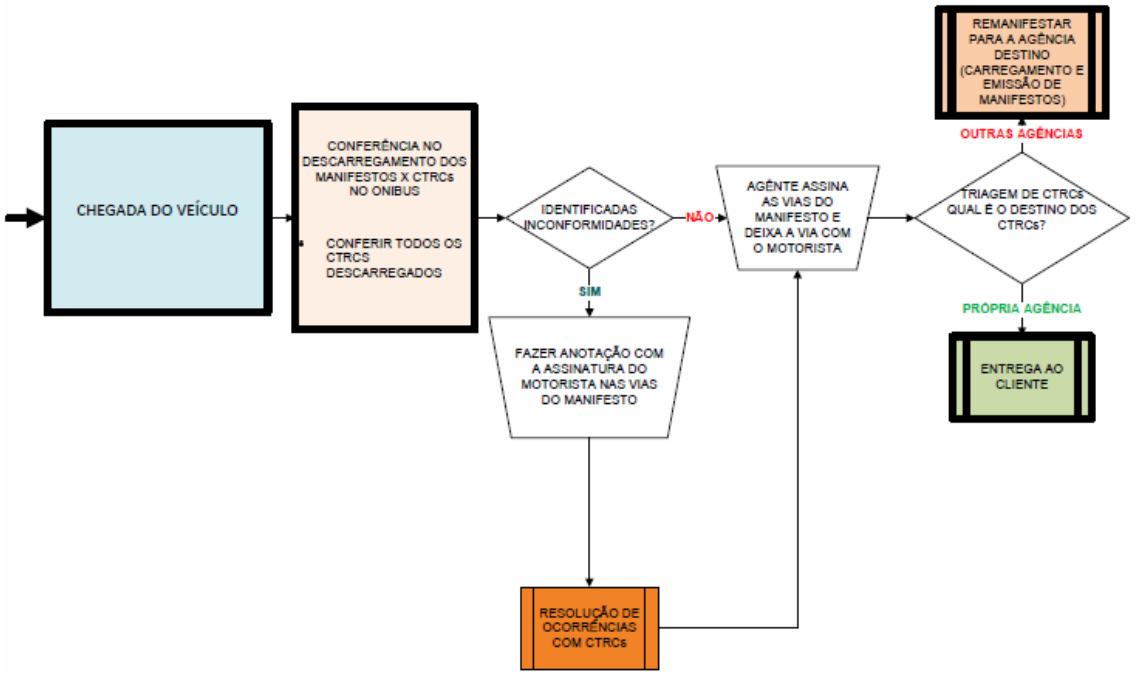


Figura 4 – Chegada de veículos e manifestos

O processo de entrega das mercadorias aos clientes é representado na Figura 5.

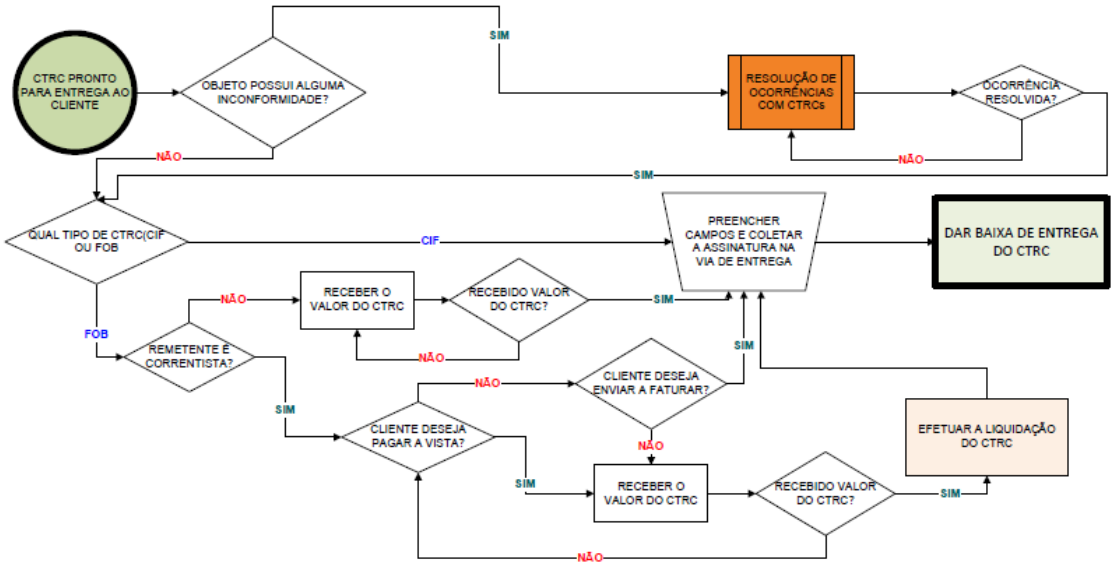


Figura 5 – Entrega ao cliente

Em cada fase, não conformidades com as encomendas podem ocorrer. Nesse momento é que o gestor das agências da empresa Branbus Transporte e Fretamento Ltda

identificou a possibilidade de agilizar e reduzir os custos de comunicação entre os envolvidos nos processos de encomendas.

Atualmente, em uma ocorrência de não-conformidade, a comunicação na empresa é efetuada por meio de Skype, *email* ou telefone. As ocorrências geralmente estão relacionadas à falta de mercadoria para entregar para o cliente como consta no manifesto de transporte; à sobra de mercadorias para entrega ao chegar no final do trajeto de entrega; discrepância entre quantidade registrada e existente para entrega para determinado cliente, volumes avariados entre outros. Em alguns tipos de ocorrência, por exemplo, falta de mercadorias, é provavel que elas sobrar em outra agência de encomendas. Nesse sentido é necessário comunicar agência por agência sobre a ocorrência até identificar a localização das mercadorias faltantes.

Para agilizar a comunicação foram pesquisadas outras soluções, sendo cogitado pelo gestõr a implantação do aplicativo WhatsApp para informar as ocorrências. Porém, o tratamento de todas as ocorrências acaba ficando mesclado, no sentido de as mensagens não serem organizadas por ocorrência, podendo gerar confusão. Nesse sentido, foi idealizado um aplicativo para envio de mensagens estilo WhattsApp no qual as mensagens de colaboração são separadas por ocorrência. Ao enviar uma nova ocorrência, todos os envolvidos são informados, ficando cientes e podem colaborar para a resolução. Os requisitos básicos informados pelo gestor das agências de encomendas são descritos no Quadro 2 da Seção 4.2 Modelagem do Sistemas.

A seguir estão as fases do ciclo de vida, de acordo com Pressman (2006), para o desenvolvimento do sistema.

Levantamento de Requisitos

Os requisitos foram levantados levando em consideração as necessidades do getor das agências de encomendas que identificou gastos de tempo e resursos desnecessários na resolução de não-conformidades.

Testes de Viabilidade

Antes de iniciar a análise do projeto, foi efetuada uma pré-analise das funcionalidades críticas do sistema. Por meio dessa análise foi identificado que a troca de mensagens utilizando o servidor do *Google Cloud Message* (GCM) seria o fator determinante para a implementação do sistema. Ao identificar a viabilidade deu-se inicio a análise e desenvolvimento.

Análise e Desenvolvimento

Inicialmente foi feita uma analise básica do sistema com diagramas de classes básicos, somente para nortear o inicio do desenvolvimento. Primeira etapa do

desenvolvimento foi criar o pacote br.com.brantur.jp com as classes que implementam as tecnologias necessárias para o desenvolvimento do software, porém sem se preocupar com a regra do negócio. Nesse processo outras classes temporárias foram criadas para testar as funcionalidades. Após o desenvolvimento do pacote br.com.brantur.jp, o pacote que implementa a regra de negócio começou a ser implementado. Durante o desenvolvimento, prototipos com funcionalidades simples eram gerados para testes simples até chegar na versão com os requisitos básicos de funcionamento.

Testes

Os testes do sistema foram feitos informalmente. Inicialmente os testes foram separados por funcionalidades. Após teste de grupos de funcionalidades foram efetuados visando verificar e validar o funcionamento dos recursos implementados em conjunto.

4 RESULTADOS

Este capítulo apresenta o aplicativo desenvolvido como resultado deste trabalho. Na Seção 4.1 é apresentada uma visão geral das funcionalidades e do escopo do sistema. Essas funcionalidades são apresentadas e modeladas como requisitos na Seção 4.2. A apresentação das funcionalidades do aplicativo é realizada por meio das suas telas com explicação dos recursos na Seção 4.3. A Seção 4.4 estão partes do código implementado com objetivo de mostrar o uso da tecnologia no desenvolvimento dos requisitos do aplicativo.

4.1 ESCOPO DO SISTEMA

O aplicativo tem como objetivo principal ser uma ferramenta para propiciar resposta rápida a não conformidades no transporte de encomendas, permitindo assim, o gerenciamento das ações imediatas tomadas nos casos de não conformidades. O sistema criará um ambiente de colaboração simples, mas que visa ser eficaz, entre os envolvidos no processo. Simples, porque serão emitidas mensagens objetivas sobre não conformidades ocorridas. E eficaz porque essas mensagens serão disparadas para todos os envolvidos no processo, agilizando a localização exata do problema e a respectiva tomada de decisão para encaminhamento da solução, quando houver.

Com o uso do aplicativo, instalado nos dispositivos móveis utilizados pelos funcionários da empresa, ao identificar uma não conformidade, o colaborador cria uma ocorrência no sistema fazendo um resumo para facilitar a identificação da ocorrência e uma descrição. Após a criação da ocorrência, todos os envolvidos são comunicados.

Por meio do ambiente criado pelo sistema, os envolvidos podem registrar todos os passos realizados no encaminhamento da solução da não conformidade. Após resolução da não conformidade é realizada a identificação de ocorrência resolvida no sistema. Esse *status* será avaliado pelo gestor geral do sistema. E se a solução for considerada eficaz na resolução da não conformidade, a ocorrência é finalizada. Caso contrário é reaberta e um novo processo de encaminhamento da solução é iniciado. Em determinados casos pode ocorrer uma decisão em que a ocorrência não será resolvida, nesses casos a ocorrência pode ser finalizada sem resolução pelo gestor geral.

O acesso ao sistema pode ser efetuado por meio de celular ou *tablet* com sistema Android. Na primeira versão do sistema somente a funcionalidade de texto estará disponível,

não sendo possível o anexo de arquivo. Relatórios não serão gerados na versão do aplicativo para Android, futuramente será implementada uma versão para *desktop* que contará com a emissão de relatórios gerenciais.

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a listagem dos requisitos funcionais identificados para o sistema. Os Requisitos Funcionais (RF) RF01, RF02, RF03, RF04, RF05, RF06, RF07, RF08 e RF09 serão implementados como resultado da realização deste trabalho de conclusão de curso. Os outros requisitos serão implemenentados no próximo protótipo. Os requisitos R10 a RF16 cosntam aqui para que seja possível definir uma visão geral do aplicativo, embora não todos os requisitos tenham sido implementados nesta versão.

Identificação	**Nome**	**Descrição**
RF01	Controlar login	Os usuários deverão digitar o login e senha no primeiro acesso ao sistema. Os dados estando corretos estes devem ser persistidos na aplicação e no próximo acesso a etapa de login não será mais necessária.
RF02	Listar Ocorrências	A primeira tela do sistema após login deve ser uma tela que lista todas as ocorrências mostrando o resumo de cada ocorrência para identificação, deve conter também opções para relatar nova ocorrência. Ao clicar em alguma ocorrência deverá aparecer uma tela mostrando as informações da ocorrência; Ao clicar em nova ocorrência a tela de cadastro de ocorrência deve aparecer;
RF03	Visualizar Ocorrência	Na tela de lista de ocorrências ao clicar em uma ocorrência uma tela é aberta com os dados da ocorrência, campo de digitação de mensagem e botão de envio de mensagem, bem como itens para alterar o status da ocorrência
RF04	Relatar Nova Ocorrência	Uma ocorrência é composta por resumo e descrição da ocorrência, o sitema deve automaticametne armazenar a data de criação da ocorrência, data de fechamento e status da ocorrência. A tela de nova ocorrência deve conter um campo de resumo e outro de descrição da ocorrência.
RF05	Atualizar Status Ocorrênia	O usuário poderá alterar o status da ocorrência nas opções do menu de atualização da tela de ocorrência. Ao alterar o status da ocorrência a mesma deve ser atualizada na tela.
RF06	Enviar Mensagem	Cada usuário poderá enviar várias mensagens para uma ocorrência. As mensagens serão compostas de um campo texto da mensagem, identificação do autor da mensagem e identificação da ocorrência a qual a mensagem se refere. Na tela de ocorrência, ao preencher o campo mensagem e clicar em enviar, a mensagem deve ser enviada para todos os envolvidos.

RF07	Receber atualização de ocorrência	Ao receber dados de atualização de uma ocorrência a mesma deve ser atualizada na tela com as novas atualizações.
RF08	Receber atualização de ocorrência	Ao receber uma nova ocorrência, ela deve ser persistida e a lista de ocorrências deve ser atualizada na tela.
RF09	Receber Nova Mensagem	Ao receber uma nova mensagem, ela deve ser persistida e a lista de mensagem da tela da ocorrência deve ser atualizada.
RF10	Cadastrar usuários	Os usuários estarão cadastrados na aplicação servidor. Com dados de nome, login, senha, email, identificador de registro, *status*.
RF11	Interface CRUD usuários	Inteface CRUD para o cadastro de usuários.
RF12	Emitir relatório gerenciais	Emitir relatórios gerenciais com informações sobre as ocorrências.
RF13	Cadastrar Categorias de ocorrências	Criar categorias para as ocorrências.
RF14	Aplicação cliente para Desktop	Interface cliente para *desktop*.
RF15	Geração de relatórios	Relatórios gerenciais na aplicação servidor.
RF16	Persistir dados na aplicação servidor	Persistencia das ocorrências e mensagens na aplicação servidor.

Quadro 2 – Requisitos funcionais

A listagem do Quadro 3 apresenta os requisitos não-funcionais do sistema. Os Requisitos Não Funcionais (RNF) explicitam regras de negócio, restrições ao sistema de acesso, requisitos de qualidade, desempenho e segurança, dentre outros.

Identificação	**Nome**	**Descrição**
RNF01	Acesso ao sistema	O primeiro acesso ao sistema será realizado por meio de *login* e senha. Esses dados devem ser persistidos para não realizar a etapa de login no próximo acesso.
RNF02	Comunicação	A troca de informações entre servidor e aplicação cliente será por meio dos serviços do Google Cloud Message. O envio e o recebimento de mensagens será assincrono.
RNF02	Base de dados	Na persistência dos dados na aplicação cliente será utilizado o SQLite.
RNF03	Aplicação cliente	A aplicação cliente deve ser implementada para ambiente Android API 15.
RNF04	Aplicação servidor	A aplicação servidor deve ser implementada em Java.

Quadro 3 – Requisitos não funcionais

Levando em consideração que serão atendidos os requisitos RF01, RF02, RF03, RF04, RF05, RF06, RF07, RF08 e RF09 nessa versão do sistema, é mostrado na Figura 6 casos de uso do sistema relacionados a esses requisitos para o sistema definido por Resposta Rápida a Ocorrências (RRO).

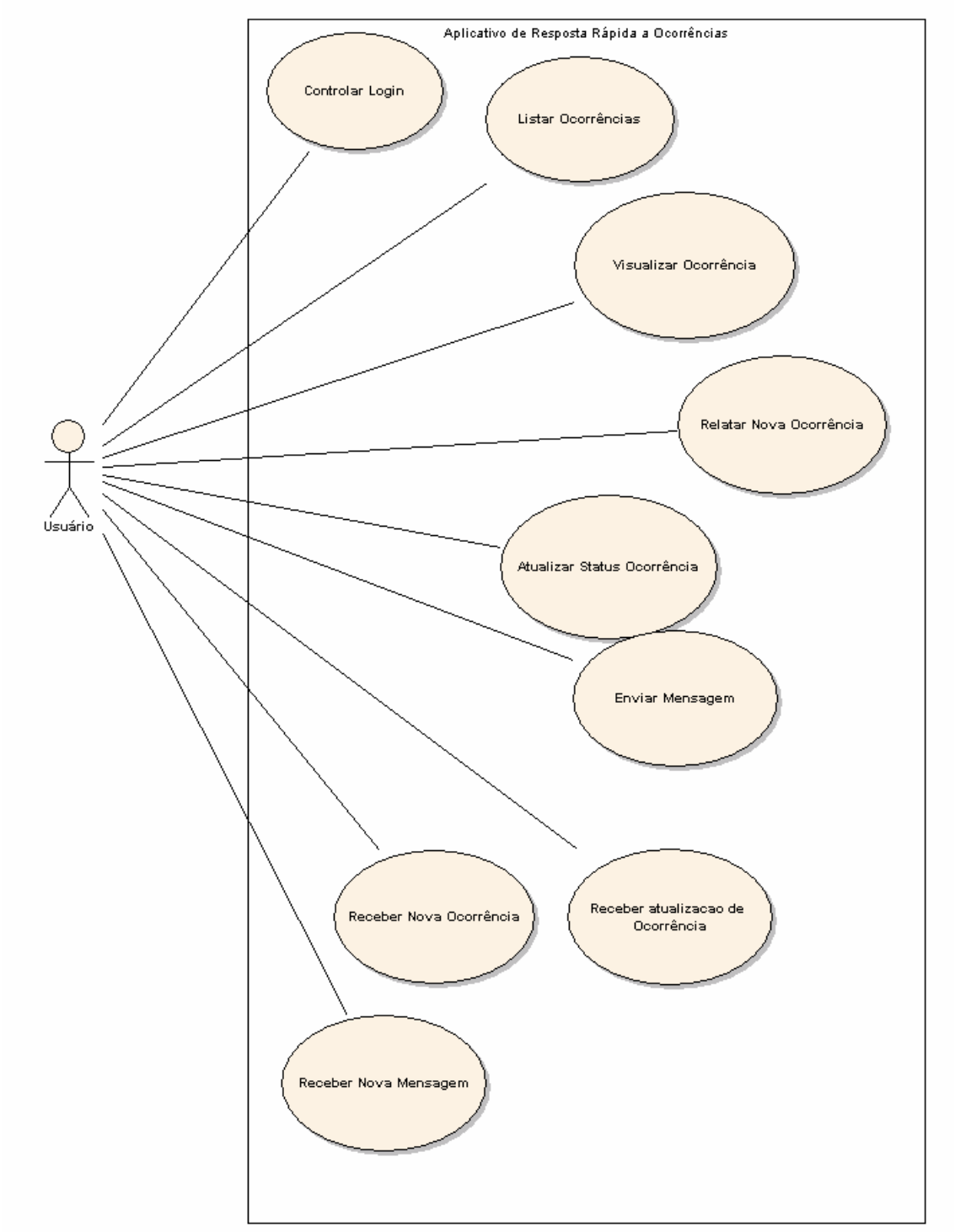


Figura 6 – Diagrama de Casos de Uso do Aplicativo RRO

O Quadro 4 mostra a descrição dos casos de uso *Controlar Login*, *Listar Ocorrências*, *Visualizar Ocorrência*, *Relatar Nova Ocorrência*, *Atualizar Status Ocorrência*, *Enviar Mensagem*, *Receber Atualização de Ocorrência*, *Receber Nova Ocorrência*, *Receber Nova Mensagem*.

Nome do Caso de Uso: Efetuar Login
Atores: Usuários
Pré Condições:
O ator deverá estar cadastrado na base de dados do Servidor
Cenário Principal:

1. O caso de uso inicia quando o sistema apresenta uma tela solicitando o login e a senha do usuário;
2. O usuário digita o login e senha;
3. O usuário clica no botão para entrar no sistema;
4. O sistema valida o login e senha;
5. Se o login for válido, o sistema persiste os dados de login no aparelho;
6. Inicia o *Caso de Uso Listar Ocorrências*;
7. Termina o caso de uso.

Cenário Alternativo 1:

1. Antes do passo 3 do Cenário Principal, o cliente clica no botão voltar antes do envio de dados do login;
2. Clicando no botão voltar implica no reinicio do Caso de Uso.

Cenário Alternativo 2:

1. Após o passo 4 do Cenário Principal, se o login for inválido, o sistema informa o usuário;
2. O caso de uso é reiniciado.

Cenário Alternativo 2:

1. Os dados de login já estão persistidos;
2. Inicia o *Caso de Uso Listar Ocorrências*;
3. Termina o caso de uso.

Nome do Caso de Uso: Listar Ocorrências
Atores: Usuários
Pré Condições:
O ator deve ter efetuado login
Cenário Principal:

1. O sistema carrega uma lista com um resumo de todas as ocorrências e um menu com um botão para nova ocorrência;
2. O usuário clica sobre uma ocorrência;
3. Inicia o *Caso de Uso Listar Ocorrências*;
4. Termina o caso de uso.

Cenário Alternativo 2:

1. No passo 1 do Cenário Principal, é recebida uma nova ocorrência;
2. A lista de ocorrências é atualizada.

Cenário Alternativo 3:

1. No passo 1 do Cenário Principal, o usuário clica no botão voltar;
2. Fecha a lista de ocorrências;
3. Termina o caso de uso.

Nome do Caso de Uso: Visualizar Ocorrência
Atores: Usuários
Pré Condições:
O usuário deve ter efetuado o login;
A tela anterior deve ser a Tela de Lista de Ocorrências.
Cenário Principal:

1. O sistema carrega uma lista com um resumo, uma descrição, lista de mensagens, e no final da tela um campo de mensagem e botão de enviar mensagem. Um menu com

opções de resolver ocorrência, fechar ocorrência, reabrir ocorrência, fechar sem resolução, deletar ocorrência tambés deve estar disponível.
2. Após o passo 1 o usuário digita uma mensagem e clica em enviar,
3. Inicia o *Caso de Uso Enviar Mensagem.*

Cenário Alternativo 1:
1. No passo 1 do Cenário Principal, o usuário clica em uma das opções disponíveis de atualizar o status da ocorrência;
2. Inicia o *Caso de Uso Atualizar Status Ocorrência;*
3. Ternina o Caso de Uso.

Cenário Alternativo 3:
1. No passo 1 do Cenário Principal, é recebida uma nova mensagem;
2. Inicia o *Caso de Uso Receber Nova Mensagem.*

Cenário Alternativo 4:
1. No passo 1 do Cenário Principal, o usuário clica no botõa voltar;
2. A tela de ocorrência é finalizada e Inicia o *Caso Uso Listar Ocorrências***.**

Nome do Caso de Uso: Relatar Nova Ocorrência
Atores: Usuários
Pré Condições:
O ator deve ter efetuado login;
O ator deve estar no *Caso de Uso Lista de Ocorrências*.
Cenário Principal:
1. O sistema carrega uma Tela com um campo texto para o resumo da ocorrência, um campo de texto para a descrição da ocorrência e um botão de enviar ocorrência.
2. O usuário preenche os campos de resumo e descrição;
3. O usuário clica em Enviar;
4. A lista de ocorrências é atualizada;
5. Retorna para o *Caso de Uso Listar Ocorrências*;
6. Termina o caso de uso.

Cenário Alternativo 2:
1. No passo 2 do Cenário Principal o usuário não preenche os dados dos campos textos;
2. O usuário clica em Enviar;
3. Aparece uma mensagem indicando que os campos devem ser preenchidos;
4. Reinicia o caso de uso.

Cenário Alternativo 3:
1. No passo 1 do Cenário Principal, o usuário clica no botão voltar;
2. Inicia o *Caso de Uso Visualizar Ocorrência*;
3. Termina o caso de uso.

Nome do Caso de Uso: Atualizar Status Ocorrência
Atores: Usuários;
Pré Condições:
O ator deve ter efetuado login;
O ator deve estar no *Caso de Uso Visualizar Ocorrência*;
Deve estar na tela de visualização de ocorrência;
Cenário Principal:
1. O usuário clica em um dos itens de atualização disponíveis no sistema.
2. Aparece uma Tela de confirmação;
3. O usuário confirma;
4. A ocorrência é atualizada com o novo status;
5. Retorna para o *Caso de Uso Visualizar Ocorrência*;
6. Termina o caso de uso.

Cenário Alternativo 2:

1. No passo 3 do Cenário Principal o usuário não confirma;
2. Retorna para o *Caso de Uso Visualizar Ocorrência*;
3. Termina o Caso de Uso.

Cenário Alternativo 3:

1. No passo 1 do Cenário Principal, o usuário clica no botão voltar;
2. Inicia o *Caso de Uso Visualizar Ocorrência*;
3. Termina o caso de uso.

Nome do Caso de Uso: Enviar Mensagem
Atores: Usuários
Pré Condições:
O ator deve ter efetuado login;
O ator deve estar no *Caso de Uso Visualizar Ocorrência*;
Deve estar na tela de visualização de ocorrência.
Cenário Principal:

1. O usuário preenche o campo mensagem;
2. O usuário clica em enviar;
3. A lista de mensagens é atualizada;
4. Retorna para o *Caso de Uso Visualizar Ocorrência*;
5. Termina o caso de uso.

Cenário Alternativo 1:

4. No passo 1 o usuário não preenche o campo de mensagem;
5. O usuário clica em enviar;
6. Retorna para o *Caso de Uso Visualizar Ocorrência*;
7. Termina o Caso de Uso.

Nome do Caso de Uso: Receber Atualização de Ocorrência
Atores: Usuários
Pré Condições:
O ator deve ter efetuado login
Cenário Principal:

1. O usuário está na tela de lista de ocorrências no *Caso de Uso Listar Ocorrências;*
2. É recebida uma mensagem com atualização de uma ocorrência;
3. A lista de ocorrências é atualizada;
4. Retorna para o *Caso de Uso Listar Ocorrências;*
5. Termina o caso de uso.

Cenário Alternativo 1:

1. O usuário esta na tela de visualização de ocorrência no *Caso de Uso Visualizar Ocorrência*;
2. É recebida uma mensagem com atualização de uma ocorrência;
3. Se a ocorrência atualizada é a mesma da tela de visualizar ocorrência;
4. A ocorrência é atualizada;
5. Retorna para o caso de uso *Caso de Uso Visualizar Ocorrência;*
6. Termina o caso de uso.

Cenário Alternativo 2:

1. O usuário esta na tela de visualização de ocorrência no *Caso de Uso Visualizar Ocorrência*;
2. É recebida uma mensagem com atualização de uma ocorrência;
3. Se a ocorrência atualizada for diferente da tela de visualizar ocorrência;
4. Retorna para o caso de uso *Caso de Uso Visualizar Ocorrência*.

Nome do Caso de Uso: Receber Nova Ocorrência

Atores: Usuários
Pré Condições:
O ator deve ter efetuado login
Cenário Principal:

1. O usuário esta na tela de lista de ocorrências no *Caso de Uso Listar Ocorrências;*
2. É recebida uma mensagem com uma nova ocorrência;
3. A lista de ocorrências é atualizada;
4. Retorna para o *Caso de Uso Listar Ocorrências;*
5. Termina o caso de uso.

Cenário Alternativo 1:

1. O usuário esta na tela de visualização de ocorrência no *Caso de Uso Visualizar Ocorrência*;
2. É recebida uma mensagem com uma nova ocorrência;
3. É mostrada uma mensagem indicando uma nova ocorrência;
4. Retorna para o caso de uso *Caso de Uso Visualizar Ocorrência;*
5. Termina o caso de uso

Nome do Caso de Uso: Receber Nova Mensagem
Atores: Usuários
Pré Condições:
O ator deve ter efetuado login
Cenário Principal:

1. O usuário esta na tela de lista de ocorrências no *Caso de Uso Listar Ocorrências;*
2. É recebida uma mensagem de uma ocorrência;
3. É mostrada uma mensagem com o resumo da mensagem recebida;
4. Termina o caso de uso

Cenário Alternativo 1:

1. O usuário esta na tela de visualização de ocorrência no *Caso de Uso Visualizar Ocorrência*;
2. É recebida uma mensagem de uma ocorrência;
3. Se a ocorrência atualizada é a mesma da tela de visualizar ocorrência;
4. A lista de mensagens da ocorrência é atualizada;
5. Retorna para o caso de uso *Caso de Uso Visualizar Ocorrência;*
6. Termina o caso de uso

Cenário Alternativo 2:

1. O usuário esta na tela de visualização de ocorrência no *Caso de Uso Visualizar Ocorrência*;
2. É recebida uma mensagem de uma ocorrência;
3. Se a ocorrência atualizada for diferente da tela de visualizar ocorrência;
4. É mostrada uma mensagem com o resumo da mensagem recebida

Quadro 4 – Descrição dos casos de uso

4.3 APRESENTAÇÃO DO SISTEMA

O Aplicativo ROO é composto por cinco telas: *Tela de Apresentação*, *Tela de Login*, *Tela de Lista de Ocorrências*, *Tela de Ocorrência*, *Tela de Nova Ocorrência*.

A tela inicial do sistema é a *Tela de Apresentação* é mostrada na Figura 7 e contém a Logo da empresa e o nome do Aplicativo. Nesta tela é verificado se os dados necessários para login já estão salvos no dispositivo. Caso estejam salvos é direcionado para a *Tela de Lista de Ocorrências;* caso contrário é direcionado para a *Tela de Login*.

Figura 7 – Tela de apresentação do Aplicativo RRO

A Figura 8 mostra a *Tela de Login,* ela é composta por um EditText para o login e um EditText para a senha e um botão para entrar no sistema.

Se for clicado no Botão entrar no RRO sem o preenchimento do login e/ou senha uma mensagem Toast é mostrada na tela informando que é necessário o preenchimento dos

campos. Se for preenchido login e senha com dados incorretos será apresentada uma mensagem de erro no login/senha e voltará para a tela de login. Caso os campos de login e senha sejam preenchidos corretamente e clicado no botão de Entrar, ao ser validado com sucesso, o login e a senha serão persistidos no dispositivo. Esses dados serão necessários para login e comunicação com o GCM. Assim, enquanto os dados estiverem salvos no dispositivo a Tela de Login não será mais apresentada.

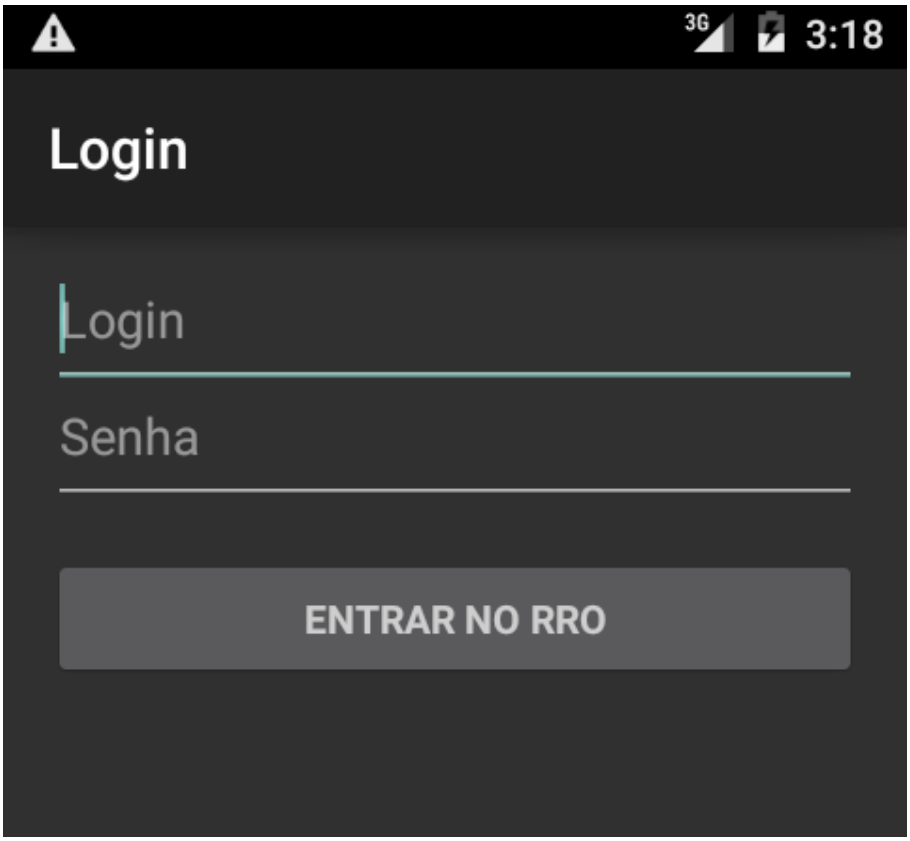


Figura 8 – Tela de Login do Aplicativo RRO

A Figura 11 mostra a *Tela de Lista de Ocorrências,* ela é composta por um menu e celulas do tipo "resumo de ocorrência" que resumem uma ocorrência, a qual é apresentada na Figura 9. Cada objeto de célula é composto por duas TextViews: uma que armazena um resumo de identificação da ocorrência e outra que indica a data de criação da ocorrência. Ao clicar em uma ocorrência abrirá a *Tela de Ocorrência* mostando os detalhes da mesma.

Figura 9 – Células do tipo "resumo de ocorrência"

Ao abrir o menu, será mostrado o item que ao ser clicado abrirá a Tela de Nova Ocorrência (Figura 10) para o registro de relato de uma nova ocorrência.

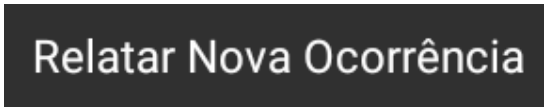


Figura 10 – Acesso ao relato de uma nova ocorrência

A qualquer momento pode ser recebida uma nova ocorrência ou atualização de ocorrência, ocorrendo o recebimento a lista de ocorrências é atualizada.

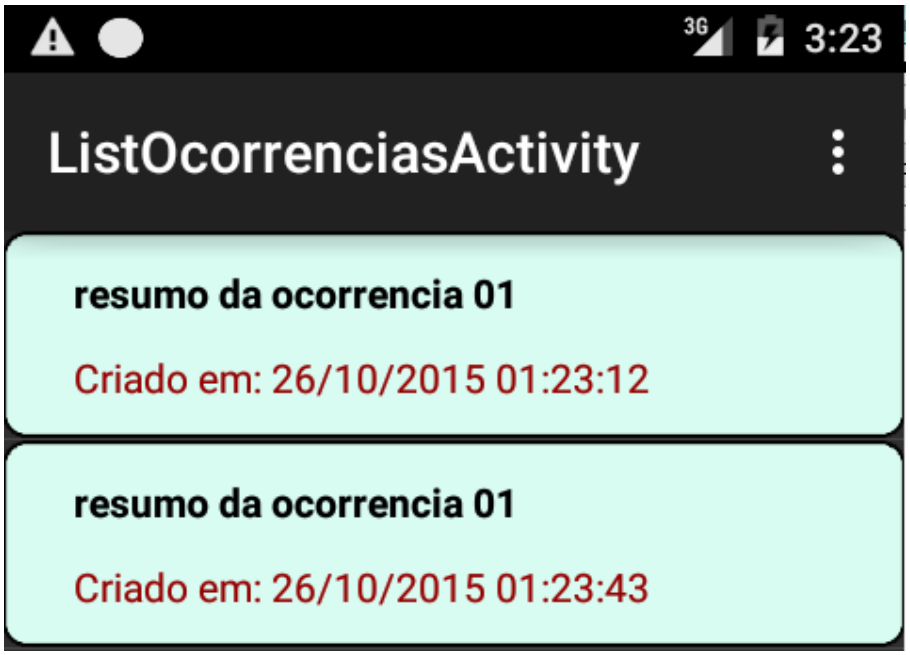


Figura 11 – Tela de Lista de Ocorrências

A Figura 12 mostra a *Tela de Nova Ocorrência.* Ela é composta por um EditText para um resumo da ocorrência, um EditText para a descrição da ocorrência e um botão para informar a ocorrência no sistema.

Se for clicado no Botão informar ocorrência sem o preenchimento do resumo e/ou descrição uma mensagem Toast é mostrada na tela informando que é necessário o preenchimento dos campos. Quando preenchidos o resumo e a descrição e clicado no botão de informar ocorrência a nova ocorrência é enviada para o servidor, a *Tela de Nova Ocorrencia* é fechada retornando para a *Tela de Lista de Ocorrências*.

Figura 12 – Tela de Nova Ocorrência

A Figura 14 mostra a *Tela de Ocorrência* que é composta por um menu com as opções de atualizar o status da ocorrênica, uma lista de celulas do tipo nome de uauário (Figura 13)

Figura 13 – Células do tipo "nome do usuário"

Essas células apresentam as mensagens enviadas na colaboração para resolver a ocorrência. Cada objeto de célula é composto por duas TextViews: uma que armazena o autor da mensagem e outra que mostra a mensagem de colaboração. Abaixo da tela existe um EditText para digitação de novas mensagens e um botão para o Envio dessas novas mensagens. Caso seja clicado no botão Enviar sem preencher o campo de mensagem nenhuma ação é efetuada.

Ao abrir o menu são apresentados os itens [Indicar Resolução da Ocorrência], [Reabrir ocorrência], [Fechar ocorrência], [Fechar sem resolução]. Ao clicar nos itens do menu a atualização do status é enviada para o servidor. A qualquer momento pode ser recebida uma

nova mensagem ou atualização do status da ocorrência, nesse caso a Tela e lista de mensagens é atualizada.

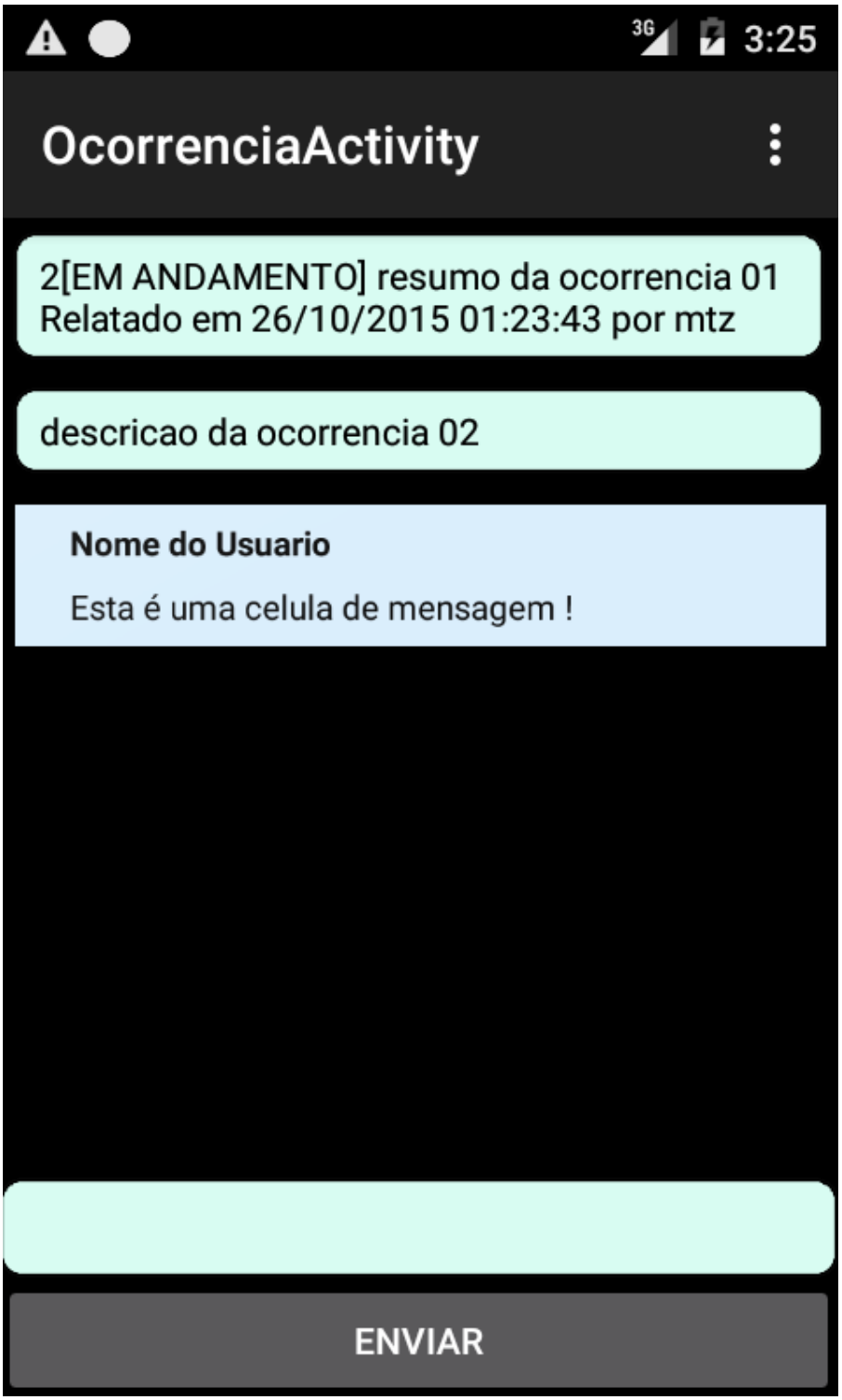


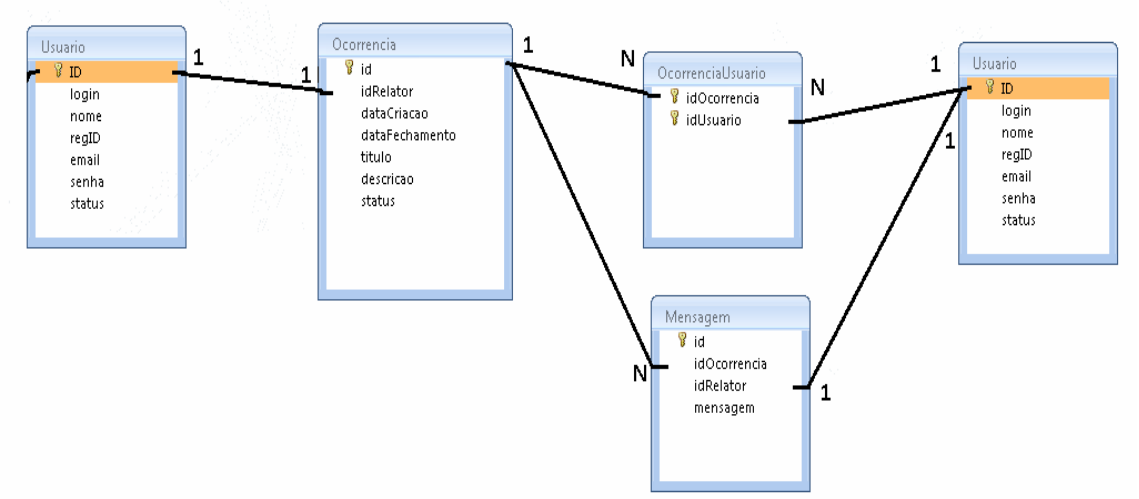
Figura 14 – Tela de Ocorrência

4.4 IMPLEMENTAÇÃO DO SISTEMA

A aplicação cliente e a aplicação servidor foram desenvolvidas em Java. O projeto foi foi dividido em dois pacotes principais br.com.brantur.jp e br.com.brantur.roogcm. Também foram identificadas as entidades para criação do banco de dados.

4.4.1 Modelo de entidade e relacionamento

Foram identificadas as entidades Ocorrência, Mensagem e Usuário no sistema. Para tornar a execução da aplicação servidor simples e sem a dependência de banco de dados, serão persistidos no servidor somente os dados dos usuários, não sendo armazenandas informações sobre as ocorrências e mensagens. A peristência no lado servidor se dá por meio de arquivo XML. Na aplicação serão persistidas as Ocorrências, Mensagens, OcorrenciasMensagens e Usuários sem os dados sigilosos como email e senha. A persistência no lado cliente se dará por meio do SQLite. A Figura 15 mostra o diagrama de entidades e relacionamento da aplicação que representa as tabelas do banco de dados da aplicação;



.Figura 15 – Diagrama de entidades e relacionamentos do banco de dados

No Quadro 5 estão os campos da tabela de Usuário. Um usuário pode relatar diversas ocorrências e colaborar enviando diversas mensagens em uma ocorrência relatada por outro usuário.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id	Integer	Não	Sim	Não	
login	Texto	Não	Não	Não	
nome	Texto	Não	Não	Não	
regID	Texto	Sim	Não	Não	Id de registro no servidor do GCM
email	Texto	Sim	Não	Não	
senha	Texto	Não	Não	Não	
status	Integer	Não	Não	Não	Status do usuario

Quadro 5 – Campos da tabela usuário

A tabela de ocorrência é apresentada no Quadro 6. Uma ocorrência possui um único relator e está vinculada a diversas mensagens de diversos usuários.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id	Integer	Não	Sim	Não	
idRelator	Integer	Não	Não	Não	Da tabela Usuário
dataCriacao	Texto	Não	Não	Não	
dataFechamento	Texto	Sim	Não	Não	
titulo	Texto	Não	Não	Não	
descricao	Texto	Não	Não	Não	
status	Integer	Não	Não	Não	Status da ocorrência

Quadro 2 – Campos da tabela ocorrência

O o Quadro 7 mostra os campos da tabela de mensagem. Uma mensagem possui um único relator e esta vinculada a uma única ocorrência.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id	Integer	Não	Sim	Não	
idRelator	Integer	Não	Não	Não	Da tabela Usuário
idOcorrencia	Integer	Não	Não	Não	Da tabela Ocorrência
mensagem	Texto	Não	Não	Não	

Quadro 7 – Campos da tabela mensagem

Uma ocorrência pode ter diversos usuários envolvidos. O Quadro 8 apresenta os campos da tabela que vincula ocorrências a usuarios.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
idOcorrencia	Numérico	Não	Sim	Sim	Da tabela Ocorrência
idUsuario	Numérico	Não	Sim	Sim	Da tabela Usuário

Quadro 3 – Campos da tabela aluno_curso

4.4.2 pacotes do projeto

Nas Figuras 16 e 17 é mostrada a estrutura do pacote *br.com.brantur.jp* da aplicação cliente e servidor, respectivamente. Esses pacotes contêm as classes responsáveis pela tecnologia envolvida no desenvolvimento do software. Nesse pacote estão classes genéricas de serialização/ deserialização de objetos, conexão com os serviços do Google, persistência de dados genérica. Essas classes e interfaces que não contêm informações que envolvem diretamente a regra de negócio do software, ou seja, são classes e interfaces que podem ser reutilizadas em diversas aplicações.

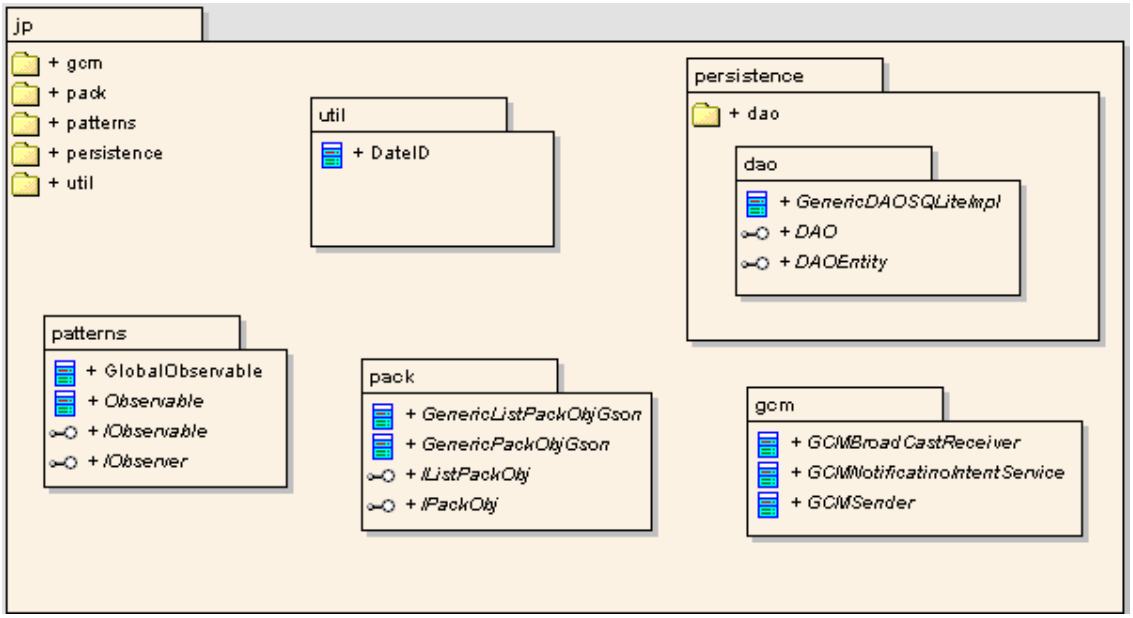


Figura 16 – Estrutura do pacote br.com.brantur.jp da aplicação cliente

Na Figura 17 está a estrutura do pacote **br.com.brantur.jp** da aplicação servidor.

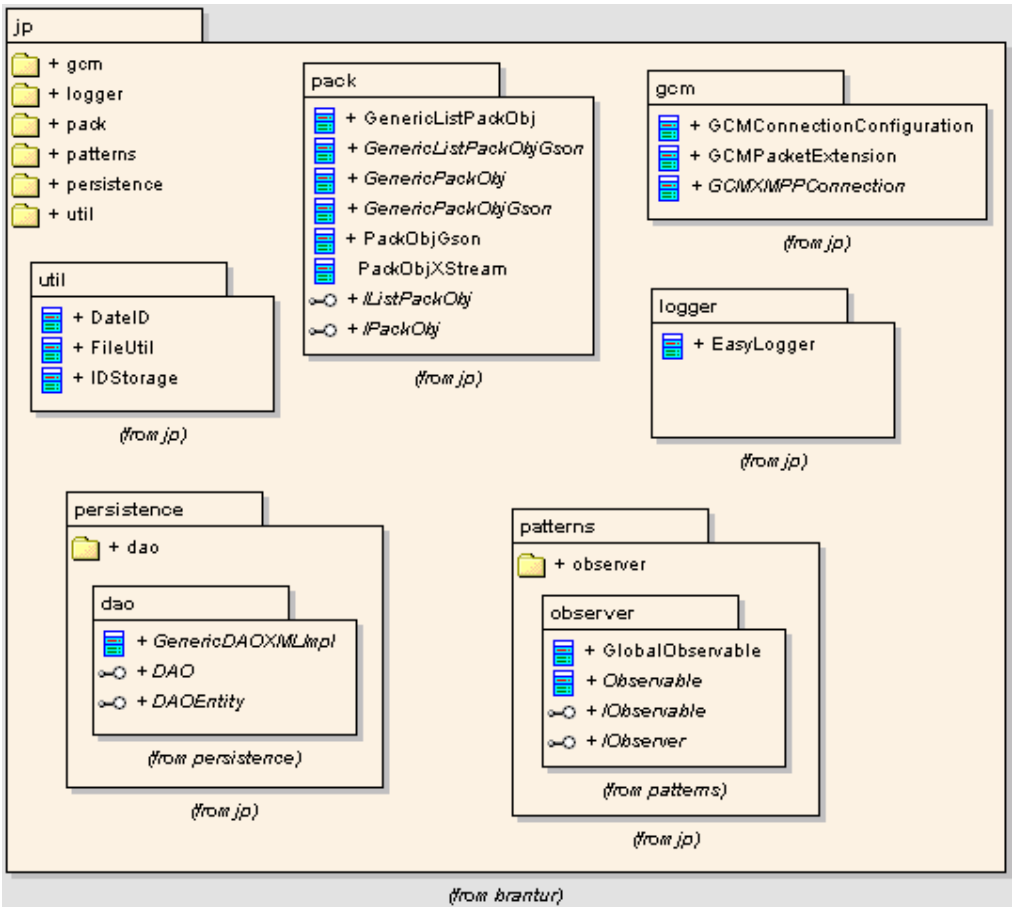


Figura 17 – Estrutura do pacote br.com.brantur.jp da aplicação servidor

A finalidade dos pacotes pertencentes ao pacote jp podem ser descritos como:

- br.com.brantur.jp.gcm: classes responsáveis pela conexão com o servidor do GCM, serviço de recebimento de mensagens e envio de mensagens;
- br.com.brantur.jp.dao: interface que deve ser implementada nas classes que representam os modelos da aplicação, interface a ser implementada nas classes DAO, classes genéricas para implementação dos DAOs.
- br.com.brantur.jp.patterns: classes e interfaces para implementação de design patterns;
- br.com.brantur.jp.logger: classes para log;
- br.com.brantur.jp.pack: classes genericas e interfaces utilizadas na serialização/serialização de objetos e lista de objetos;
- br.com.brantur.jp.util: classes de utilidades gerais;

Nos pacotes da aplicação cliente e servidor podem-se destacar as classes chaves para o desenvolvimento do software, que são as classes de conexão com o servidor do GCM e as classes de serialização genérica.

A Figura 18 e Figura 19 representa a estrutura do pacote **br.com.brantur.rrogcm** da aplicação cliente e servidor. Esses pacotes contêm as classes responsáveis pelos modelos, interfaces e regras de negócio do software. Nesse pacote estão as classes controladoras das interfaces, as classes de conexão e serviços de recebimento das mensagens do Google, já com as configurações dos registros da aplicacao, camada de persistência da aplicação, modelos dos objetos e adaptadores das listas das interfaces gráficas. Elas são, basicamente as classes e interfaces criadas para a implementação desta aplicação específica.

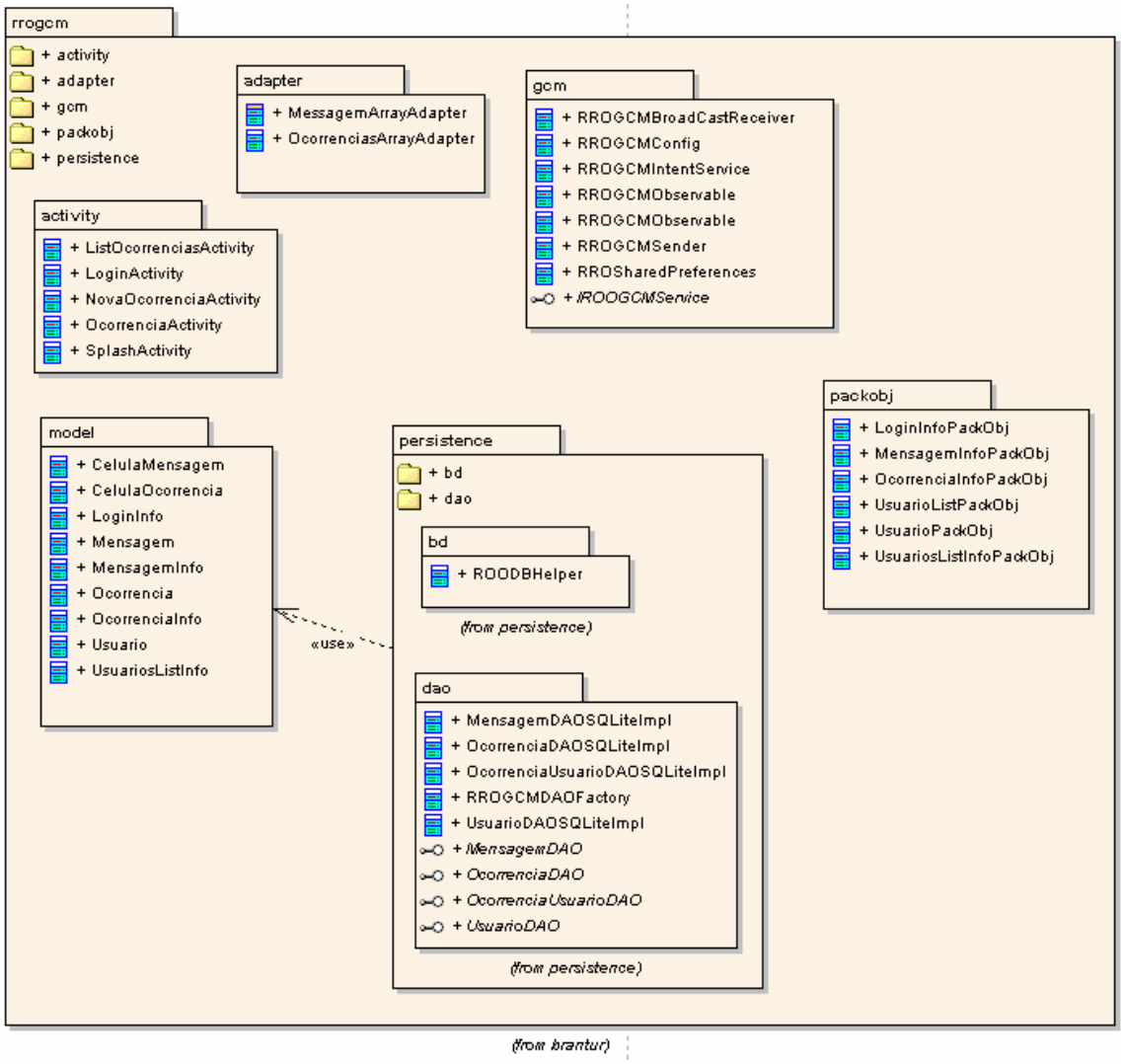


Figura 18 – Estrutura do pacote br.com.brantur.rrogcm da aplicação cliente

Na Figura 19 está representada a estrutura do pacote br.com.brantur.rrogcm da aplicação servidor.

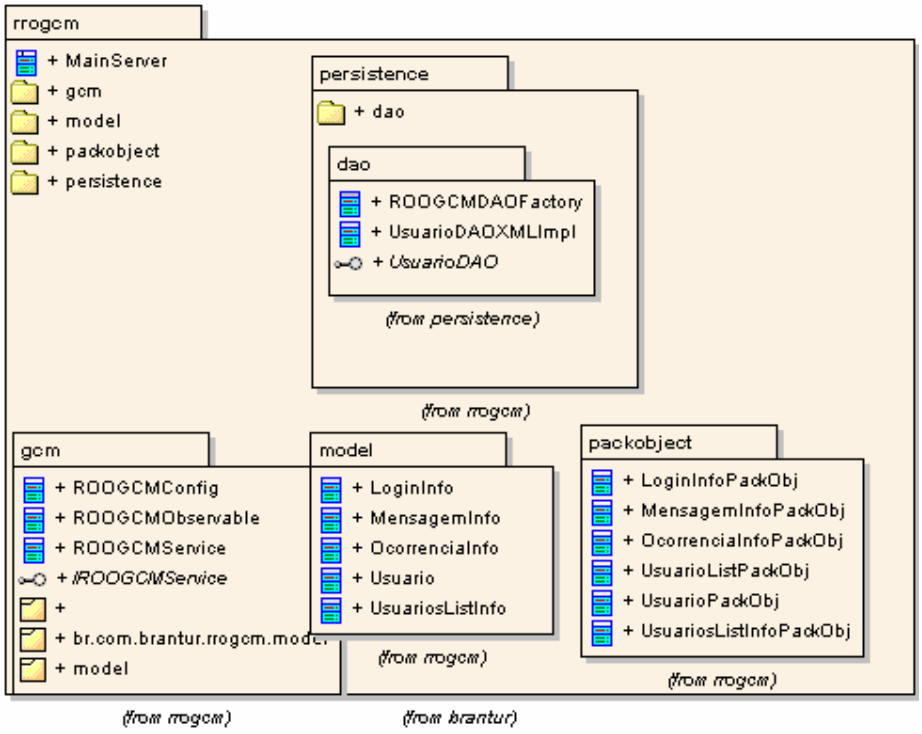


Figura 19 – Estrutura do pacote br.com.brantur.rrogcm da aplicação servidor

A finalidade dos pacotes pertencentes ao pacote **rrogcm** podem ser descritos como:

- **br.com.brantur.rrogcm.gcm**: classes responsáveis pela conexão com o servidor do GCM, serviço de recebimento de mensagens e envio de mensagens;
- **br.com.brantur.rrogcm.model**: classes que representam os modelos e entidades do sistema.
- **br.com.brantur.rrogcm.packobject**: classes responsáveis por serializar/ deserializar os objetos *pack* e lista de objetos pack do sistema;
- **br.com.brantur.rrogcm.persistence**: *factory* de DAOs que instancia o DAO no tipo de base desejada, interface dos DAOs das entidades do sistema, Implementação dos DAOs utilizados no sistema.

Nesse pacote destacam-se as classes de conexão com o servidor do Google que notificam o resto da aplicação ao receber mensagens. As classes de serialização/deserialização que empacotam os objetos do pacote model e enviam para o servidor.

4.4.3 Descrição das Classes

Devido à quantidade de classes, a descrição das classes será desmembrada em pacotes para melhor compreencão e apresentação e devido a similaridade de pacotes entre os códigos da aplicação cliente e servidor serão descritas as classes da aplicação servidor e cliente.

As Figuras 20 e 21 mostram o diagrama de classes do pacote **br.com.brantur.jp** da aplicação cliente e servidor. As classes do pacote jp, são as classes reutilizáveis do sistema e podem ser consideradas as ferramentas essenciais para a implementação das regras do negócio do sistema.

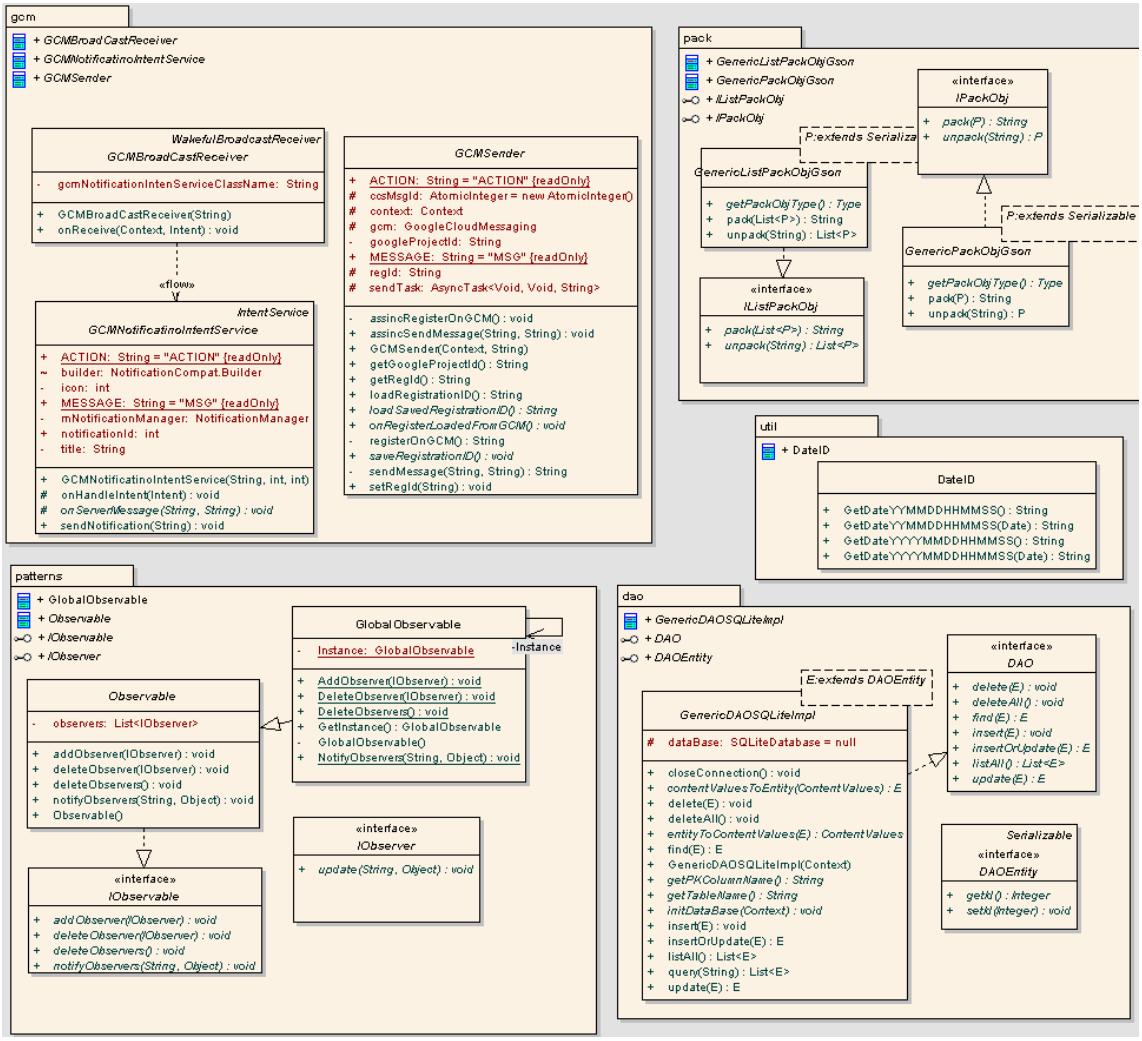


Figura 20 – Diagrama de classes do pacote br.com.brantur.jp da aplicação Cliente

Na Figura 21 está o diagrama de classes do pacote **br.com.brantur.jp** da aplicação servidor.

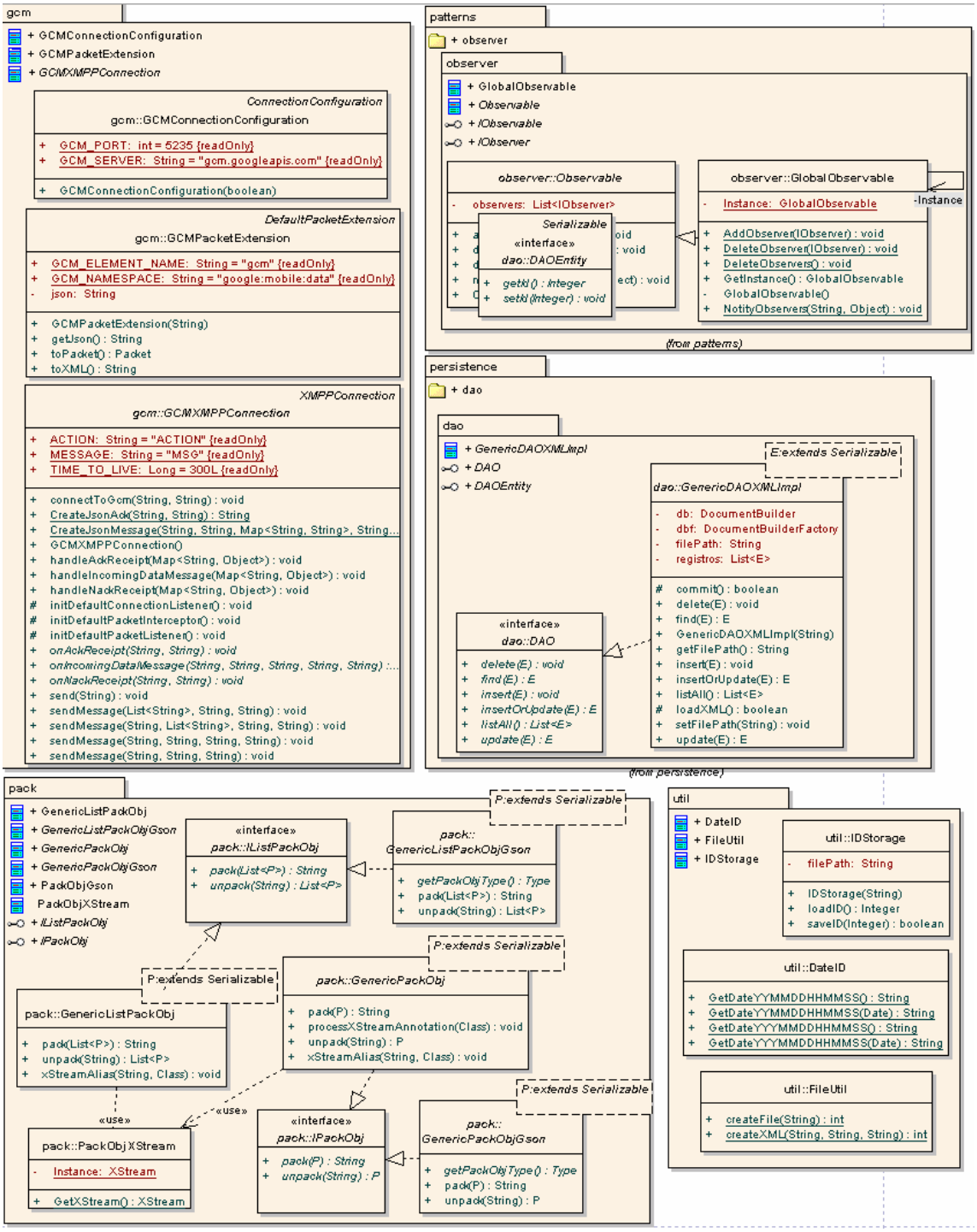


Figura 21 – Diagrama de classes do pacote br.com.brantur.jp da aplicação Servidor

As classes que representam os modelos de pacotes de informações, modelos que representam as entidades da base de dados e modelos de interfaces fazem parte do pacote **br.com.brantur.rrogcm.model** estão na Figura 19 para a aplicação cliente e na Figura 22 para a aplicação servidor.

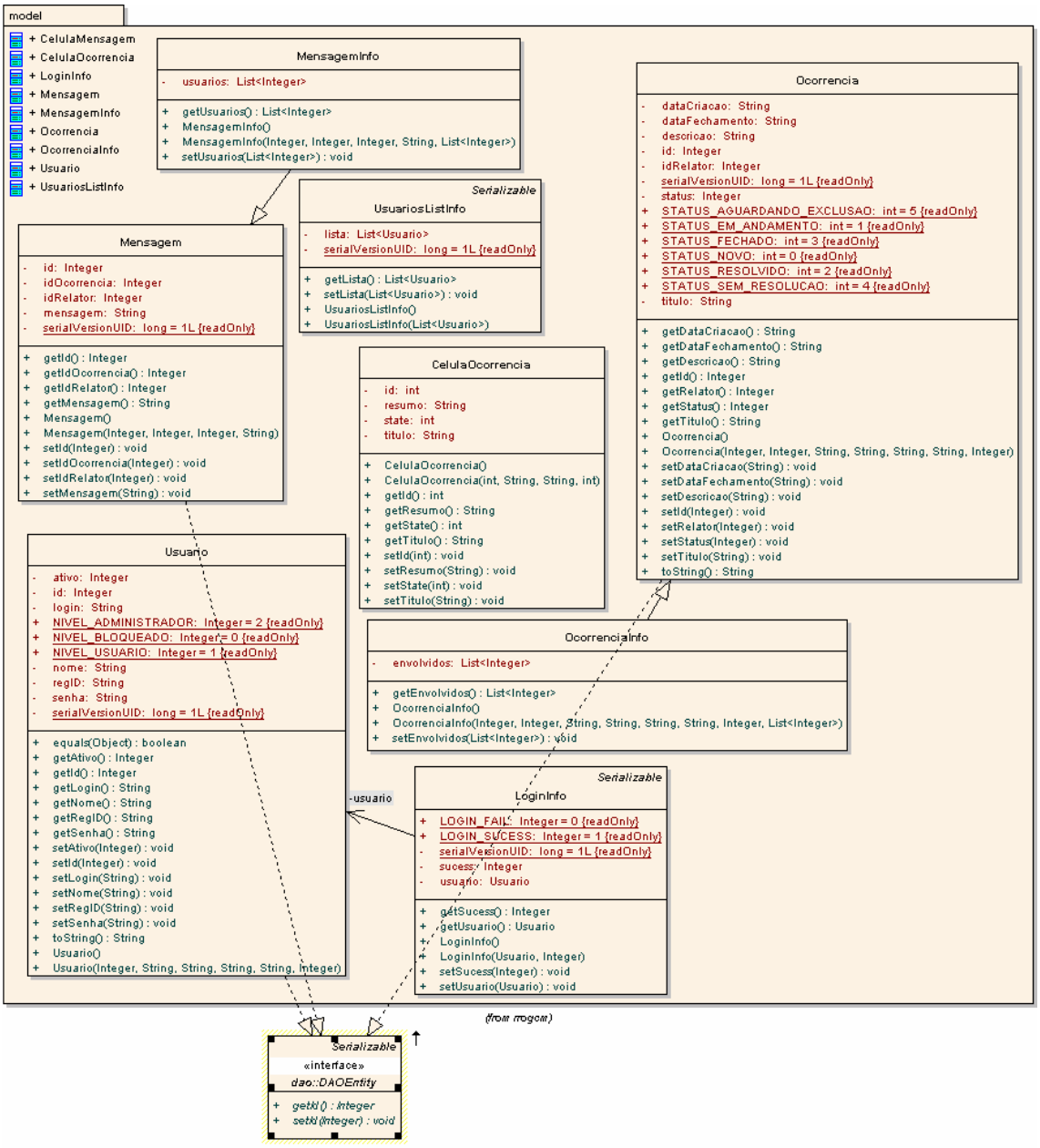


Figura 22 – Diagrama de Classes do pacote br.com.brantur.rrogcm.model da aplicação cliente

Figura 23 – Diagrama de Classes do pacote br.com.brantur.rrogcm.model da aplicação servidor

As mensagens entre cliente e servidor serão enviadas no formato JSON, nesse sentido foi criado o pacote **br.com.brantur.rrogcm.packobj** que é responsavel por

serializar/deserializar os objetos dos modelos de informação. As Figuras 24 e 25 mostram as classes da aplicação cliente e servidor respectivamente.

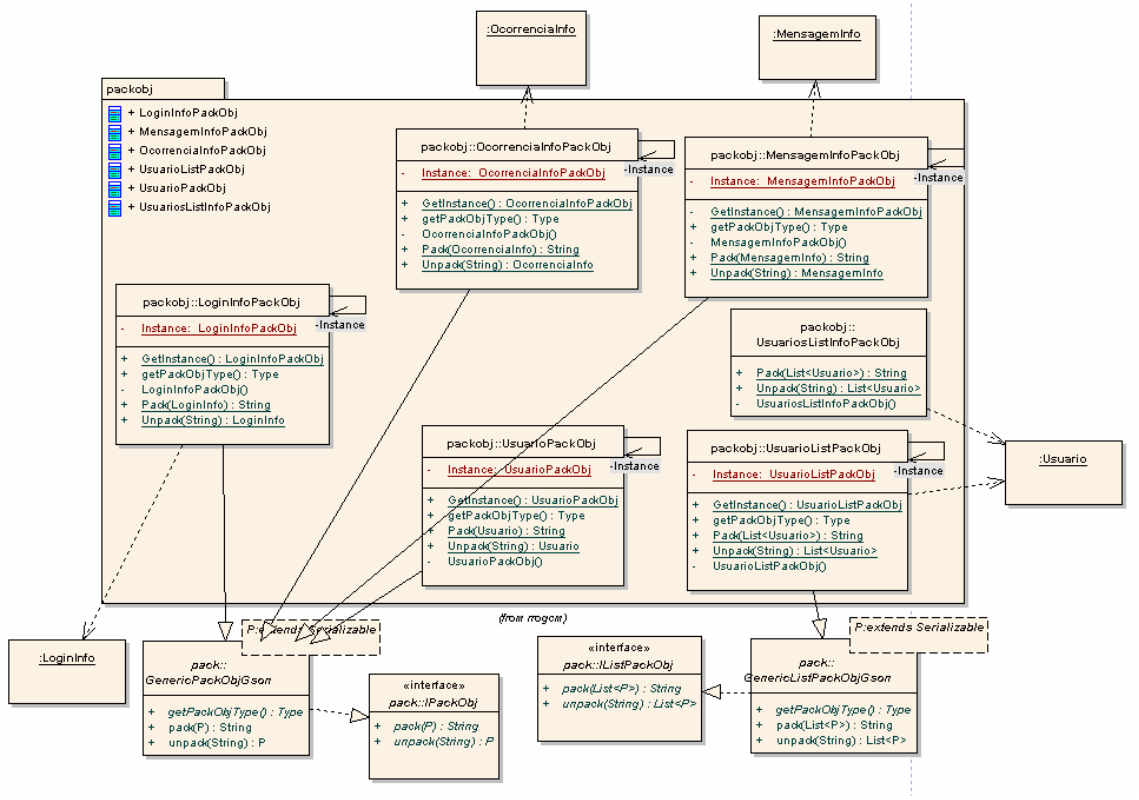


Figura 24 – Diagrama de Classes do pacote br.com.brantur.rrogcm.packobj da aplicação cliente

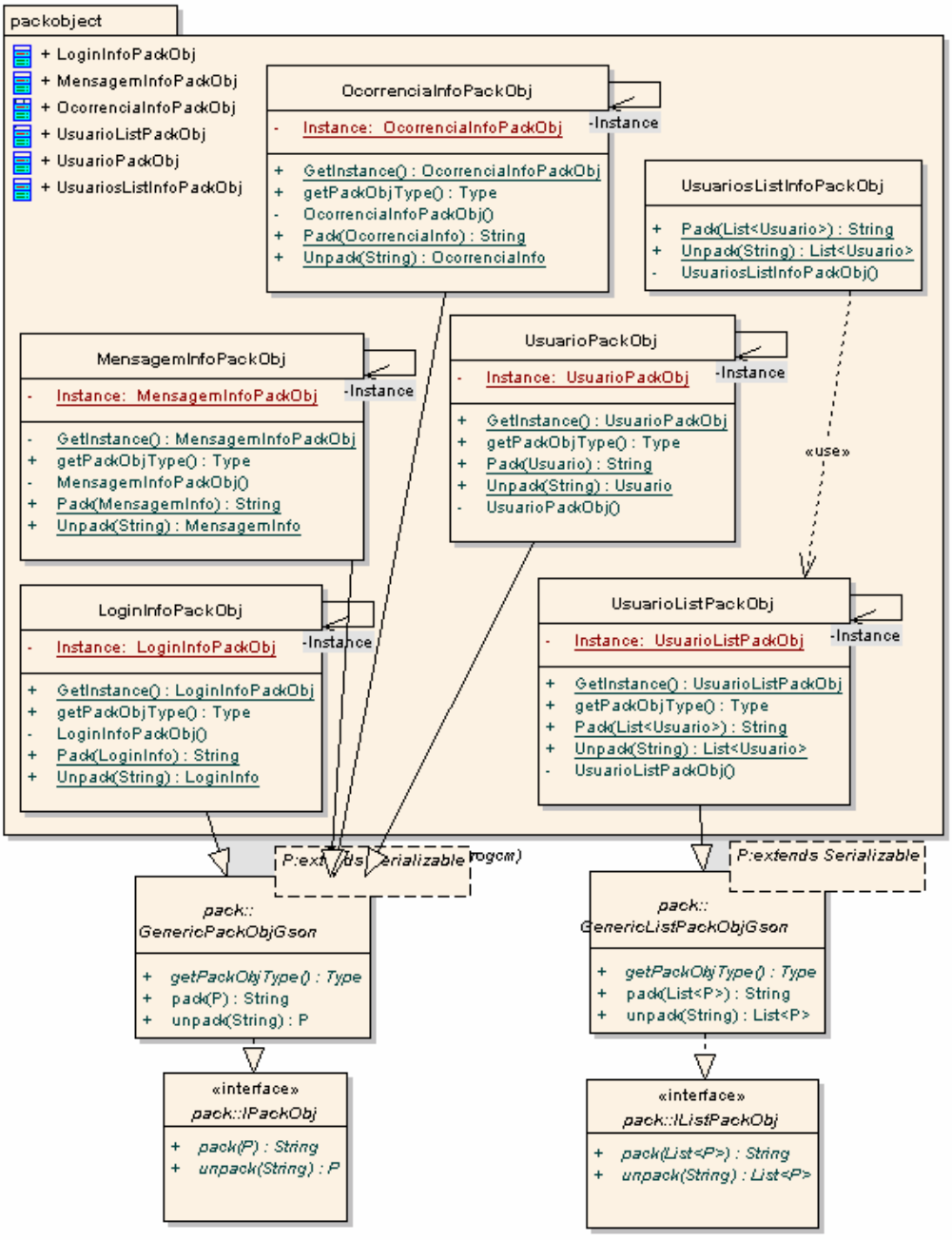


Figura 25 – Diagrama de Classes do pacote br.com.brantur.rrogcm.packobj da aplicação servidor

As classes responsáveis pela implementação da persistencia estão dentro do pacote **br.com.brantur.rrogcm.persistence** cujo diagrama de classes é mostrado na Figura 26 para a aplicação cliente e Figura 27 para a aplicação servidor.

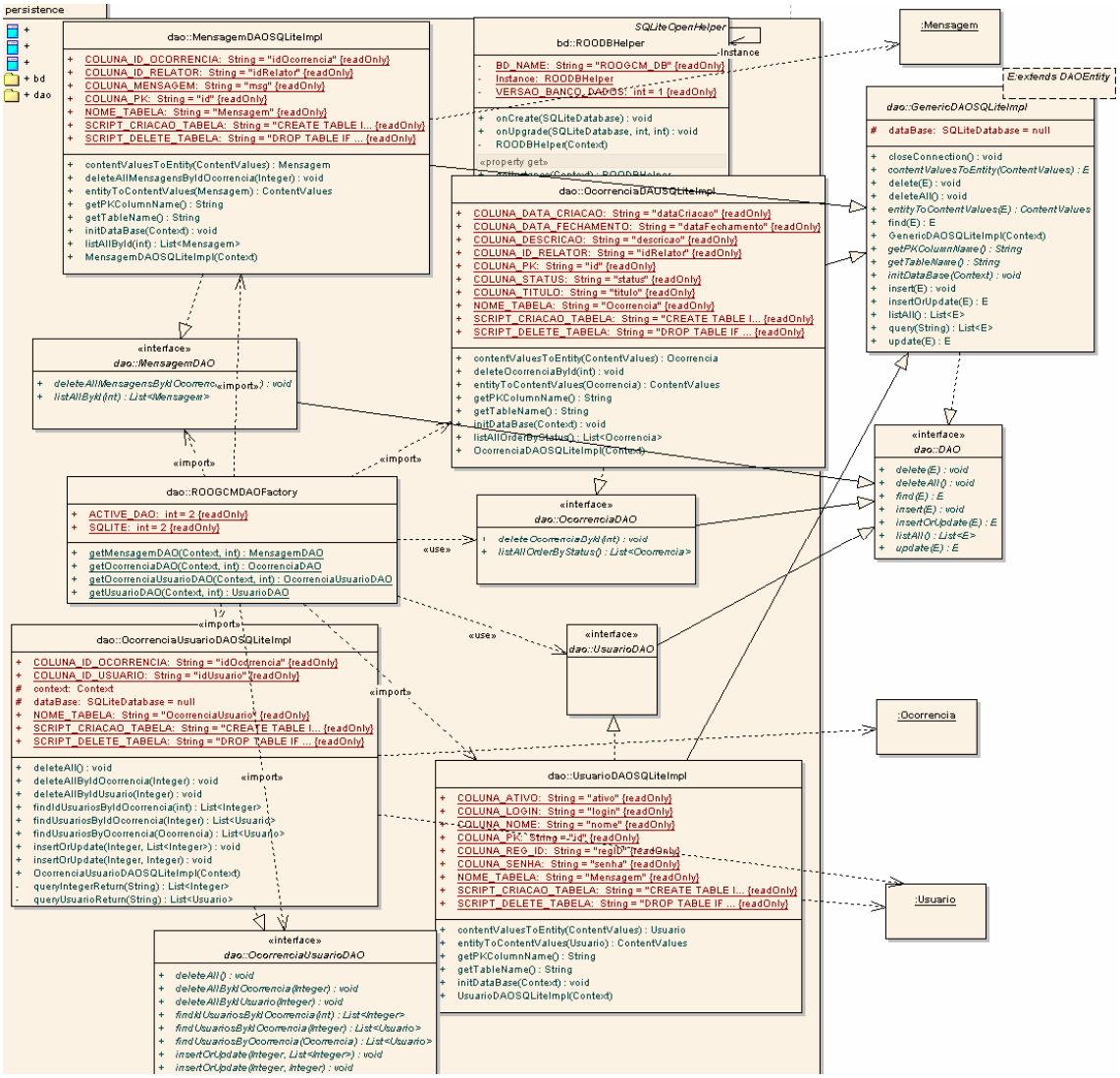


Figura 26 – Diagrama de classes do pacote br.com.brantur.rrogcm.persistence da aplicação cliente

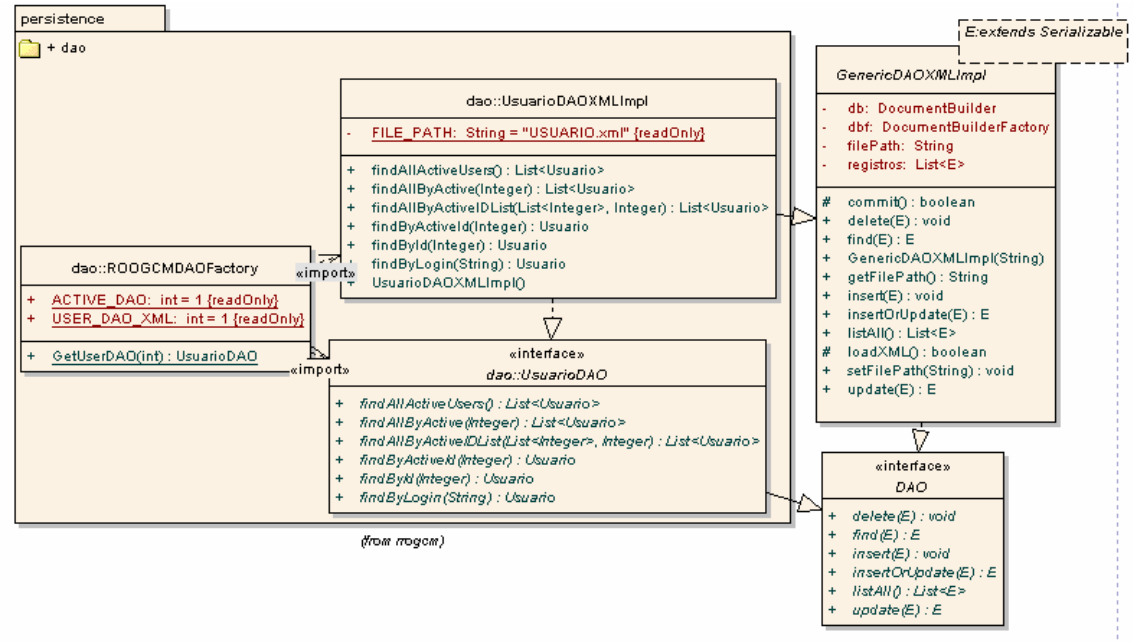


Figura 27 – Diagrama de classes do pacote br.com.brantur.rrogcm.persistence da aplicação servidor

As classes responsáveis por receber, enviar, notificar outros objetos e constantes de configuração da conexão estão dentro do pacote **br.com.brantur.rrogcm.gcm** mostrado na Figura 28 e Figura 29 na aplicação cliente e servidor respectivamente. Para não tornar o diagrama muito extenso graficamente foram colocadas relações diretamente com alguns pacotes para condensar o espaço do diagrama.

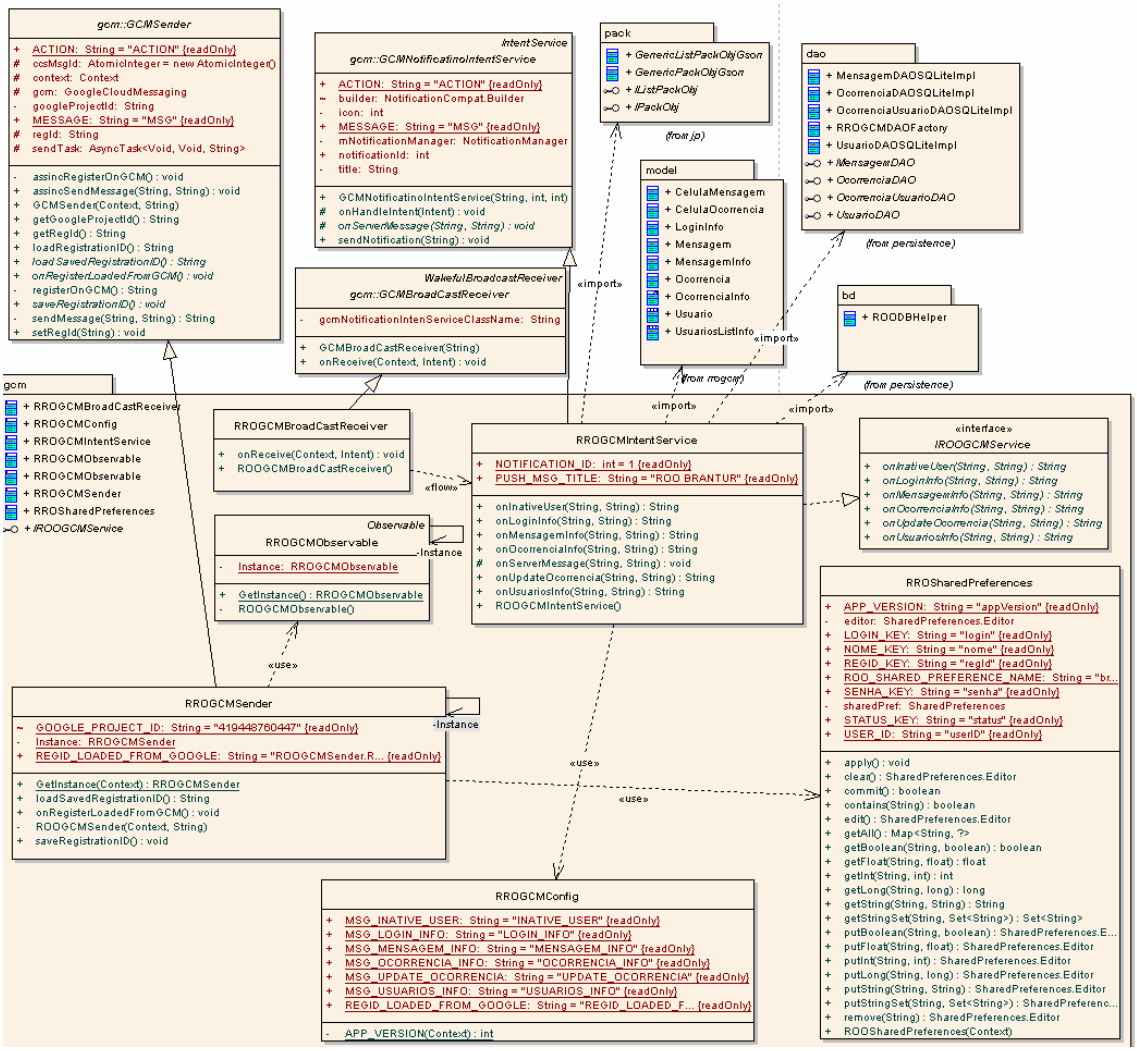


Figura 28 – Diagrama de classes do pacote br.com.brantur.rrogcm.gcm da aplicação cliente

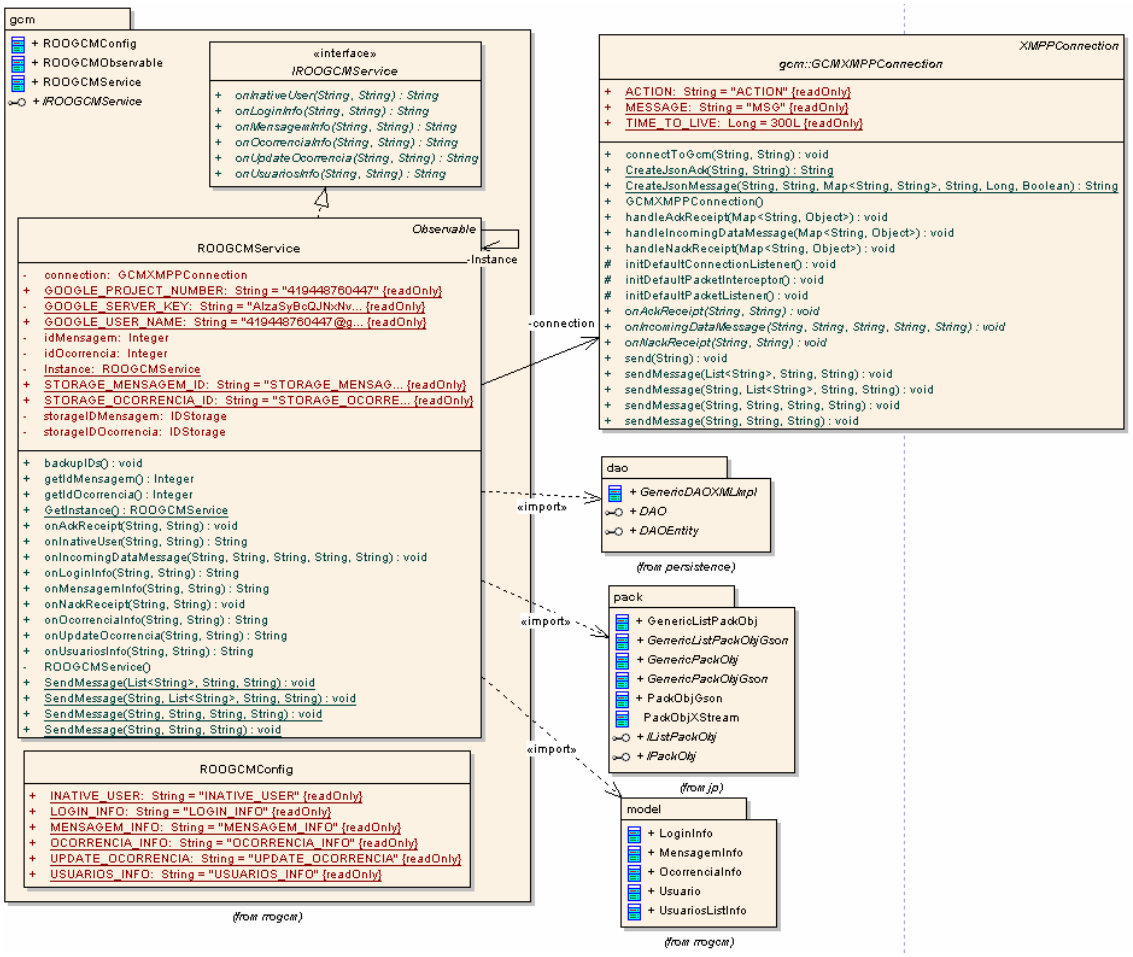


Figura 29 – Diagrama de classes do pacote br.com.brantur.rrogcm.gcm da aplicação servidor

As classes responsáveis por controlar as interfaces estão no pacote **br.com.brantur.rrogcm.activity.** Este pacote existe somente na aplicação servidor e é mostrado na Figura 30.

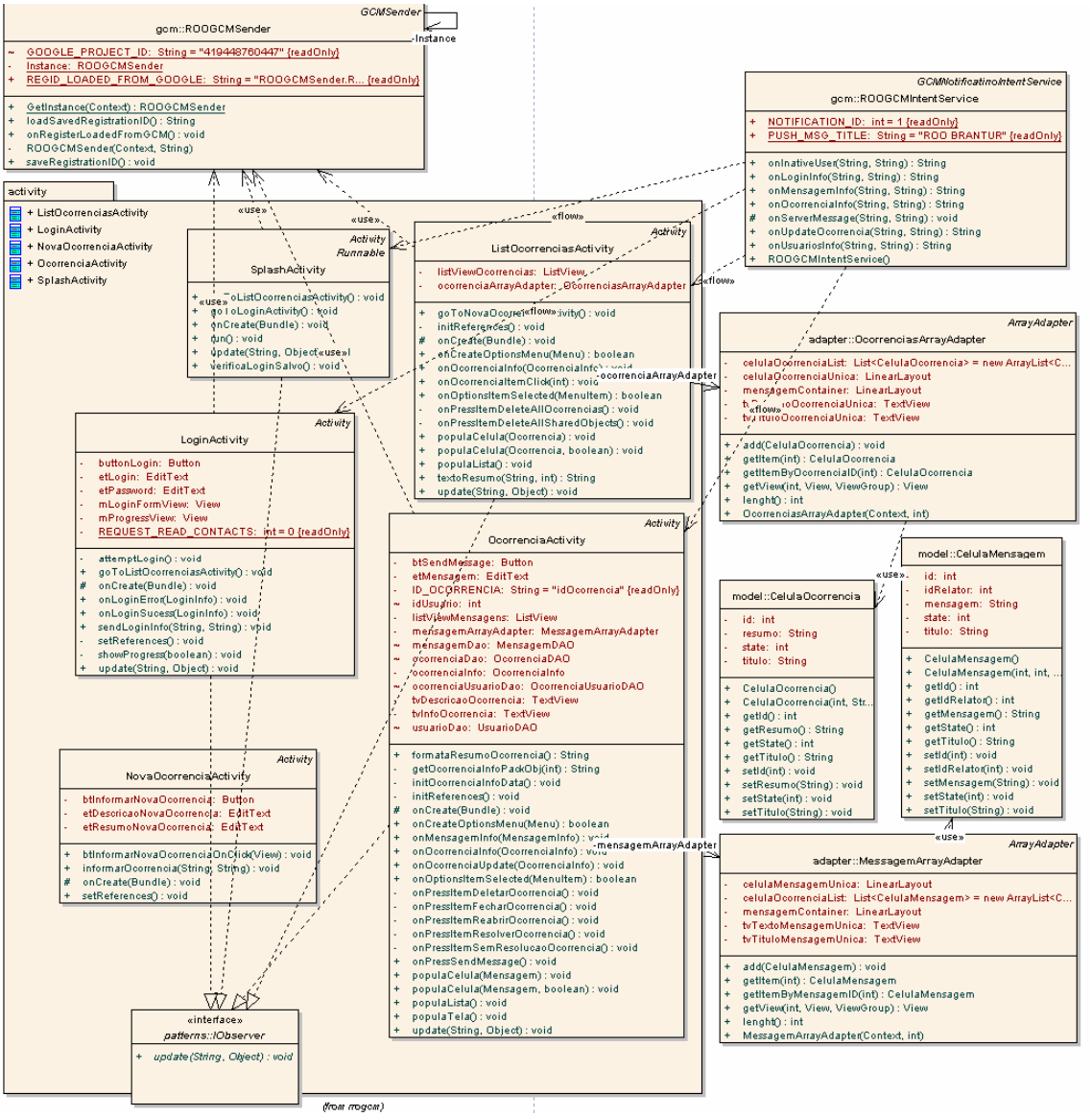


Figura 30 – Diagrama de classes do pacote br.com.brantur.rrogcm.activity da aplicação cliente

4.4.4 Principais códigos

Nesta subseção serão comentados os principais trechos códigos das classes que possuem as funcionalidades principais na implementação do sistema. Os códigos comentados serão as classes de conexão com o servidor do GCM, serializadores genéricos dos objetos e DAOs genéricos com implementação em XML.

Na aplicação cliente e servidor as mensagens contendo as informações são feitas por meio de objetos de informação, na qual cada objeto de informação possui suas características. Para cada classe de modelo de informação é criado uma classe que extende

GenericPackObjGson que é responsável por serializar/deserializar o objeto de informação. A Classe abstrata GenericPackObjGson pode ser visualizada na Listagem 1. Na implementação da Classe GenericPackObjGson a serialização/deserialização é efetuada pela utilização da biblioteca Gson que serializa/deserializa objetos no formato JSON. Para poder ser implantada genericamente o método para deserializar, foi necessário criar o método abstrato getPackObjType que deve ser implementado na classe concreta e deve retornar o tipo do objeto a ser serializado.

```
package br.com.brantur.jp.pack;

import java.io.Serializable;
import java.lang.reflect.Type;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

public abstract class GenericPackObjGson<P extends Serializable> implements IPackObj<P> {

	/**
	 * converte  o objeto generico em uma string JSON
	 */
	@Override
	public String pack(P jsonObj) {
		Gson gson = new Gson();
		TypeToken<P> typeToken = new TypeToken<P>() {
		};

		return gson.toJson(jsonObj, typeToken.getType());
	}

	/**
	 * converte a string JSON no objeto Generico
	 */
	@Override
	public P unpack(String strPackObj) {
		Gson gson = new Gson();

	   @SuppressWarnings("unchecked")
		P p = (P)gson.fromJson(strPackObj, this.getPackObjType());

	   return p;
	}

	/**
	 * metodo abstrato que deve tetornar o tipo do objeto a ser serializado/deserializado
	 * @return
	 */
	public abstract Type getPackObjType();
}
```

Listagem 1 – Classe abstrata GenericPackObj

Na aplicação servidor, a tabela de usuários é persistida em XML, a classe responsável por persistir objetos em formato XML é a classe abstrata GenericDAOXMLImpl.

Por meio dela é possível converter objetos ou lista de objetos em arquivos XML. A chave para a implementação desse DAO genérico no formato XML é a utilização da biblioteca XStream que serializa/deserializa objetos no formato XML ou JSON. A Listagem 2 mostra os métodos da classe GenericDAOXMLImpl mais relevantes, o método loadXML() é responsável por carregar o XML, deserializar em uma lista de objetos, o método commit é responsável por gravar a lista de objetos no formato XML. Todas as atualizações, inserções, excluões são finalizadas com o método commit(). Para serializar/deserializar os objetos no formato XML foi utilizado um objeto XStream pertencente ao pacote **com.thoughtworks.xstream.**

```
package br.com.brantur.jp.persistence.dao;

import ....

/**
 * DAO generico implementado em xml
 * @author jp
 *
 * @param <E>
 */
public abstract class GenericDAOXMLImpl<E extends Serializable> implements DAO<E> {
	....


	/**
	 * carrega o arquivo xml e popula a lista de objetos
	 * @return true caso arquivo carregado
	 */
	@SuppressWarnings("unchecked")
	protected boolean loadXML() {

		try {
			//novo arquivo
			File file = new File(this.filePath);
			//verifica se é diretorio
			if (file.isDirectory()) {
				return false;//se sim retorna false
			} else if (file.exists()) {
				//tenta efetuar o parse do arquivo
				Document doc = this.db.parse(file);
				// Serialize DOM
				OutputFormat format = new OutputFormat(doc);

				// stringWriteras a String
				StringWriter stringOut = new StringWriter();
				XMLSerializer serial = new XMLSerializer(stringOut, format);

				serial.serialize(doc);

				//instancia do XStream utilizado para serializar/deserializar
				//objetos é setado como xml a serialização
				XStream x = new XStream(new DomDriver("UTF-8"));
				//converte o xml em uma lista de objetos
				this.registros = (List<E>) x.fromXML(stringOut.toString());

				return true;
			} else {
				return false;
			}
		} catch (IOException ioe) {
			ioe.printStackTrace();
			return false;
		}catch(SAXException se){
			se.printStackTrace();
			return false;
```

```
		}
	}

	/**
	 * salva as alteraçoes efetuadas na lista de objetos no arquivo XML
	 * @return
	 */
	protected boolean commit() {
		FileOutputStream fos = null;
		try {
			//novo arquivo
			File file = new File(this.filePath);
			//verifica se é diretorio
			if (file.isDirectory()) {
				return false;//false se for diretorio
			} else if (file.exists()) {//se arquivo existe
				//Objeto XStream para serializacao em xml
				XStream x = new XStream(new DomDriver("UTF-8"));
				//instancia fos
				fos = new FileOutputStream(this.filePath);
				//serializa a lista de objetos
				x.toXML(this.registros, fos);
				return true;
			} else {
				return false;
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			return false;
		} finally {
			try {
				if (fos != null) {
					fos.close();
				}
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

	...implementações do DAO
```

Listagem 2 – Classe abstrata GenericDAOXMLImpl

Na aplicação servidor a comunicação com o servidor do GCM é feita pela classe ROOGCMService que extende GCMXMPPConnection mostrada na listagem Listagem 3. A classe GCMXMPPConnection é responsável pelo serviço de comunicação com o serviço XMPP do CCS. Ela é implementada utilizando a biblioteca org.jivesoftware.smack, ela proporciona comunicação assincrona com o CCS.

```
package br.com.brantur.jp.gcm;

import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.jivesoftware.smack.ConnectionListener;
import org.jivesoftware.smack.PacketInterceptor;
import org.jivesoftware.smack.PacketListener;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smack.filter.PacketTypeFilter;
import org.jivesoftware.smack.packet.Message;
import org.jivesoftware.smack.packet.Packet;
import org.jivesoftware.smack.packet.PacketExtension;
import org.jivesoftware.smack.provider.PacketExtensionProvider;
import org.jivesoftware.smack.provider.ProviderManager;
import org.json.simple.JSONValue;
```

```
import org.json.simple.parser.ParseException;
import org.xmlpull.v1.XmlPullParser;

import br.com.brantur.jp.logger.EasyLogger;

/**
 * efetua a conexao com o servidor do GCM
 * @author jp
 *
 */
public abstract class GCMXMPPConnection extends XMPPConnection {

        /**
         * string mapeador das acoes
         */
    public static final String ACTION = "ACTION";
    /**
     * string mapeadora da mensagem
     */
    public static final String MESSAGE = "MSG";
    /**
     * tempo de vida das mensagens no servidor(máx 4 semanas)
     */
    public static final Long TIME_TO_LIVE = 300L;//2419200L;

    /**
     * construtor padrao
     */
       public GCMXMPPConnection() {
               super(new GCMConnectionConfiguration(true));
               // Add GcmPacketExtension
               ProviderManager.getInstance().addExtensionProvider(
                               GCMPacketExtension.GCM_ELEMENT_NAME,
                               GCMPacketExtension.GCM_NAMESPACE,
                               new PacketExtensionProvider() {

                                       @Override
                                       public PacketExtension parseExtension(XmlPullParser
parser)
                                                       throws Exception {
                                               String json = parser.nextText();
                                               GCMPacketExtension packet = new
GCMPacketExtension(json);
                                               return packet;
                                       }
                               });

               // TODO Auto-generated constructor stub

       }

       /**
        * conecta no google cloud utilizando as credenciais passadas no construtor
        *
        * @param username
        *            GCM_SENDER_ID@gcm.googleapis.com
        * @param password
        *            API Key
        * @throws XMPPException
        */
       public void connectToGcm(String username, String password) {

               // -Dsmack.debugEnabled=true
               XMPPConnection.DEBUG_ENABLED = true;

               try {
                       this.connect();
                       this.initDefaultConnectionListener();
                       this.initDefaultPacketListener();
                       this.initDefaultPacketInterceptor();
                       this.login(username, password);
               } catch (XMPPException e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();
               }
       }
```

```
        /**
         * inicia o listener padrao da conexao
         */
        protected void initDefaultConnectionListener() {

                this.addConnectionListener(new ConnectionListener() {

                                                ....
                                                ....
                });
        }

        /**
         * adiciona o Listener de pacotes
         */
        protected void initDefaultPacketListener() {

                // Handle incoming packets
                this.addPacketListener(new PacketListener() {

                        @Override
                        public void processPacket(Packet packet) {
                                // logger.log(Level.INFO, "Received: " + packet.toXML());
                                Message incomingMessage = (Message) packet;
                                GCMPacketExtension gcmPacket = (GCMPacketExtension)
incomingMessage
                                                .getExtension(GCMPacketExtension.GCM_NAMESPACE);
                                String json = gcmPacket.getJson();
                                try {
                                        @SuppressWarnings("unchecked")
                                        Map<String, Object> jsonObject = (Map<String, Object>)
JSONValue
                                                        .parseWithException(json);

                                        // present for "ack"/"nack", null otherwise
                                        Object messageType = jsonObject.get("message_type");

                                        if (messageType == null) {
                                                // Normal upstream data message
                                                GCMXMPPConnection.this

        .handleIncomingDataMessage(jsonObject);

                                                // Send ACK to CCS
                                                String messageId = jsonObject.get("message_id")
                                                                .toString();
                                                String from = jsonObject.get("from").toString();
                                                String ack = CreateJsonAck(from, messageId);
                                                send(ack);
                                        } else if ("ack".equals(messageType.toString())) {
                                                // Process Ack
                                                handleAckReceipt(jsonObject);
                                        } else if ("nack".equals(messageType.toString())) {
                                                // Process Nack
                                                handleNackReceipt(jsonObject);
                                        } else {

                                        }
                                } catch (ParseException e) {


                                } catch (Exception e) {

                                }
                        }
                }, new PacketTypeFilter(Message.class));
        }

        /**
         * inicia o Interceptador de Pacotes
         */
        protected void initDefaultPacketInterceptor() {

                // Log all outgoing packets
                this.addPacketInterceptor(new PacketInterceptor() {
                        @Override
                        public void interceptPacket(Packet packet) {
```

```
                }
        }, new PacketTypeFilter(Message.class));
}

/**
 * Sends a downstream GCM message.
 */
public void send(String jsonRequest) {
        Packet request = new GCMPacketExtension(jsonRequest).toPacket();
        this.sendPacket(request);
}

/**
 * Handles an upstream data message from a device application.
 *
 * <p>
 * This sample echo server sends an echo message back to the device.
 * Subclasses should override this method to process an upstream message.
 */
public void handleIncomingDataMessage(Map<String, Object> jsonObject) {

        String messageId = jsonObject.get("message_id").toString();

        String from = jsonObject.get("from").toString();

        // PackageName of the application that sent this message.
        String category = jsonObject.get("category").toString();

        @SuppressWarnings("unchecked")
        Map<String, String> payload = (Map<String, String>) jsonObject
                        .get("data");

        String action = payload.get(ACTION);
        String msgData = payload.get(MESSAGE);

        this.onIncomingDataMessage(messageId, from, category, action, msgData);

}

public void handleAckReceipt(Map<String, Object> jsonObject) {
        String messageId = jsonObject.get("message_id").toString();
        String from = jsonObject.get("from").toString();

        onAckReceipt(messageId, from);
}

public void handleNackReceipt(Map<String, Object> jsonObject) {
        String messageId = jsonObject.get("message_id").toString();
        String from = jsonObject.get("from").toString();
        EasyLogger.Log("handleNackReceipt", "Nack from: " + from
                        + " messageId: " + messageId);
        this.onNackReceipt(messageId, from);
}

/**
 * Creates a JSON encoded ACK message for an upstream message received from
 * an application.
 *
 * @param to
 *            RegistrationId of the device who sent the upstream message.
 * @param messageId
 *            messageId of the upstream message to be acknowledged to CCS.
 * @return JSON encoded ack.
 */
public static String CreateJsonAck(String to, String messageId) {
        Map<String, Object> message = new HashMap<String, Object>();
        message.put("message_type", "ack");
        message.put("to", to);
        message.put("message_id", messageId);
        return JSONValue.toJSONString(message);
}

/**
 * Creates a JSON encoded GCM message.
 *
 * @param to
```

```
         *             RegistrationId of the target device (Required).
         * @param messageId
         *             Unique messageId for which CCS will send an "ack/nack"
         *             (Required).
         * @param payload
         *             Message content intended for the application. (Optional).
         * @param collapseKey
         *             GCM collapse_key parameter (Optional).
         * @param timeToLive
         *             GCM time_to_live parameter (Optional).
         * @param delayWhileIdle
         *             GCM delay_while_idle parameter (Optional).
         * @return JSON encoded GCM message.
         */
        public static String CreateJsonMessage(String to, String messageId,
                        Map<String, String> payload, String collapseKey, Long timeToLive,
                        Boolean delayWhileIdle) {
                Map<String, Object> message = new HashMap<String, Object>();
                message.put("to", to);
                if (collapseKey != null) {
                        message.put("collapse_key", collapseKey);
                }
                if (timeToLive != null) {
                        message.put("time_to_live", timeToLive);
                }
                if (delayWhileIdle != null && delayWhileIdle) {
                        message.put("delay_while_idle", true);
                }
                message.put("message_id", messageId);
                message.put("data", payload);
                return JSONValue.toJSONString(message);
        }

        /**
         * envia uma mensagem para uma lista de regIds
         */
        public void sendMessage(List<String> toDeviceRegIdList, String action, String message)
{

                String messageId = String.valueOf(new Date().getTime());
                for (String regID : toDeviceRegIdList) {
                        this.sendMessage(messageId, regID, action, message);
                }

        }

        /**
         * envia uma mensagem para uma lista de regids
         */
        public void sendMessage(String messageId, List<String> toDeviceRegIdList, String
action, String message) {
                this.sendMessage(messageId,toDeviceRegIdList,action,message);
        }
        /**
         * envia uma mensagem
         */
        public void sendMessage(String messageId, String toDeviceRegId, String action, String
message) {
                Map<String, String> payload = new HashMap<String, String>();
                payload.put(GCMXMPPConnection.ACTION, action);
                payload.put(GCMXMPPConnection.MESSAGE, message);
                payload.put("EmbeddedMessageId", messageId);
                String collapseKey = "datamsgcollapsedkey";
                Long timeToLive = TIME_TO_LIVE;
                Boolean delayWhileIdle = true;
                this.send(GCMXMPPConnection.CreateJsonMessage(toDeviceRegId, messageId,
payload,
                                collapseKey, timeToLive, delayWhileIdle));
        }
        /**
         * envia uma mensagem
         */
        public void sendMessage( String toDeviceRegId, String action, String message) {
                String messageId = String.valueOf(new Date().getTime());
                this.sendMessage(messageId, toDeviceRegId, action, message);
        }
        /**
```

```
         * metodo executado no sucesso do envio de uma mensagem
         */
        public abstract void onAckReceipt(String messageId, String from);

        /**
         * metodo executado ao falhar o recebimento de uma mensagme
         */
        public abstract void onNackReceipt(String messageId, String from);

        /**
         * metodo executado ao receber uma mensagem
         */
        public abstract void onIncomingDataMessage(String messageId, String from, String
category, String action, String msg);
}
```

Listagem 3 – Classe abstrata GCMXMPPConnection

Na aplicação cliente, as classes responsáveis pela comunicação com o servidor do GCM são ROOGCMBroadCastReceiver, ROOGCMIntentService e RROGCMSender. Para configurar o serviço de recebimento de notificações o arquivo manifest deve conter as referências para o ROOGCMBroadCastReceiver e o ROOGCMIntentService conforme mostra a Listagem 4.

```
......
......

<receiver
    android:name=".roogcm.gcm.RROGCMBroadCastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />

        <category android:name="com.javapapers.android" />
    </intent-filter>
</receiver>

<service android:name=".roogcm.gcm.RROGCMIntentService" />

......
......
```

Listagem 4 – Trecho onde é configurado o receiver e o intent service no arquivo AndroidManifest.xml

A Classe ROOGCMService herda de GCMSender mostrada na Listagem 5 é a classe abstrata que possui implementados métodos de conexão, registro e envio de mensagens no GCM, responsável por enviar menssagens para o servidor do GCM. Ela instancia um objeto GoogleCloudMessaging e efetua o pedido de registro, ao receber o registro chama o método abstrato saveRegistrationID (método que deve salvar o registration ID) e onRegisterLoadFromGCM (método que deve notificar a aplicação que o registro foi obtido), que são implementados na classe ROOGCMSender.

```
package br.com.brantur.jp.gcm;

import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;

import com.google.android.gms.gcm.GoogleCloudMessaging;

import java.io.IOException;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Efetua conexao com o GCM, e envia mensagens upstream
 * Created by jp on 18/10/2015.
 */
public abstract class GCMSender{

    /**
     * acao mapeada nas strings enviadas
     */
    public static final String ACTION = "ACTION";
    /**
     * mensagem mapeada nas strings enviadas
     */
    public static final String MESSAGE = "MSG";

    protected Context context;

    /**
     * conexao com o GCM
     */
    protected GoogleCloudMessaging gcm;
    /**
     * variavel que armazenara o ID de registro
     */
    protected String regId;

    /**
     * tarefa assyncrona
     */
    protected AsyncTask<Void, Void, String> sendTask;
    /**
     * gera id das mensagens
     */
    protected AtomicInteger ccsMsgId = new AtomicInteger();

    /**
     * id do projeto do google
     */
    private String googleProjectId;

    /**
     * construtor padrao
     * @param context
     * @param googleProjectId id do projeto do google
     */
    public GCMSender(Context context, String googleProjectId){
        this.context = context;
        this.googleProjectId = googleProjectId;
    }

    /**
     * Envia mensagem
     * @param action tipo de requisicao
     * @param msg packobj con os dados da requisicao
     */
    private String sendMessage(final String action, final String msg) {
```

```
                Bundle data = new Bundle();
                data.putString(ACTION, action);
                data.putString(MESSAGE, msg);
                String id = Integer.toString(ccsMsgId.incrementAndGet());

                try {
                    gcm.send(GCMSender.this.getGoogleProjectId() +
"@gcm.googleapis.com", id,
                            data);
                    Log.d("GCMClient", "sendMessage - ACTION: " + action + " - MSG:
" + msg);
                } catch (IOException e) {
                    e.printStackTrace();
                    return "error";
                }
                return "sucess";
            }

    /**
     * Envia mensagem assincrona
     * @param action tipo de requisicao
     * @param msg packobj con os dados da requisicao
     */
    public void assincSendMessage(final String action, final String msg) {

        sendTask = new AsyncTask<Void, Void, String>() {
            @Override
            protected String doInBackground(Void... params) {
                return GCMSender.this.sendMessage(action,msg);
            }
            @Override
            protected void onPostExecute(String result) {
                sendTask = null;
            }
        };
        sendTask.execute(null, null, null);
    }

    /**
     * carrega o registration id, salvo ou do servidor do google
     * @return
     */
    public String loadRegistrationID() {

        this.loadSavedRegistrationID();

        if (this.regId.equals("")) {
            this.assincRegisterOnGCM();
        }

        return this.regId;
    }

    /**
     * faz o pedido de registro assincrono
     */
    private void assincRegisterOnGCM() {
        new AsyncTask<Void, Void, String>() {
            @Override
            protected String doInBackground(Void... params) {
                return GCMSender.this.registerOnGCM();
            }

            @Override
            protected void onPostExecute(String msg) {
                Log.d("GCMClient", "Registrado no servidor do Google ");
            }
```

```
        }.execute(null, null, null);
    }

    /**
     * faz o pedido de registro
     */
    private String registerOnGCM() {
        String msg = "";
        try {
            if (gcm == null) {
                gcm = GoogleCloudMessaging.getInstance(context);
            }

            String str = gcm.register(getGoogleProjectId());

            int i=0;
            while (str.equals("SERVICE_NOT_AVAILABLE")){
                i++;
                Log.d("RECONECTANDO... ", "Tentativa : " + String.valueOf(i));
                Thread.sleep(1000);
                str = gcm.register(getGoogleProjectId());
            }

            this.regId = str;

            Log.d("GCMClient", "Registrado: " + GCMSender.this.getRegId());
            GCMSender.this.saveRegistrationID();
            this.onRegisterLoadedFromGCM();
        } catch (IOException ex) {
            msg = "Error :" + ex.getMessage();
            Log.d("GCMClient", "Error: " + msg);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Log.d("GCMClient", "AsyncTask completed: " + msg);
        return msg;
    }

    public String getGoogleProjectId() {
        return googleProjectId;
    }

    public String getRegId() {
        return regId;
    }

    public void setRegId(String regId) {
        this.regId = regId;
    }

    /**
     * metodo executado ao obter o registro
     */
    public abstract void onRegisterLoadedFromGCM();

    /**
     * deve gravar o regId nas SharedPreferences
     */
    public abstract void saveRegistrationID();

    /**
     * Implementar em caso de armazenamento do registration id
     * @return register ID salvo
     */
    public abstract String loadSavedRegistrationID();
}
```

Listagem 5 – Classe abstrata GCMSender

5 CONCLUSÃO

Neste trabalho foi identificada a possibilidade de melhoria e agilidade na tratativa de ocorrências de não-conformidades no transporte de cargas por meio do desenvolvimento de um aplicativo computacional que agiliza a resolução de não conformidades no processo de transporte de mercadorias, criando um ambiente de colaboração entre os envolvidos no processo.

Todos os objetivos propostos desse trabalho foram cumpridos, tendo como resultado o Aplicativo nomeado como ROO (Resposta Rápida a Ocorrências), que notifica rapidamente todos os envolvidos em casos de ocorrências de não conformidades no transporte de cargas.

Os principais desafios na implementação do aplicativo estiveram relacionados ao uso de mensagens por meio dos servidores do GCM, serialização genérica de objetos e lista de objetos e persistência genérica em XML. Para isso foram estudadas e utilizadas as bibliotecas Smack, Gson, XStream, Google Play Services, entre outras.

Como melhorias futuras ficam centradas na implementação dos requisitos RF11 ao RF16 do Quadro 2, bem como criação de categorias, notificação de ocorrência por email, possibilidade de colaboração com a ocorrência via *email*, envio de arquivos anexos junto com as mensagens e melhorias na interface gráfica.

REFERÊNCIAS

BALLOU, Ronald H. **Business logistics management**. 5 ed., Prentice Hall, 2001

MACHADO, Taylor M. **Contribuição ao estudo da quantificação de centros de distribuição de encomendas.** Dissertação (mestrado) Departamento de Engenharia Civil e Ambiental da Faculdade de Tecnologia da Universidade de Brasília, 2006.

MANIFESTO ELETRÔNICO DE DOCUMENTOS FISCAIS - MDF-e. Disponível em: <https://mdfe-portal.sefaz.rs.gov.br/Site/Faq>. Acesso em: 10 abr. 2015.

MARQUES, Ana Isabel R. **Optimização da cadeia de execução de encomendas Yves Rocher – Portugal, S.A.** Dissertação (mestrado). Mestrado Integrado em Engenharia Industrial e Gestão. Faculdade de Engenharia da Universidade do Porto. 2010.

PAURA, Glávio Leal. **Fundamentos da logística**. eTec Brasil, 2012. Disponível em: <http://redeetec.mec.gov.br/images/stories/pdf/proeja/fundamentos_logistica.pdf>. Acesso em: 10 abr. 2015.

PIAO, Huishu; LI, Bin; WANG, Chiyu. **Logistics standard systems based on the architecture of supply chain logistics systems**. In: IEEE International Conference on Automation and Logistics, 2009, p. 1653-1657.

PRESSMAN, Roger. **Engenharia de software.** Rio de Janeiro: McGrawHill, 2006.

WU, Wenzheng; JU, Songdong; XU, Jie. **Research on logistics resources integration in view of logistics networking**. 2010 International Conference of Information Science and Management Engineering, 2010, p. 326-330.