Maurício Vieira de Souza

SACI
Sistema de Apoio à Coleta de Informações

Porto Alegre

2014

Maurício Vieira de Souza

SACI
Sistema de Apoio à Coleta de Informações

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal do Rio Grande do Sul – UFRGS

Instituto de Informática

Curso de Ciência da Computação

Orientador: Prof. Dr. Marcelo Soares Pimenta

Porto Alegre

2014

Este trabalho é dedicado ao meu pai pela paciência e incentivo,
que tornaram esse trabalho possível.

Agradecimentos

Os agradecimentos principais são direcionados aos meus familiares pelo apoio, carinho e incentivo, à minha namorada pelo companherismo e paciência nos momentos difíceis e aos meus amigos que me apoiaram durante essa jornada.

Agradecimentos especiais são direcionados ao Marcelo Pimenta, meu orientador, que me auxiliou nessa caminhada, e à Stringhini Marketing, que contribuiu e ainda contribuirá para o desenvolvimento desse trabalho.

"A mente que se abre a uma nova idéia,
jamais voltará ao seu tamanho original.
(Albert Einstein)

Resumo

O tema principal deste trabalho é o desenvolvimento de um sistema para coleta e gerenciamento de dados. A proposta consiste no desenvolvimento de dois módulos principais: um módulo web, onde é realizada a gerência da infraestrutura, e outro módulo para *Android*, onde é feita a coleta de dados diretamente no campo. Inicialmente, é feito um panorama sobre o cliente-alvo e suas necessidades. A seguir são apresentados conceitos e tecnologias utilizadas, seguidos da descrição da implementação. Por fim, são mostrados alguns cenários de uso do sistema, seguidos por resultados, limitações e trabalhos futuros.

Palavras-chaves: Android. GWT. Serviços de Localização. Pesquisa de Mercado.

Abstract

This work focuses on the development of a system for data collection and management. The proposal consists in the development of two main modules: a web module, used for infrastructure management, and an *Android* module, where the data collection is made directly in the field. We start by giving an overview of the target customer and his needs. Then, we describe concepts and technologies used for this work, followed by the description of our implementation. Finally, we present some use cases for the system, followed by results, limitations, and future work.

Key-words: Android. GWT. Location-based Services. Market Research.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
DAO	Data Access Object
DTO	Data Transfer Object
GPS	Global Positioning System
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSNI	JavaScript Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LBS	Location-based Services
ORM	Object-relational Mapping
PDA	Personal Digital Assistant
PDV	Ponto-De-Venda
POJO	Plain Old Java Object
PPI	Pixels Per Inch
REST	Representational State Transfer
RIA	Rich Internet application
SDK	Software Development Kit
SOA	Service-oriented architecture
SGBD	Sistema de Gerenciamento de Banco de Dados
XML	Extensible Markup Language

Sumário

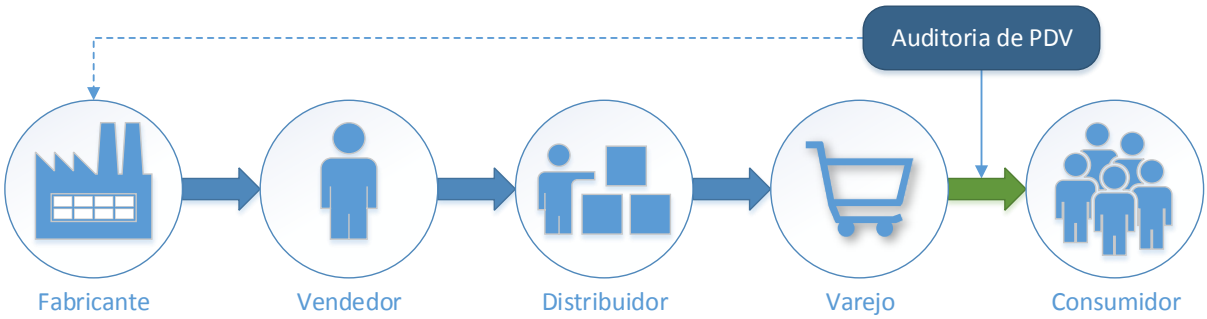
1 Introdução

Este capítulo descreve o negócio de pesquisa de mercado, mais especificamente a auditoria de ponto-de-venda, descrevendo os principais profissionais envolvidos, seguido pela motivação e objetivo desse trabalho. No final também é apresentada a estrutura em que o texto está organizado.

1.1 Contextualização

A indústria tem interesse em conhecer como seus produtos estão expostos em cada local de venda, verificando também a presença da concorrência, comunicação[1], preço dos produtos, condições de armazenamento, entre outros. A maioria das decisões de compra são feitas no próprio ponto-de-venda. Esta nova tendência justifica o aumento da importância dada ao merchandising e às promoções nas próprias lojas.

Figura 1 – Imagem da auditoria do ponto-de-venda



Fonte: Produzido pelo autor

Para verificar a execução das estratégias comerciais da indústria, junto com seus consumidores no ambiente dos estabelecimentos comerciais, é feita uma auditoria no ponto-de-venda (PDV), ilustrada na Figura 1. São objetivos desta auditoria:

a) verificar a quantidade de marcas e produtos expostos;

b) caracterizar a presença de produtos estranhos, que são disponibilizados em locais que deveriam conter somente produtos de um determinado fabricante;

c) consultar o PDV quanto à satisfação dos serviços prestados pela indústria.

[1] A comunicação de marketing é o meio pelo qual as empresas buscam informar, persuadir e lembrar os consumidores, direta ou indiretamente, sobre os produtos e marcas que comercializam (KOTLER; KELLER, 2006).

As informações coletadas na auditoria servirão para a indústria estabelecer e verificar estratégias de precificação, composição do mix de produtos, políticas comerciais e a remuneração dos vendedores. Essa auditoria envolve uma estrutura bastante complexa de processos (procedimentos e pessoas). Para garantir uma informação imparcial e de qualidade, é necessário o seguinte conjunto de profissionais:

a) analista de marketing: cria o estudo de mercado a ser aplicado de acordo com os objetivos e necessidades dos clientes;

b) estatístico: define as varáveis da coleta de dados e a composição da amostra de estabelecimentos a serem pesquisados;

c) gerente de campo: gerencia toda a realização do campo, que inclui a distribuição e gerência das tarefas;

d) supervisor: responsável pela garantia do padrão de realização da coleta de dados feita pelo entrevistador, certificando-se que todas as entrevistas sejam realizadas da mesma forma, desde a execução da amostra, abordagem do entrevistado e aplicação das perguntas;

e) entrevistador: responsável pela aplicação do instrumento de coleta de dados;

f) verificador: verifica a veracidade da fonte e da informação coletada;

g) crítico: responsável por verificar a completude e consistência dos dados coletados.

A separação entre essas atividades é rígida, embora no mercado encontrem-se situações onde o mesmo profissional desempenha mais de um desses papéis, mas isso é completamente desaconselhável.

1.2 Motivação

O grande desafio da coleta de dados entre as empresas do mercado é reduzir o número de profissionais envolvidos e aumentar a qualidade da informação coletada. O número elevado de pessoas envolvidas e o uso de procedimentos manuais impactam negativamente na qualidade da informação gerada, pois a coleta sofre com variabilidade decorrente da diferença do comportamento das pessoas.

1.3 Objetivo

Este trabalho tem como objetivo a criação de um sistema que reduz a variabilidade na coleta de dados nas auditorias de pontos de vendas, facilitando e substituindo processos já existentes.

1.4 Estrutura do Texto

Este trabalho está organizado da seguinte maneira: O Capítulo 2 faz um panorama sobre o cliente alvo, apresenta a situação atual, dificuldades e a proposta para a resolução do problema. O Capítulo 3 apresenta as tecnologias utilizadas e trabalhos relacionados. No Capítulo 4 é discutida a arquitetura e solução de implementação utilizadas. O Capítulo 5 mostra algumas interações com o sistema desenvolvido. Por fim, no Capítulo 6 são apresentados resultados, limitaçoes e trabalhos futuros.

2 Estudo de Caso

Este capítulo descreve a empresa utilizada como cliente-alvo deste trabalho, apontando as principais dificuldades enfrentadas atualmente, seguido da solução proposta que visa resolver essas dificuldades.

2.1 A Empresa

Foi utilizada, como cliente-alvo deste trabalho, uma empresa de marketing de atuação nacional, com destaque para a região sul do país que conta hoje com 5 clientes de grande porte. Sua principal área de atuação é a pesquisa de mercado, com foco no varejo de produtos de alto giro[1].

A empresa possui cerca de 60 funcionários, incluindo analistas de marketing, estatísticos, gerentes de campo, supervisores, verificadores, críticos e entrevistadores entre outros. A quantidade de entrevistadores pode variar de acordo com a demanda, podendo estes estarem vinculados a uma outra empresa parceira.

2.2 Situação Atual

A área de TI, por não ser o objetivo da empresa, é pouco desenvolvida, tornando necessária a contratação de serviços externos para suprir suas necessidades. Atualmente a empresa utiliza uma solução de software comercial utilizada para a realização das auditorias de pontos-de-venda, porém esta apresenta as seguintes dificuldades:

a) por não possuir módulo de gerenciamento, toda a configuração da estrutura de suporte às entrevistas é feita diretamente no banco de dados através de comandos *SQL*[2], procedimento, este de alto risco;

b) por não ser uma ferramenta desenvolvida especificamente para o foco da empresa, as adequações solicitadas somente são implementadas se forem de interresse amplo e, ainda assim acarretando custo financeiro;

c) possui licença por dispositivo móvel, o que nem sempre é adequada aos modelos de negócio, como, por exemplo levantamentos que envolvam grande número de entrevistadores em um curto espaço de tempo;

[1] Produtos de alto giro são aqueles que tem venda rápida, fazendo com que sejam constantemente renovados no estoque.

[2] *SQL*, ou Structured Query Language, é a linguagem de pesquisa declarativa padrão para banco de dados relacional.

d) por não ter listagem com informação de endereço ou suporte a *LBS*[3], o entrevistador precisa fazer um planejamento prévio e utilizar material externo, impactando na eficiência;

e) possui apenas um modelo de sincronização com o servidor, não sendo possível realizar uma sincronização parcial para fins gerenciais;

f) possui baixa usabilidade de software, dificultando a interação com a aplicação.

2.3 Solução Proposta

A proposta consiste no desenvolvimento de dois módulos principais: um módulo *Web*, onde é realizada a gerência da infraestrutura, e outro para *Android*, onde é feita a coleta de dados.

Todas as principais tarefas, que hoje são feitas diretamente no banco de dados, serão executadas no módulo gerenciador, como por exemplo atribuição de tarefas, gerência de projetos, pesquisas e amostras, conguração de formulários e atribuição de tarefas.

Serão usadas as melhores práticas de usabilidade no módulo *Android*, garantindo que o sistema seja consistente e agradável. É importante que o módulo possua interação direta com um mapa para que seja facilitada a identificação dos alvos a serem entrevistados. Também é necessário que exista um tipo mais leve de sincronização, onde o entrevistador somente sinaliza que certas tarefas estão concluídas, não acarretando o uso elevado de dados móveis.

Mais detelhes sobre a implementação desses módulos podem ser encontrados no Capítulo 4 e alguns cenários da aplicação podem ser vistos no Capítulo 5.

[3] *Location-based Services (LBS)* é explicado posteriormente na subseção 3.1.3.

3 Tecnologias e Trabalhos Relacionados

Neste capítulo são descritas as tecnologias utilizadas nesse trabalho: *Google Web Toolkit*, *Android*, *Location-based Services*, *Object-relational Mapping* e *RESTful Web Services*. Também são apresentados trabalhos relacionados.

3.1 Tecnologias Utilizadas

Foram utilizadas diversas tecnologias no desenvolvimento desse trabalho. Nesta seção serão descritas as principais delas.

3.1.1 Google Web Toolkit - GWT

Com a modernização dos navegadores e o avanço da internet, os sistemas web tornaram-se cada vez maiores e mais complexos, dificultando a criação e manutenção dos mesmos. Enxergando esse problema, o *Google*, em 2006, apresentou o *Google Web Toolkit (GWT)* que tinha como objetivo ajudar os desenvolvedores a criar e manter códigos *JavaScript* complexos utilizando *Java* (KEREKI, 2010).

Usando *GWT*, desenvolvedores podem desenvolver e depurar sistemas *AJAX*[1] utilizando *Java*. Os principais componentes incluídos no *GWT* são:

a) *GWT Java-to-JavaScript Compiler*: Converte o programa escrito em *Java* para *JavaScript*;

b) *GWT Development Mode*: Permite que o desenvolvedor execute aplicações *GWT* em modo de desenvolvimento (O aplicativo roda em *Java* na *JVM*, possibilitando a depuração normal do código);

c) *JRE emulation library*: Implementação *JavaScript* das classes mais utilizadas da biblioteca padrão do *Java* (como a maioria das classes dos pacotes *java.lang* e *java.util*);

d) *GWT Web UI class library*: Conjunto de interfaces e classes customizáveis para a criação de *Widgets*.

As principais características do *GWT* são a geração de múltiplas versões da aplicação para se adaptar aos diversos browsers e realização de otimização e ofuscamento do código. Isto pode ser visto na Figura 2. Também podemos citar algumas vantagens e desvantagens:

[1] *AJAX* é uma técnica utilizada em sistemas web que possibilita a comunicação assíncrona com o servidor (GARRETT et al., 2005).

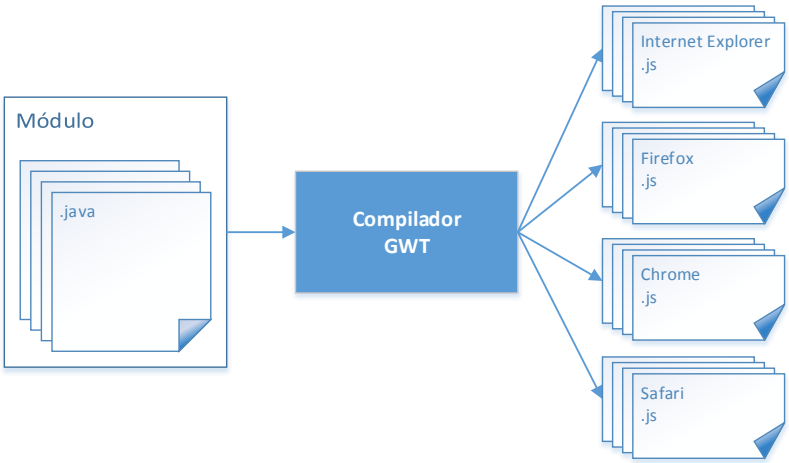
a) vantagens:

- suporta teste unitário utilizando *JUnit*[2];
- elimina trechos de código não utilizados;
- é um projeto que possui seu código aberto desde 2012;
- suporte a internacionalização padrão do *Java*;
- controle total sobre a aplicação e a possibilidade de integração direta com *JavaScript* utilizando *JSNI*[3].

b) desvantagens:

- não é possível utilizar todas as classes da biblioteca padrão do *Java*, apenas um sub-conjunto delas;
- as páginas geradas pelo *GWT* não são indexáveis pelos mecanismos de busca;
- desenvolvedores podem reclamar que o desenvolvimento é mais lento comparado à utilização direta do *JavaScript*, porém o código gerado pelo *GWT* será provavelmente mais robusto, rápido e compatível;
- biblioteca padrão de widgets muito limitada, porém, com as ferramentas fornecidas pelo próprio *GWT*, é possível construir seus próprios widgets e, devido ao *JSNI*, diversas bibliotecas *JavaScript*, já consolidadas no mercado, fornecem versões encapsuladas para o *GWT*.

Figura 2 – Esquema da compilação do *GWT*



Fonte: Produzido pelo autor

[2] *JUnit* é um framework de código aberto criado por Erich Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação *Java*.

[3] O *JavaScript Native Interface (JSNI)* permite escrever códigos diretamente em *JavaScript* dentro de uma aplicação *GWT*, podendo inclusive utlilizar trechos de código *Java* dentro do *JavaScript* e vice-versa (CHAGANTI, 2007).

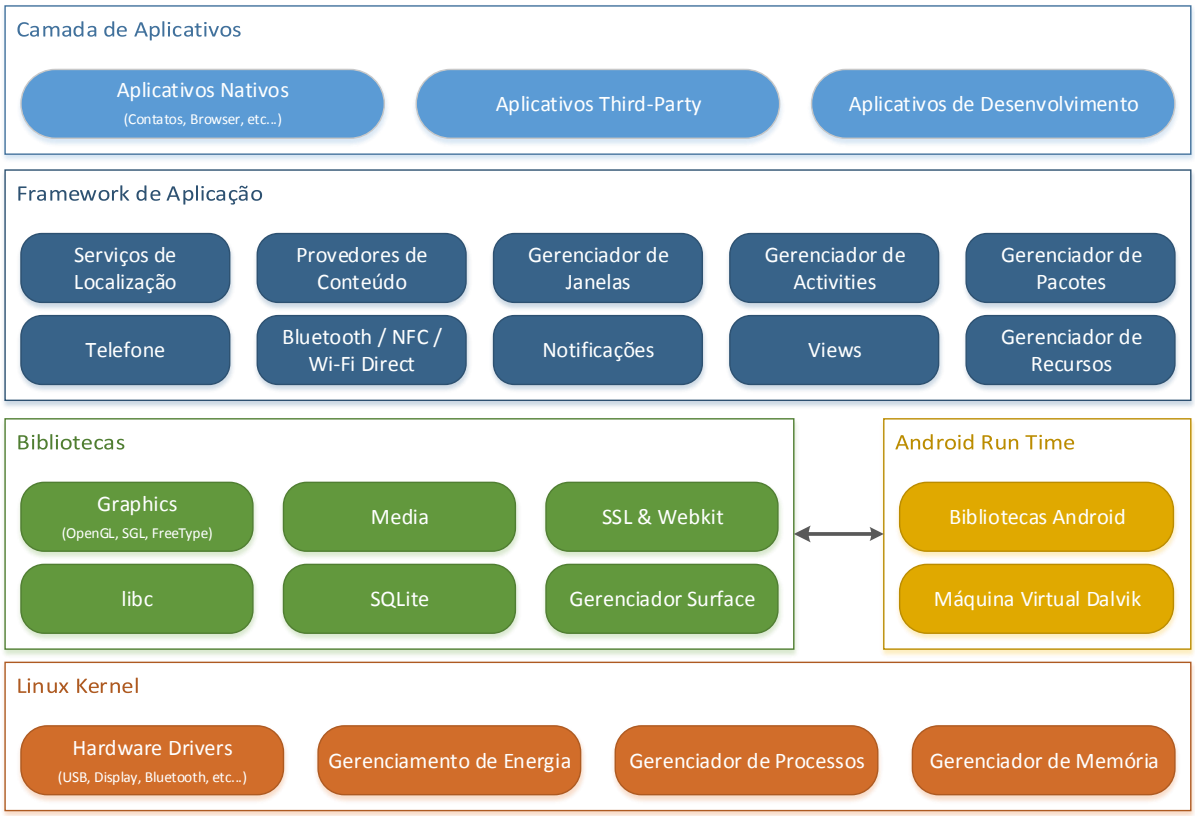
O *Google Web Toolkit* foi utilizado nesse trabalho para a criação de um módulo web de configuração e gerenciamento explicado posteriormente na subseção 4.4.3.

3.1.2 Android

Com o intuito de desenvolver uma nova plataforma para dispositivos móveis, o *Google* adquiriu uma empresa startup homônima no ano de 2005. Mais tarde, empresas líderes no segmento de dispositivos móveis se reuniram com o objetivo de padronizar uma plataforma de código aberto e livre, podendo, assim, responder às necessidades dos consumidores e desenvolvedores na área das plataformas móveis.

Como mostra a Figura 3, a arquitetura do *Android* pode ser exibida como uma pilha de softwares para dispositivos móveis que inclui um sistema operacional, middleware e aplicações principais (MEIER, 2012). A linguagem da plataforma é em *Java*, assim, trabalha com a ideia de portabilidade de hardware, estando disponível, de forma gratuita e aberta, através de um kit de desenvolvimento de software (*SDK*).

Figura 3 – Esquema da arquitetura do *Android*



Fonte: Produzido pelo autor com base em Meier (2012)

A principal característica do *Android* é o sistema aberto, onde qualquer fabricante pode optar pelo seu uso sem a necessidade de pagar por qualquer tipo de licenciamento,

possuindo ainda liberdade para fazer modificações e modelar o sistema da forma que achar mais adequado. Outra característica relevante é a versatilidade, pois permite que os fabricantes possam produzir aparelho para diversos nichos, já que há possibilidade de intervenção no sistema.

Dentre as principais vantagens de se desenvolver para *Android* podemos destacar:

a) baixo investimento e alto retorno do mesmo: as ferramentas para se desenvolver para *Android* são disponibilizadas de forma gratuita e não exigem nenhuma plataforma específica; portanto, não ocasionando nenhum custo adicional para o desenvolvimento;

b) código aberto: a arquitetura do *Android* possui código aberto, possibilitando interação com a comunidade sobre as novidades e rumos tomados no desenvolvimento;

c) fácil de integrar: aplicativos *Android* podem ser facilmente integrados com sistemas *Web* ou com outros aplicativos;

d) múltiplos canais de venda: diferentemente das outras plataformas móveis, as aplicações *Android* podem ser distribuídas de diferentes maneiras. Não é necessário uma dependência estrita a um único mercado para a distribuição e venda dos aplicativos;

e) fácil adoção: as aplicações são desenvolvidas utilizando *Java*, que é amplamente utilizada e possui um rico conjunto de bibliotecas. Qualquer desenvolvedor *Java* pode desenvolver aplicativos para *Android*, tornando muito fácil sua adoção.

O *Android* foi utilizado como plataforma nesse trabalho para a criação de um módulo móvel explicado posteriormente na subseção 4.4.4.

3.1.3 Location-based Services - LBS

Uma das funções mais importantes adicionadas à crescente popularização dos smartphones é o acesso a serviços baseados em localização (*LBS*), permitindo ao usuário descobrir, acessar e aproveitar da melhor forma o ambiente que os cerca. O LBS existe desde a implantação do Sistema de Posicionamento Global (*GPS*) na década de 70. Mas somente a partir da década de 90 ganhou mais destaque e despertou um maior interesse. Isso se deve principalmente às operadoras de redes móveis que começaram a oferecer serviços de dados, surgindo assim, a necessidade de integração com informações de localização.

Segundo Schiller e Voisard (2004), o termo serviços baseados em localização (*LBS*) é um conceito que visa representar as aplicações que integram localização geográfica com uma noção geral de serviços. Conforme ilustrado na Figura 4, um *LBS* é composto de alguns componentes básicos como:

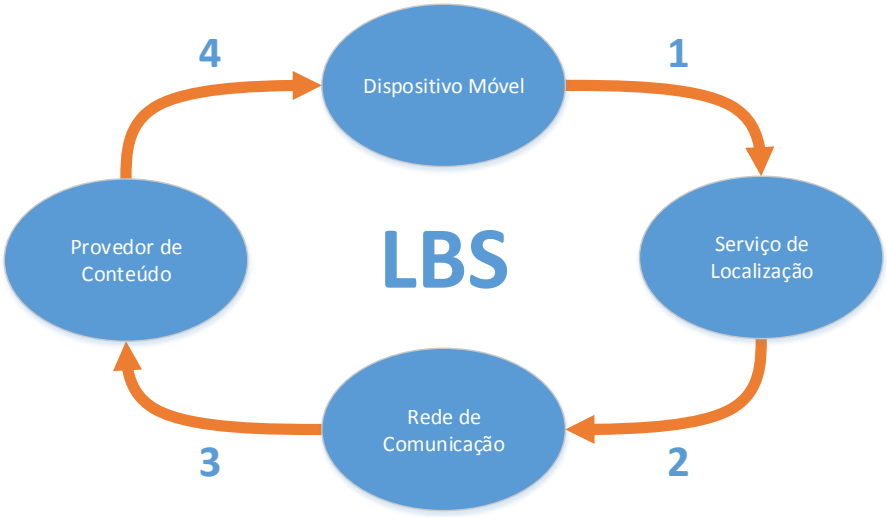
a) dispositivo móvel: dispositivos usados para pedir informação, onde os resultados podem ser apresentados como áudio, vídeo, imagem, texto, etc;

b) rede de comunicação: é a forma de comunicação do dispositivo móvel com os provedores de conteúdo;

c) componente de localização: é responsável por determinar a posição, podendo ser determinada pela rede móvel, *GPS* ou *Wi-Fi*;

d) provedor de conteúdo: é a fonte de dados da aplicação que vai utilizar a informação de localização fornecida pelo dispositivo móvel.

Figura 4 – Esquema dos componentes de um *Location-based Service*



Fonte: Produzido pelo autor com base em Schiller e Voisard (2004)

Location-based Services foram utilizados nesse trabalho para o entrevistador se localizar em relação aos pontos onde serão realizadas as entrevistas. Maiores detalhes na subseção 4.4.4.

3.1.4 Object-relational Mapping - ORM

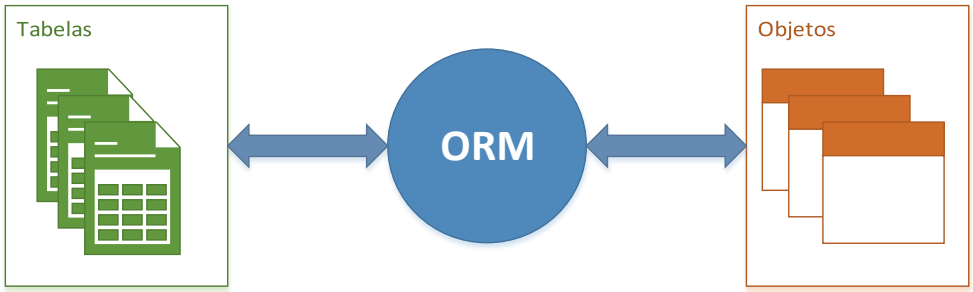
Os sistemas estão cada vez mais complexos e linguagens orientadas a objetos são as mais utilizadas no desenvolvimento atual. Além disso, o modelo mais utilizado quando se pretende persistir dados ainda é o modelo relacional, porém este modelo e a orientação a objetos são incompatíveis.

Como forma de resolver esse problema surgiu a idéia do mapeamento objeto-relacional, que segundo Christian e Gavin (2006) é a persistência de maneira automática e transparente dos objetos de um aplicativo para tabelas em um banco de dados relacional. Como pode ser visto na Figura 5, o ORM basicamente transforma dados de uma representação para a outra.

O ORM realiza a difícil tarefa de gerenciar as interações da aplicação com o banco de dados. Uma vez criados os mapeamentos necessários para os objetos da aplicação, estes gerenciam completamente o acesso aos dados. Não é necessário escrever nenhum código *SQL*, isto proporciona uma melhor separação entre a aplicação e a forma que será feita a persistência da mesma.

Essa separação possibilita ao sistema a possibilidade de utilizar banco de dados com dialetos de *SQL* diferentes, já que o dialeto pode ser especificado como parâmetro.

Figura 5 – Diagrama da trasformação entre representações feita pelo *ORM*



Fonte: Produzido pelo autor com base em Christian e Gavin (2006)

Existem inúmeros benefícios de se utilizar uma ferramenta *ORM* no desenvolvimento de software, a seguir estão alguns deles:

a) produtividade: a persistência geralmente é parte significativa em uma aplicação típica e o tempo utilizado para sua implementação também é. Ao se utilizar uma ferramenta *ORM*, a quantidade de código dificilmente irá diminuir, porém grande parte do código é gerado automaticamente ou reutilizado;

b) reuso de código: como as entidades do banco de dados são representadas como classes, toda a reutilização proveniente da orientação a objetos pode ser aproveitada;

c) design: uma boa ferramenta *ORM* é desenvolvida utilizando diversos padrões de projeto[4], que, por sua vez, conduzem o desenvolvedor a utilizar boas práticas de programação;

d) facilidade de manutenção: todo código gerado pelo *ORM* é presumivelmente testado, portanto não é necessário se preocupar em testá-lo exaustivamente. Também é possível refatorar o esquema do banco de dados ou do modelo de classes sem afetar a maneira com que a aplicação utiliza os objetos da camada de persistência.

[4] Padrões de Projeto, ou *Design Patterns*, descrevem soluções ótimas para problemas recorrentes. Mais informações podem ser encontradas em Freeman et al. (2004)

Neste trabalho foram utilizados o *Hibernate* como solução *ORM* no servidor e o *ORMLite* no Android, ambos explicados respectivamentes nas sessões 4.4.2 e 4.4.4.

3.1.5 RESTful Web Services

A necessidade de integração entre diferentes aplicações desenvolvidas em diferentes linguagens, foi o que motivou a criação do conceito de *Web Service*. O chamado *RESTful Web Service*, ilustrado na Figura 6, é o tipo mais utilizado hoje em dia devido à sua simplicidade, escalabilidade e versatilidade provenientes do uso dos princípios *REST*[5].

De acordo com Rodriguez (2008), uma implementação concreta de um *RESTful Web Service* deve seguir quatro princípios básicos de design:

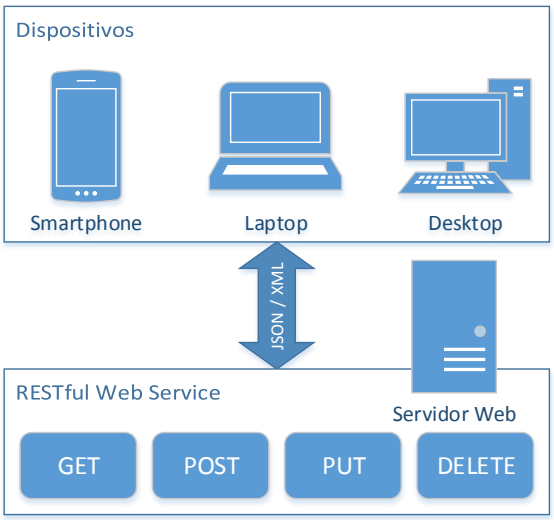
a) uso explícito de métodos *HTTP*: *POST*, *GET*, *PUT* e *DELETE* são usados explicitamente nas requisições;

b) não possuir estado: cada requisição é independente uma da outra, isto é, requisições que incluem todos os dados necessários para sua execução;

c) expor endereços como uma estrutura de diretórios: endereços de serviços devem ser intuitivos ao ponto de serem facilmente descobertos e utilizados;

d) transferir *XML*, *JSON* ou ambos: o formato em que a informação é trasferida é outro ponto importante. Ambos formatos são textuais, simples e inteligíveis.

Figura 6 – Diagrama de operação de um *RESTful Web Service*



Fonte: Produzido pelo autor com base em Rodriguez (2008)

[5] *REST* ou *Representational State Transfer* é um conjunto de princípios que definem de que maneira padrões web como *HTTP* devem ser usados. Mais informações sobre *REST* podem ser encontradas em (FIELDING, 2000).

Neste trabalho toda a comunicação entre o Android e o Servidor foi feita utilizando *RESTful Web Services*. Mais detalhes serão discutidos na subseção 4.4.4.

3.2 Trabalhos Relacionados

O trabalho de Tavares (2011) consiste em propor sugestões que facilitam o uso de dispositivos móveis que são utilizados na coleta de dados estatísticos, especialmente *Personal Digital Assistants (PDAs)*, com o objetivo de deixar o trabalho dos entrevistadores mais eficiente, agilizando a entrada de dados, ajudanto na leitura das perguntas para melhorar o entendimento dos informantes, maximizando a qualidade e a integridade dos dados coletados.

O trabalho de Farias e Kuhn (2013) tem como objetivo realizar uma avaliação e comparação entre as ferramentas disponíveis para projeto e desenvolvimento de aplicativos *Android* e suas possíveis alternativas.

Já o trabalho de Junior e Pimenta (2013) apresenta as principais diferenças entre a engenharia de software do modelo clássico e a aplicada ao desenvolvimento de aplicativos para dispositivos móveis. Tem por objetivo explicar os aspectos relevantes no planejamento e desenvolvimento de aplicativos, evidenciando as vantagens do estabelecimento de técnicas de engenharia de software específicas para o desenvolvimento de aplicativos móveis.

4 Desenvolvimento e Implementação

Este capítulo descreve o desenvolvimento e a implementação do sistema descrito na seção 2.3. É apresentada uma contextualização do desenvolvimento, seguida da visão geral e arquitetura do sistema. Depois é feita uma descrição sobre a infraestrutura utilizada e é exibido o modelo de dados do sistema. Finalmente são apresentados detalhes da implementação dos módulos do sistema.

4.1 Contextualização

Por uma questão de independência, o desenvolvimento foi feito utilizando-se infraestrutura própria, o que direcionou para a utilização de ferramentas livres, preferencialmente de código aberto, sem que, com isso, deixasse de considerar as peculiaridades da empresa. Os dados, por exemplo, foram mantidos em uma estrutura relacional, uma vez que os mesmos são utilizados em outras áreas da empresa.

O cliente definiu somente um conjunto de objetivos gerais para o sistema, não especificando exatamente como deveriam ser realizadas as tarefas. Devido a isto, foi utilizada a abordagem de prototipação, de forma incremental e interativa, com reuniões frequentes e participação intensa do cliente nas tomadas de decisões.

A versão de desenvolvimento do sistema foi utilizada como protótipo e esteve sempre disponível para que o usuário pudesse interagir com o mesmo, de forma a validá-lo em relação às suas expectativas.

4.2 Visão Geral

Como ilustrado na Figura 7, a solução é composta de três partes: o cliente web, o cliente android e o servidor.

O cliente *Web* foi construído utilizando uma interface *RIA*[1], possibilitando uma melhor interação com o usuário e pode ser utilizado tanto pelo usuário gestor quanto pelo entrevistador. Neste módulo é possivel realizar as seguintes atividades:

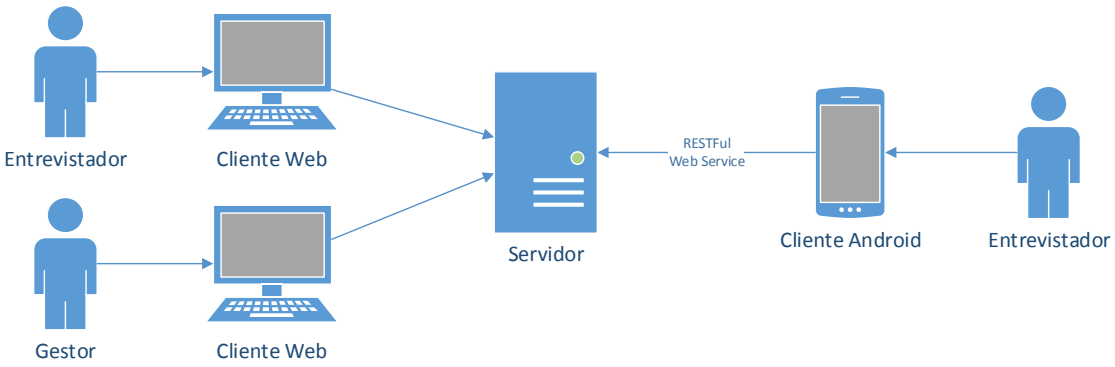
a) cadastro de clientes, usuários e dispositivos;

b) criação e gerenciamento de formulários;

c) definição de tipos e domínios utilizados pelos formulários;

[1] *Rich Internet Application (RIA)*: Aplicação web que lembra uma aplicação desktop, possibilitando interações sofisticadas com o usuário, processamento no lado do cliente, comunicação assíncrona e multimídia. (FRATERNALI; ROSSI; SÁNCHEZ-FIGUEROA, 2010)

Figura 7 – Esquema da visão geral do sistema



Fonte: Produzido pelo autor

d) administração de projetos e pesquisas;

e) gerenciamento de amostras e alvos;

f) atribuição, execução e verificação de tarefas;

g) controle de uso dos dispositivos.

O cliente *Android* foi desenvolvido utilizando a própria *API*[2] do *Android*, já que não existe a necessidade de porte para outra plataforma. Seu foco é a usabilidade e toda comunicação com o servidor é feita via *Web Services*. Este módulo é utilizado pelos entrevistadores e tem as seguintes funcionalidades:

a) realização de entrevistas;

b) sincronização de tarefas e resultados;

c) suporte multiusuário;

d) persistência relacional;

e) utilização de serviços *LBS*.

O servidor foi desenvolvido em *Java*, com o auxílio de diversos frameworks e é responsável por assegurar o acesso, persistência e transferência segura dos dados entre os diferentes clientes. O servidor é utilizado somente pelos outros módulos e tem as seguintes funcionalidades:

a) toda a transferência de dados é feita via *HTTPS*;

b) persistência independente do *SGBD* utilizado;

c) utiliza soluçao de segurança robusta e bem abrangente;

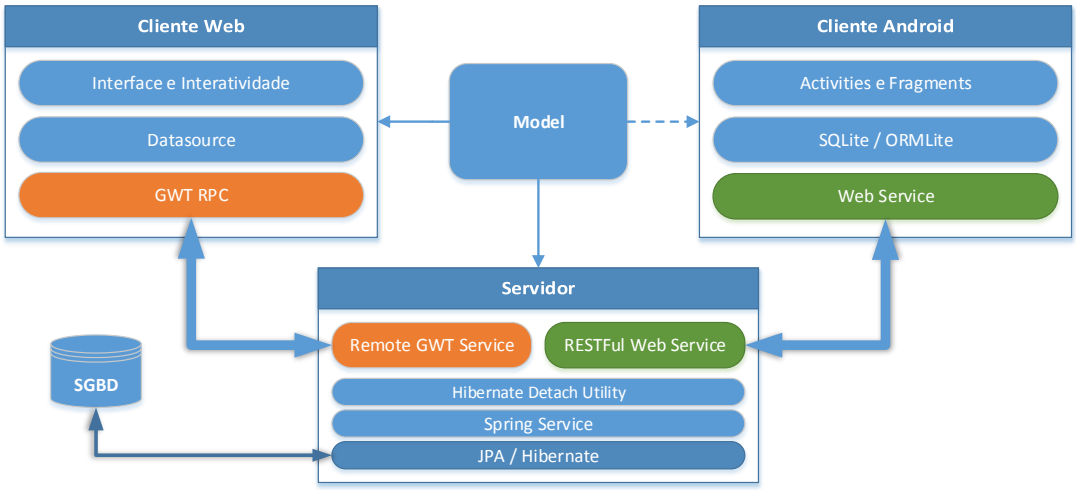
d) possui uma interface *REST* para se comunicar com o cliente *Android*.

[2] *Application Programming Interface (API)*: especificação estabelecida por um software para a utilização das suas funcionalidades. (BLOCH, 2008)

4.3 Arquitetura

A solução de arquitetura adotada utiliza um padrão *client-MVC*, também conhecida como arquitetura *RIA/SOA*[3], onde todos os aspectos da apresentação são controlados no lado do cliente e o servidor provê acesso seguro e consistente aos dados.

Figura 8 – Diagrama de arquitetura do sistema



Fonte: Produzido pelo autor

Como pode ser ilustrado na Figura 8, o model é compartilhado entre todos os módulos, porém no *Android* é utilizado um subconjunto do mesmo. A comunicação com o cliente *Web* é feita utilizando *GWT RPC* e com o *Android* utilizando *RESTFul Web Services*, ambos sobre *HTTPS*.

4.4 Implementação

A descrição da implementação foi dividida em partes por motivos de organização e apresenta uma descrição da infraestrutura e do modelo de dados utilizados, seguido do detalhamento da implementação de cada um dos três módulos do sistema.

4.4.1 Infraestrutura e Modelo de Dados

Como listado na Tabela 1, diversas ferramentas gratuitas e de altíssima qualidade foram utilizadas durante o desenvolvimento deste projeto.

Para a realização do mapamento objeto-relacional, explicado na subseção 3.1.4, foi utilizado o framework *Hibernate*, que é atualmente a referência de mercado em *ORM*

[3] *Service-oriented architecture (SOA)*: estilo de arquitetura em que as funcionalidades de uma aplicação devem ser disponibilizadas em forma de serviços (ERL, 2005).

Tabela 1 – Ferramentas utilizadas no desenvolvimento

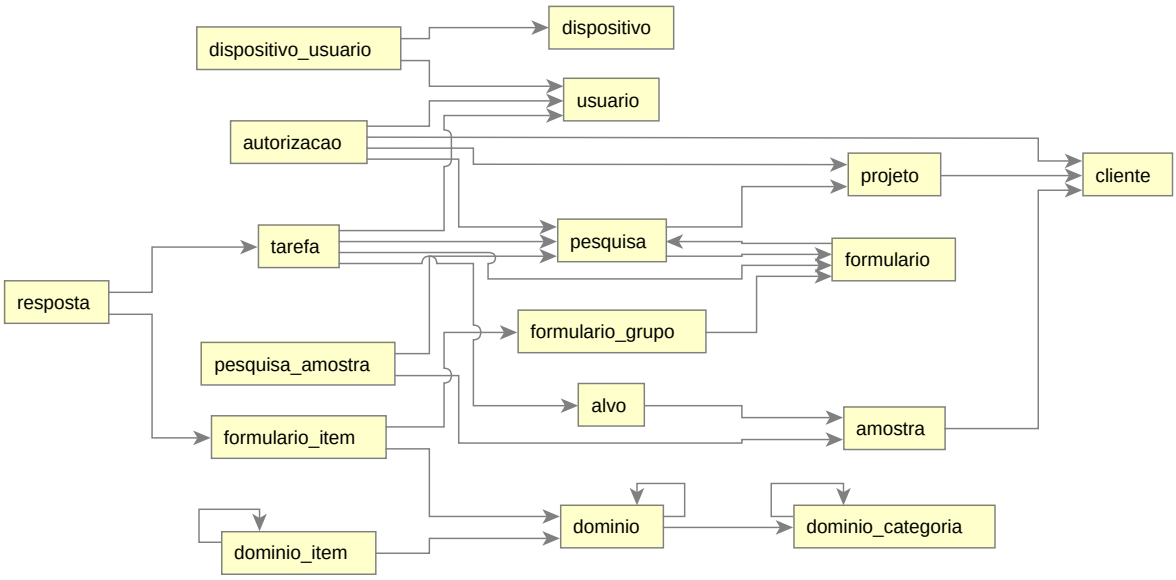
IDE de desenvolvimento	Eclipse Kepler
SGBD	PostgreSQL 9
Database Tool	DbVisualizer 9
Servlet Container	Apache Tomcat 7/8
Java Runtime Environment	Oracle Java 7
REST Debugger	WizTools RESTClient
KeyStore Manager	KeyStore Explorer 5

Fonte: Produzido pelo autor

e implementa a especificação *JPA*[4]. Todas as classes persistidas e suas relações foram definidas com o uso do *Hibernate*, porém foram utilizadas somente anotações presentes na especificação.

A partir da análise do sistema, foram construídas as classes de negócio que resultaram na estrutura referencial ilustrada na Figura 9. O modelo e a descrição das classes de negócio presentes no diagrama podem ser encontrados no Apêndice A.

Figura 9 – Modelo referencial geral



Fonte: Produzido pelo autor

Devido à diversidade dos estabelecimentos visitados e ao objetivo do levantamento desejado pelo cliente é necessário segmentar o projeto por critérios, como faixa de renda dos clientes, tamanho do estabelecimento, faturamento ou outros. Para isso, os projetos do cliente são compostos de pesquisas, que possuem seus próprios formulários (conjunto de perguntas) e amostras de alvos a serem entrevistados.

[4] *Java Persistence API (JPA)*: especificação *Java* para a persistência de dados utilizando mapeamento objeto-relacional. Mais informações podem ser encontradas em Christian e Gavin (2006).

A tarefa e seu conjunto de respostas representam a entrevista aplicada, associando o entrevistador, o alvo, a pesquisa e o formulário corrente no momento da criação da tarefa.

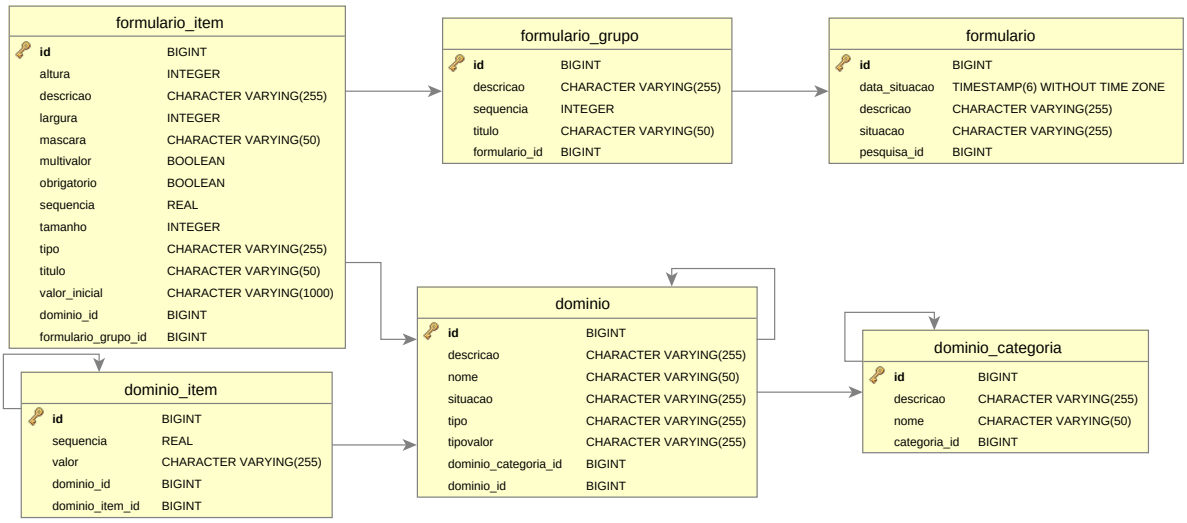
Essa vinculação da tarefa com o formulário é necessária devido ao ciclo de vida da tarefa ser mais curto que o da pesquisa, permitindo que o formulário evolua sem impactar nas tarefas já criadas.

O maior desafio encontrado na persistência deste projeto foi a modelagem das entidades necessárias à construção dos formulários dinâmicos. O modelo referencial, gerado a partir das classes de domínio, pode ser visualizado na Figura 10.

Com esse modelo é possível definir formulários e domínios de respostas possíveis e prover uma forma de organização para os mesmos. Um formulário é composto de grupos que, por sua vez, são compostos de itens.

Os itens, na prática, são as perguntas a serem respondidas; cada item estará vinculado a um domínio, de forma a determinar as possíveis respostas a serem fornecidas. Ainda poderá ser definida a forma como o item será apresentado na tela, por exemplo, máscara, tipo de widget e valor inicial.

Figura 10 – Modelo referencial dos formulários dinâmicos



Fonte: Produzido pelo autor

O domínio poderá ser algo simples como um número inteiro, ou algo mais complexo como uma lista de valores definidos pelo usuário, ou ainda poderá ser uma lista vinculada a uma outra lista, como por exemplo, cidades pertencentes a estados.

4.4.2 Servidor

O servidor foi implementado em 3 camadas: a camada de comunicação, que é composta dos *Servlets*[5] *GWT* e dos *Web Services*, a camada de serviços e a camada de acesso aos dados. Todas essas camadas estão cobertas pela solução de segurança adotada.

4.4.2.1 Comunicação

A comunicação com o cliente *Web* é feita utilizando *GWT RPC*, que serializa os objetos a serem transferidos. Como as classes do modelo são persistidas pelo *Hibernate*, este modifica os objetos reescrevendo seu *bytecode*[6], isso significa que, quando algum desses objetos vai ser transferido o mecanismo de serialização do *GWT RPC* não sabe mais como proceder, já que seu *bytecode* difere do código fonte (CHANDEL, 2009). Pare esse problema foram encontradas as seguintes soluções:

a) *Data Transfer Object (DTO)*: consiste na criação de simples *Plain Old Java Objects (POJOs)* contendo somente os campos que podem ser acessados no lado do cliente. Pode ser muito difícil evitar a explosão no número de objetos duplicados;

b) *Dozer*: é uma biblioteca que gera automaticamente *DTOs* e pode ser usada para clonar as entidades do *Hibernate* antes das mesmas serem serializadas pelo mecanismo do *GWT RPC*;

c) *Gilead*: é uma biblioteca que substitui as proxies não inicializadas por *null* e coleções persistidas por coleções básicas. *Gilead* ainda armazena toda informação necessária para recriar esses objetos sem a necessidade de fazer chamadas para o banco de dados. É uma ótima solução, apesar de possuir um escopo muito maior do que o necessário nesse trabalho, porém foi descontinuada pelo seu desenvolvedor.

d) *Hibernate Detach Utility*: classe utilitária que substitui as proxies não inicializadas por *null* e coleções persistidas por coleções básicas. Realiza somente o necessário para esse trabalho, tendo performance maior e complexidade menor que utilização do *Gilead*.

Nesse trabalho foi utilizado o *Hibernate Detach Utility* e o mesmo é usado nos objetos antes que os mesmos sejam serializados pelo *GWT RPC*, impactando muito pouco no desempenho e preservando as relações entre os objetos, evitando, dessa forma, o problema gerado pelo uso do *DTO*.

[5] *Servlet* é uma classe *Java* usada para estender as funcionalidades de um servidor, normalmente usada em aplicativos *Web*.

[6] *Bytecode Java* é uma forma intermediária de código, que é interpretada pelas *Máquinas Virtuais Java (JVMs)*.

Os web services foram implementados utilizando o framework *Jersey*, que é um dos mais utilizados para a construção de *RESTful Web Services* em *Java*, e a biblioteca *GSON*, que foi utilizada para converter objetos *Java* em sua representação *JSON* e vice-versa.

4.4.2.2 Serviços

A camada de serviços foi mantida por uma decisão de projeto, fazendo somente a ligação entre a camada de comunicação e a camada de acesso aos dados, garantindo total separação entre essas camadas. Em alguns dos serviços é feita a adequação dos dados para a camada de comunicação.

4.4.2.3 Acesso aos Dados

O acesso aos dados é feito utilizando o padrão *Data Access Object (DAO)*, que separa as regras de acesso ao banco de dados das regras de negócio da aplicação, o que é bom, pois garante que ambas as partes possam evoluir independentemente.

A gerência de transações é realizada pelo framework *Spring*, que gerencia todo o ciclo de vida dos objetos que se comunicam com o banco de dados, sendo possível propagar transações até que as mesmas não sejam mais necessárias.

4.4.2.4 Segurança

A segurança foi implementada utilizando o *Spring Security*, que é um framework que se propõe a realizar autenticação e autorização para aplicações *Java*. O sistema desenvolvido permite que o usuário faça autenticação com login e senha ou utilizando certificação digital.

4.4.3 Cliente Web

A complexidade das tarefas que são desempenhadas no cliente *Web* exigem uma interface bastante rica e responsiva, por isso foi utilizado o *GWT*, que permite que a lógica da aplicação fique toda no cliente, garantindo que após o carregamento inicial do sistema, o mesmo só se comunique com o servidor quando for realmente necessário.

4.4.3.1 Interface e Interação

Devido a limitação dos componentes visuais da biblioteca padrão do *GWT*, foi utilizada a plataforma *SmartGWT*, que aproveita-se do *JSNI*, encapsulando os componentes visuais da biblioteca *JavaScript SmartClient* em classes Java que serão utilizadas pelo compilador *GWT*. O *SmartGWT* oferece uma camada de apresentação completa com widgets altamente poderosos e versáteis.

4.4.3.2 Datasource

Outro componente importante do *SmartGWT* é o Datasource, que gerencia o acesso aos dados no cliente, incluindo toda a mecânica de fetch, cache e paginação dos dados. Além disso o datasource também controla os componentes visuais vinculados a ele, garantindo uma separação bem clara entre a interface e a lógica da aplicação.

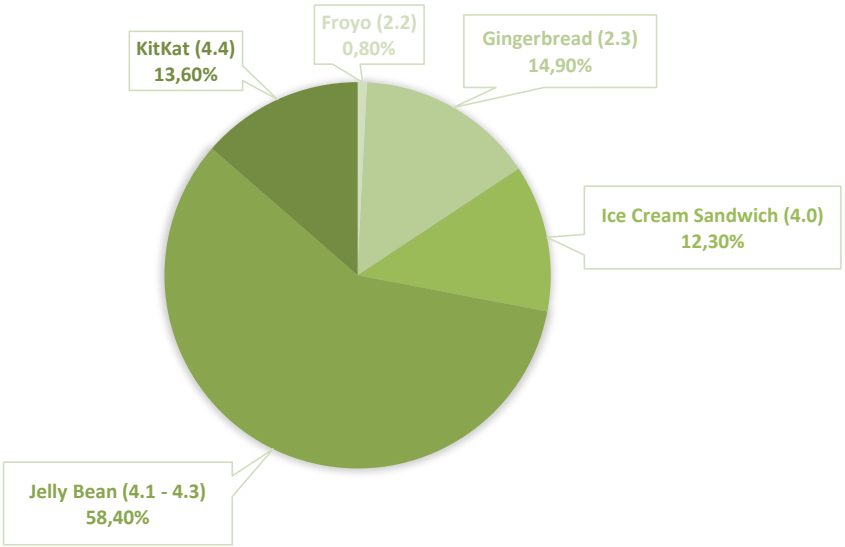
4.4.4 Cliente Android

O cliente *Android* foi construído levando em consideração diversas recomendações feitas pelo *Google* nas sessão de *Design e Develop* do site de desenvolvimento *Android*, garantindo excelente usabilidade de software e organização do código.

4.4.4.1 Compatibilidade

Como pode ser visto na Figura 11, atualmente, 84,3% dos dispositivos utilizam versão 4.0 ou superior do *Android* e, como existe diferença substancial da versão 4.0 em relação às anteriores e todos os dispositivos do cliente já possuem versão igual ou superior a essa, foi decidido apenas suportar dispositivos a partir dessa versão, o que facilita o desenvolvimento, já que reduz drasticamente o uso da biblioteca de compatibilidade[7].

Figura 11 – Gráfico da distribuição das versões do *Android*



Fonte: Produzido pelo autor com base Google (2014)

[7] O *Android Support Library* é um pacote que contém diversas bibliotecas que podem ser incluídas nas aplicações com o objetivo principal de portar novas funcionalidades para versões antigas do *Android*, porém algumas funcionalidades estão presentes somente nesse pacote.

4.4.4.2 Persistência

A autonomia da bateria é algo muito importante para o entrevistador e este é aconselhado a utilizar o dispositivo móvel em modo avião, necessitando ser possível realizar todas as tarefas sem acesso à internet. Levando-so isso em consideração, foi desenvolvida uma estrutura completa de persistência de formulários, respostas e tarefas igual a utilizada nos outros módulos.

Por motivos de organização e facilidade de manutenção do código, foi utilizado o framework *ORMLite*, que permite a realização do mapeamnto objeto-relacional no *Android* e tem funcionamento muito semelhante ao do *Hibernate*, porém mais limitado. Apenas uma parte das entidades estão presentes no cliente *Android* já que é um módulo focado apenas na execução das tarefas.

4.4.4.3 Location-based Services

Parte da dificuldade existente no sistema, atualmente utilizado pela empresa, deve-se ao fato de que o entrevistador necessita de material externo para poder organizar sua rota diária. Definir aonde ir é muito mais fácil com o auxílho de um mapa que mostre a localização atual do entrevistador e de suas tarefas pendentes.

Para fins de economizar dados móveis e bateria, toda a geolocalização dos alvos é feita previamente no servidor, garantindo que, mesmo em modo offline, seja possível visualizar os alvos no mapa. Adicionalmente, quando uma entrevista é concluída, a localização atual é salva para fins de conferência.

4.4.4.4 Usabilidade

O módulo *Android* foi desenvolvido seguindo os princípios de design fornecidos pelo *Google* com o objetivo de garantir boa usabilidade por construção. Foram realizados testes com dispositivos de diversos tamanhos e densidades de tela (Tabela 2) a fim de verificar a adequação da interface do sistema às diferentes configurações.

Tabela 2 – Dispositivos utilizados no desenvolvimento

Dispositivo	**Tamanho da Tela (pol.)**	**Densidade (ppi)**
Samsung Galaxy Ace II	3.8	246
Motorola Moto G	4.5	326
Samsung Galaxy S III	4.8	306
LG G2	5.2	424
LG G Pad 8.3	8.3	273

Fonte: GSMArena (2014)

5 Cenários de Uso

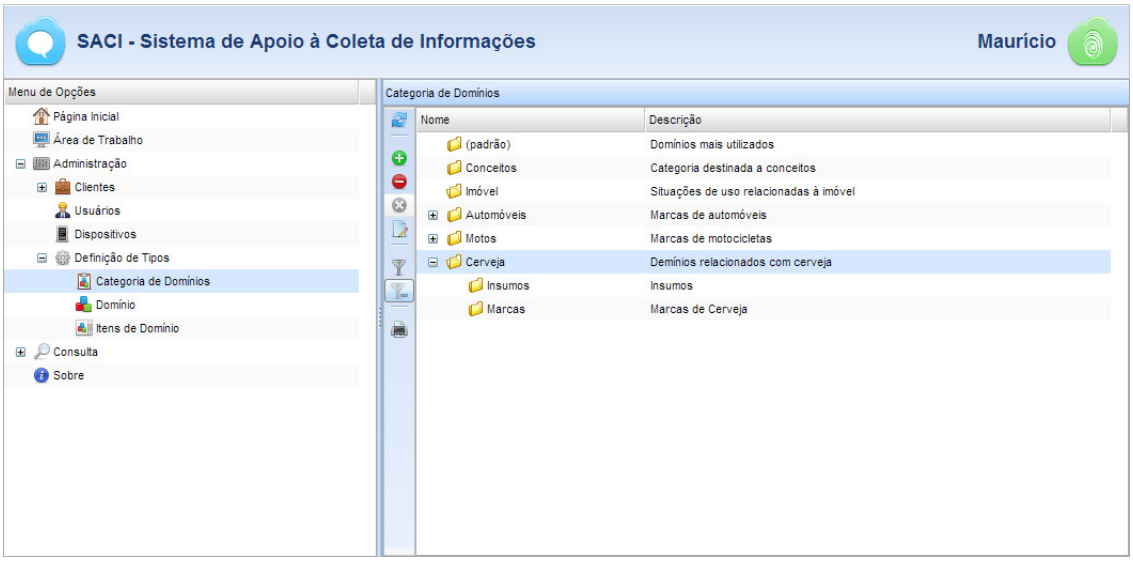
Neste capítulo serão apresentadas diversas telas do sistema, fornecendo uma visão geral de como são feitas algumas configurações do modelo dinâmico de coleta de informações, e sua execução no dispositivo móvel.

5.1 Configuração de Domínios

Na base do sistema temos a configuração de domínos de respostas possíveis, podendo ser configurados: categorias de domínios, domínios e itens de domínio.

As categorias de domínio, ilustradas na Figura 12, têm como objetivo a organização dos diversos domínios em categorias e subcategorias. A categoria (padrão) é destinada aos domínios mais utilizados, como por exemplo números inteiros, texto, data, etc. As categorias e subcategorias podem ser rearranjadas a qualquer momento, bastando arrastar para a posição desejada.

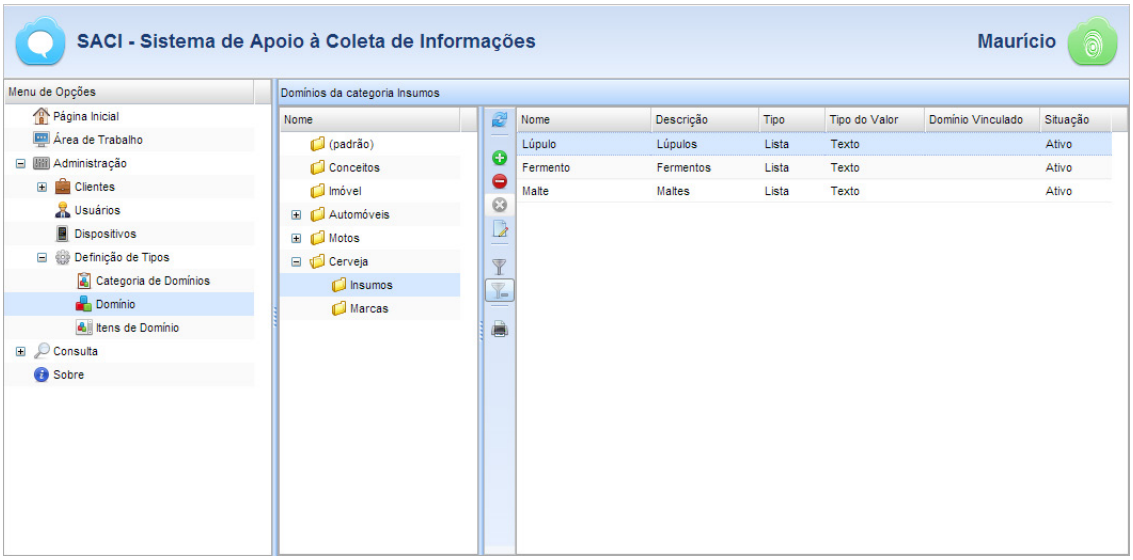
Figura 12 – Tela de configuração de categorias de domínio



Fonte: Produzido pelo autor

A definição dos domínios que as respostas podem assumir é feita através da tela ilustrada na Figura 13. Os domínios podem ser simples, como texto, data ou número, ou algum valor de uma lista, que é o caso de domínios do tipo lista ou lista vinculada. Assim como as categorias de domínio, os domínios podem ser rearranjados nas diversas categorias, bastando arrastá-los para a categoria desejada.

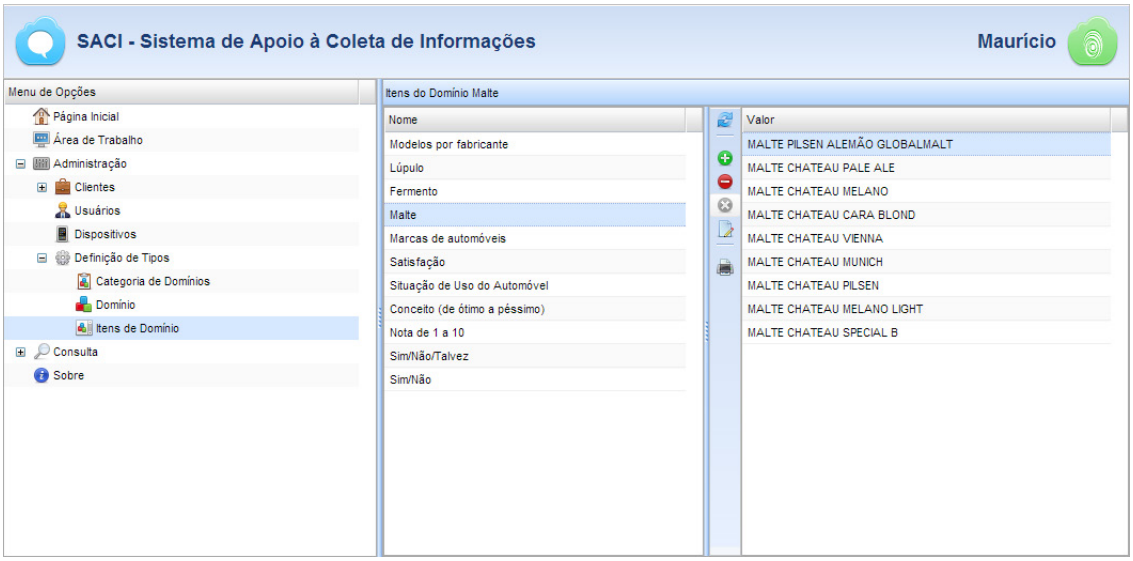
Figura 13 – Tela de configuração dos domínios



Fonte: Produzido pelo autor

Para os domínios do tipo lista ou lista vinculada, é necessário especificar a lista de valores possíveis para a resposta. Essa configuração é realizada na tela ilustrada na Figura 14.

Figura 14 – Tela de configuração dos ítens de domínio



Fonte: Produzido pelo autor

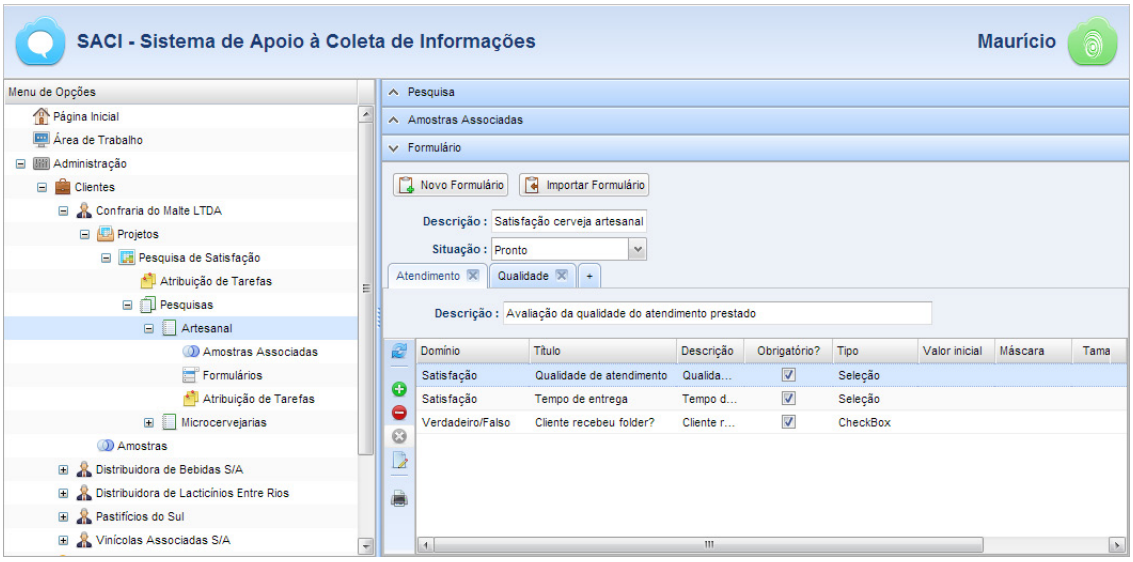
5.2 Configuração de Projeto

Os projetos são definições de um cliente para um conjunto de pesquisas com objetivos comuns, porém com abordagens diferentes em função do segmento, faixa de renda, etc.

Cada pesquisa tem seu próprio formulário, que pode ser construído com base em um formulário já existente ou não. O formulário a ser aplicado é definido na tela ilustrada na Figura 15, onde são definidas as perguntas, seus domínios e agrupamentos.

A pesquisa pode precisar de alterações no seu formulário, isso é feito criando-se um novo formulário, podendo ou não aproveitar o anterior. Todos os formulários que já foram associados com a pesquisa podem ser visualizados na tela de formulários, que pode ser acessada no submenu da própria pesquisa.

Figura 15 – Tela de configuração da pesquisa



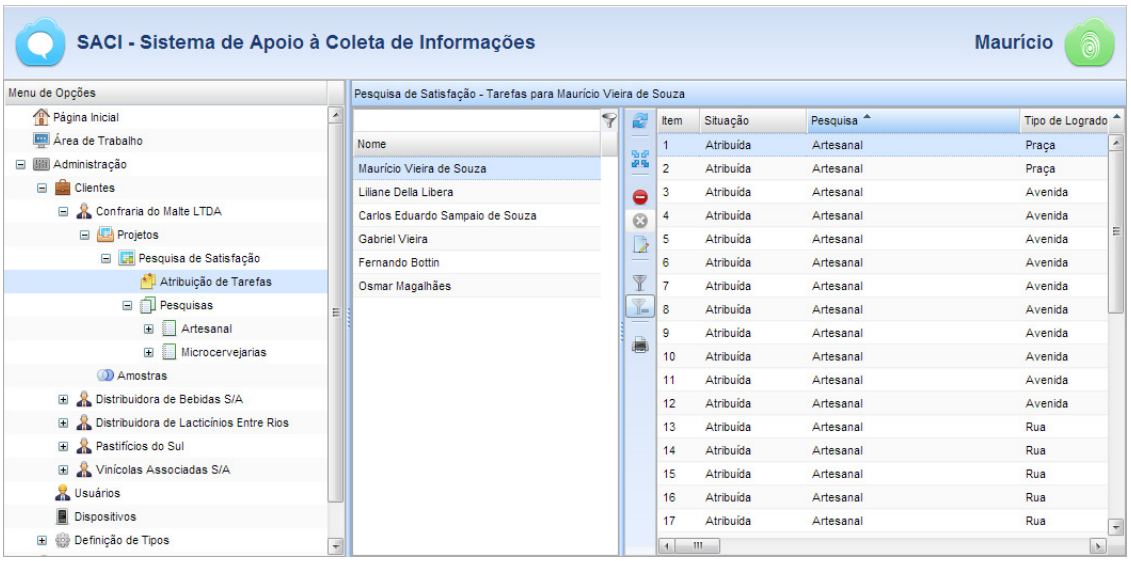
Fonte: Produzido pelo autor

5.3 Atribuição de Tarefas

Quando uma pesquisa é configurada, também são indicadas quais amostras a mesma pode utilizar para a atribuição de tarefas. Os alvos disponíveis para a geração das tarefas são os que fazem parte das amostras associadas a pesquisas de um determinado projeto.

Depois de selecionados os alvos, o sistema gera automaticamente as tarefas, verificando os formulários corretos para cada pesquisa que os alvos estão associados. A atribuição de tarefas é realizada na tela ilustrada na Figura 16.

Figura 16 – Tela de atribuição de tarefas

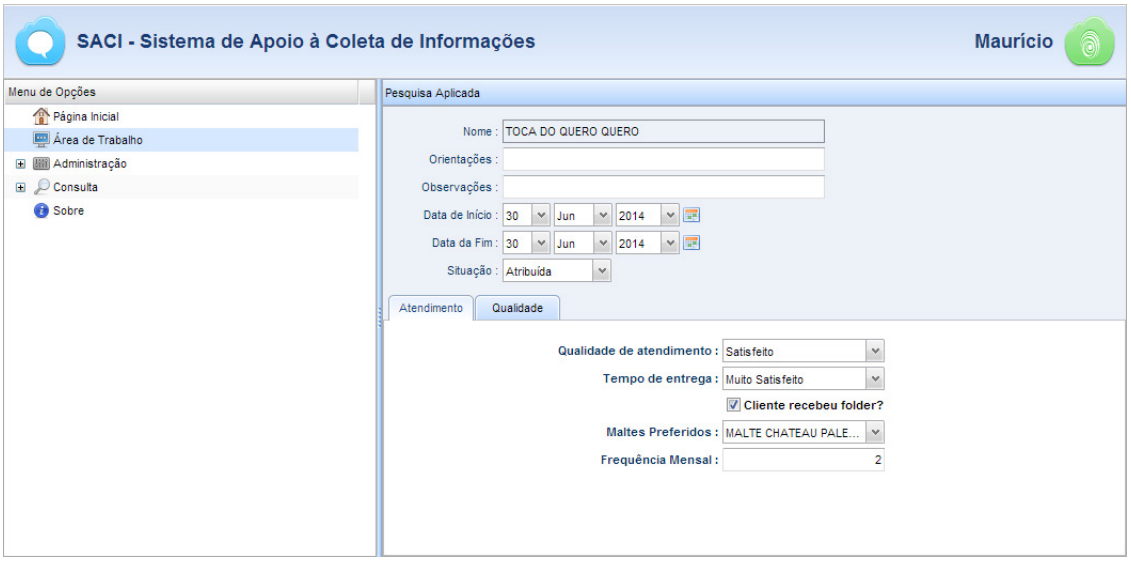


Fonte: Produzido pelo autor

5.4 Verificação de Tarefas

O módulo *Web* conta também com um módulo de execução e verificação de tarefas, ilustrado na Figura 17. O principal objetivo dessa funcionalidade é a verificação das entrevistas e pequenas correções quando for necessário.

Figura 17 – Tela de execução do formulário na emphWeb



Fonte: Produzido pelo autor

5.5 Menu de Opções

A árvore de opções é construída de forma dinâmica, onde cada nodo, quando expandido, traz as opções dinâmicamente de acordo com o perfil e conteúdo existente, mantendo desta forma, somente as opções já navegadas. Como pode ser visto na Figura 16, os clientes, projetos e pesquisas são listados diretamente no menu para facilitar a navegação.

5.6 Execução das Tarefas

Um dos problemas enfrentados é o fato do sistema atual não possuir nenhum suporte a *LBS*, tornando difícil a organização dos entrevistadores. Na solução desenvolvida, as tarefas podem ser exibidas em forma de lista, ilustrado na Figura 18, ou em forma de mapa como ilustrado na Figura 19.

A exibição, em forma de mapa, mostra a localização de todas as tarefas, juntamente com a localização atual do entrevistador(ponto azul na figura), facilitando muito o uso do sistema já que não necessita de nenhum material externo.

Figura 18 – Tela de tarefas

Fonte: Produzido pelo autor

Figura 19 – Tela de tarefas *LBS*

Fonte: Produzido pelo autor

O usuário pode interagir diretamente com os objetos do mapa ou com os itens da lista para acessar a tela de detalhes da tarefa, ilustrada na Figura 20.

Na tela de detalhes da tarefa é possível iniciar a entrevista ou indicar que a mesma não pode ser realizada, fornecendo o motivo. Também são exibidos o mapa com a localização do alvo, orientações para o entrevistador e todos os detalhes que não aparecem na listagem.

Depois de iniciar a entrevista, o sistema vai para a tela onde o formulário dinâmico seja respondido. Essa tela é ilustrada na Figura 21 e representa o mesmo formulário ilustrado na Figura 17, mas executado no *Android*. Os grupos do formulário podem ser navegados utilizando *Swipe Views*[1], o que torna a navegação muito mais fluída e agradável.

Figura 20 – Tela de detalhes

Fonte: Produzido pelo autor

Figura 21 – Tela do formulário

Fonte: Produzido pelo autor

[1] *Swipe Views* permitem que o usuário mova-se eficientemente de um item para o outro utilizando um gesto simples, tornando a navegação mais natural.

6 Conclusão

Este trabalho procurou resolver os problemas apontados no seção 2.2, proporcionando uma interface padrão para o gerenciamento, de forma que toda e qualquer atividade é realizada exclusivamente através da interface do sistema, evitando qualquer interação direta com a infraestrutura como praticado atualmente.

O módulo *Android* foi desenvolvido seguindo diversas recomendações sobre usabilidade e de acordo com as necessidades da empresa, tornando o processo de coleta de informações mais agradável e eficiente. Devido à utilização de *LBS*, o processo realizado pelo entrevistador, na hora de escolher aonde ir, foi bastante simplificado, aumentando a produtividade em campo.

Foi relatado pelo especialista do cliente-alvo que a usabilidade do sistema desenvolvido superou as expectativas e é muito superior a do sistema atualmente em uso, tanto em facilidade de uso, aparência e adaptabilidade.

O sistema desenvolvido nesse trabalho continua em fase de desenvolvimento, sendo necessário implementar as três formas de sincronização e realizar testes com usuários (entrevistadores e gestores).

Como trabalhos futuros, estão a possibilidade de roteirizar os alvos a serem pesquisados, a criação de um dashboard no cliente *Android* e no cliente *Web*, a adição de outros papéis na interação com o sistema e a geração de relatórios.

Referências

BLOCH, J. *Effective java (the java series)*. [S.l.]: Prentice Hall PTR, 2008. Citado na página 36.

CHAGANTI, P. *Google Web Toolkit: GWT Java AJAX Programming: a Practical Guide to Google Web Toolkit for Creating AJAX Applications with Java*. [S.l.]: Packt Publishing Ltd, 2007. Citado na página 28.

CHANDEL, S. Using gwt with hibernate. *Google Web Toolkit*, 2009. Citado na página 40.

CHRISTIAN, B.; GAVIN, K. *Java Persistence with Hibernate*. [S.l.]: USA, Manning, 2006. Citado 3 vezes nas páginas 31, 32 e 38.

ERL, T. *Service-oriented architecture (SOA): concepts, technology, and design*. [S.l.]: Prentice Hall Englewood Cliffs, 2005. Citado na página 37.

FARIAS, F. M.; KUHN, C. Estudo e avaliação do processo de desenvolvimento em android. 2013. Citado na página 34.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, 2000. Citado na página 33.

FRATERNALI, P.; ROSSI, G.; SÁNCHEZ-FIGUEROA, F. Rich internet applications. *Internet Computing, IEEE*, IEEE, v. 14, n. 3, p. 9–12, 2010. Citado na página 35.

FREEMAN, E. et al. *Head first design patterns*. [S.l.]: "O'Reilly Media, Inc.", 2004. Citado na página 32.

GARRETT, J. J. et al. Ajax: A new approach to web applications. 2005. Citado na página 27.

GOOGLE. *Dashboards | Android Developers*. 2014. Disponível em: <http://developer.android.com/about/dashboards/index.html>. Citado na página 42.

GSMARENA. 2014. Disponível em: <http://www.gsmarena.com/>. Citado na página 43.

JUNIOR, M. A. O.; PIMENTA, M. S. Engenharia de software para aplicativos móveis: Desafios, alternativas e direções. 2013. Citado na página 34.

KEREKI, F. *Essential GWT: building for the web with Google Web toolkit 2*. [S.l.]: Pearson Education, 2010. Citado na página 27.

KOTLER, P.; KELLER, K. L. Administração de marketing. São Paulo, 2006. Citado na página 21.

MEIER, R. *Professional Android 4 application development*. [S.l.]: John Wiley & Sons, 2012. Citado na página 29.

RODRIGUEZ, A. Restful web services: The basics. *Online article in IBM DeveloperWorks Technical Library*, v. 36, 2008. Citado na página 33.

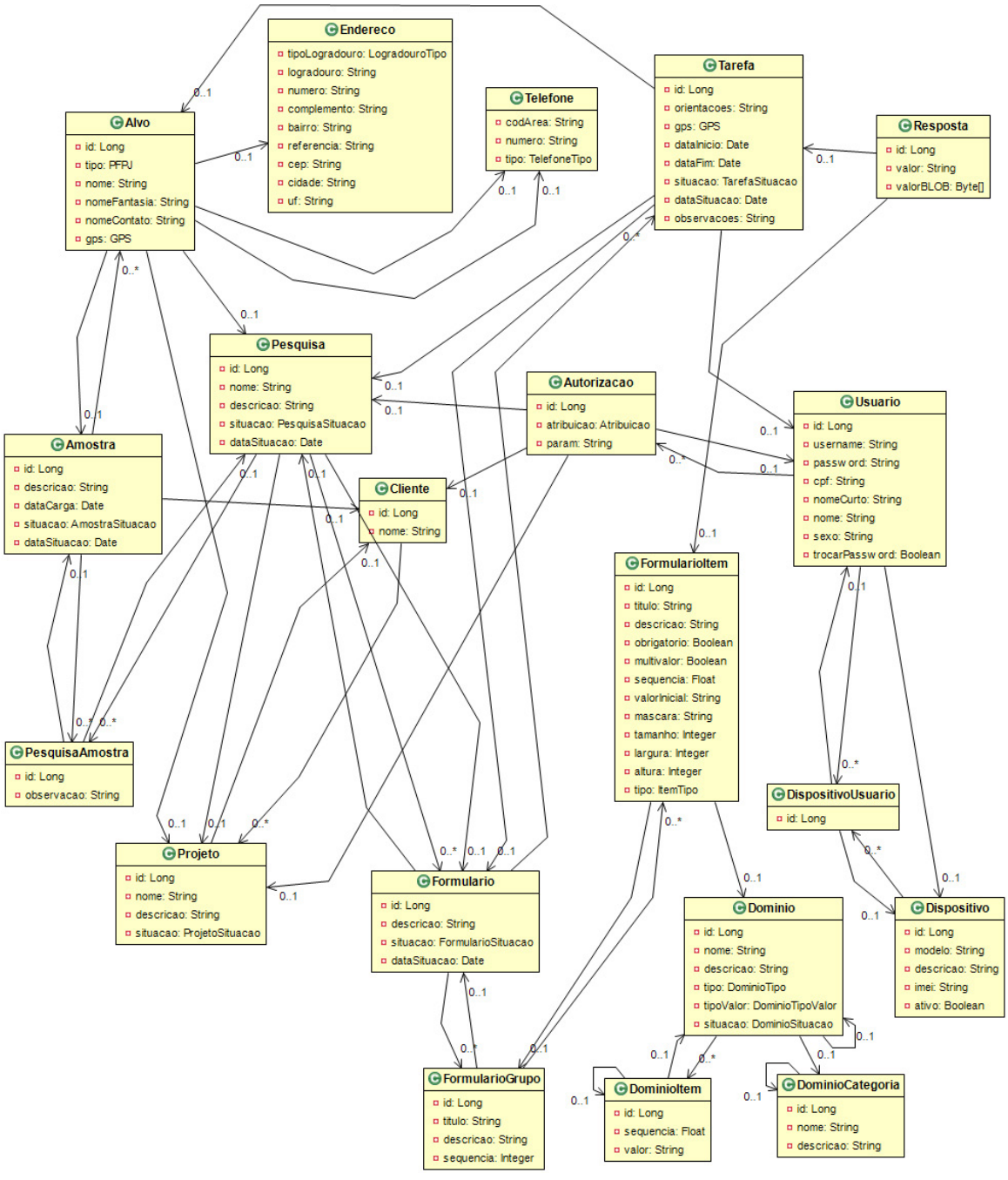
SCHILLER, J.; VOISARD, A. *Location-based services*. [S.l.]: Elsevier, 2004. Citado 2 vezes nas páginas 30 e 31.

TAVARES, P. Z. Estudo de usabilidade para pdas utilizados em coleta de dados nas entrevistas pessoais para pesquisas domiciliares. 2011. Citado na página 34.

APÊNDICE A – Descrição das Entidades

Neste apêndice são apresentados o modelo (Figura 22) e a descrição das classes de negócio (Tabela 3) ilustradas na Figura 9.

Figura 22 – Diagrama de classes das entidades



Fonte: Produzido pelo autor

Tabela 3 – Descrição das entidades

Entidade	**Descrição**
dominio	Domínio que uma resposta pode assumir.
dominio_item	Itens pertencentes a domínios do tipo lista ou lista vinculada.
dominio_categoria	Categorias que tem como objetivo organizar os domínios existentes.
formulario	Formulário que será utilizado para a realização de entrevistas.
formulario_item	Item de formulário que engloba a pergunta, seu domínio e forma de exibição.
formulario_grupo	Grupos utilizados para organizar o formulário, agrupando seus itens.
alvo	Alvo o qual a entrevista deve ser aplicada, contendo endereço, telefone, localização geográfica, nome do contato, etc.
amostra	Conjunto de alvos.
pesquisa	Segmentação de um projeto.
pesquisa_amostra	Associação de amostras com pesquisas.
resposta	Resposta para um determinado item de formulário e tarefa.
usuario	Usuário utilizador do sistema.
dispositivo	Dispositivo móvel utilizado para realização das entrevistas.
dispositivo_usuario	Associação entre usuários e dispositivos que podem ser utilizados.
projeto	Projeto de coleta de informações associado ao cliente.
cliente	Cliente para o qual se realiza entrevistas.
autorizacao	Atribuição que o usuário possui em determinada área do sistema.

Fonte: Produzido pelo autor