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

MARCELO AFANACI JUNIOR

AGREGUEI: SISTEMA AGREGADOR DE OFERTAS PARA PLATAFORMA ANDROID

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2012

MARCELO AFANACI JUNIOR

AGREGUEI: SISTEMA AGREGADOR DE OFERTAS PARA PLATAFORMA ANDROID

Monografia de Especialização apresentada ao Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Especialista em Tecnologia Java”.

Orientador: Prof. Alexandre Reis Graeml

CURITIBA

2012

TERMO DE APROVAÇÃO

AGREGUEI: SISTEMA AGREGADOR DE OFERTAS PARA PLATAFORMA ANDROID

por

MARCELO AFANACI JUNIOR

Esta monografia foi apresentada em 27 de julho de 2012 como requisito parcial para a obtenção do título de Especialista em Tecnologia Java – Departamento Acadêmico de Informática – Universidade Tecnológica Federal do Paraná. O candidato apresentou o trabalho para a Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
ALEXANDRE REIS GRAEML
Prof. Orientador

_____
JOÃO ALBERTO FABRO
Membro da Banca Avaliadora

_____
ADOLFO GUSTAVO SERRA SECA NETO
Membro da Banca Avaliadora

AGRADECIMENTOS

A Deus, primeiramente pelo dom da vida, sabedoria e infinita motivação.

A minha mãe Maria José Afanaci, pois se não fosse por ela, nada disso seria possível.

A minha namorada Vanely, por me acompanhar em cada passo dessa caminhada com muito amor, carinho e dedicação.

Ao professor Alexandre Reis Graeml pela orientação, atenção e paciência durante o desenvolvimento deste trabalho.

RESUMO

AFANACI JR, Marcelo. **Agreguei:** Sistema agregador de ofertas para plataforma Android. 2012. Monografia (Especialização em Tecnologia Java) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

O Comércio eletrônico vem ganhando espaço a cada dia entre os consumidores brasileiros e contribuindo no aumento do lucro das empresas. Aproveitando esse cenário animador um antigo modelo de negócio se fortaleceu novamente, os sites de compras coletivas. Esse projeto tem por objetivo apresentar como se deu o surgimento do modelo de compras coletivas, como foi inserido no panorama do comércio eletrônico brasileiro, seu crescimento e as novas necessidades desse modelo. Serão apresentadas as tecnologias utilizadas para desenvolver a solução proposta para o problema de excesso de informação gerada pelos sites de compras coletivas, os agregadores de ofertas. Por fim, mas não menos importante, é mostrada a aplicação resultante do estudo juntamente com a explicação de como foi pensada sua arquitetura.

Palavras-chave: Compra Coletiva, Compra em Grupo, Agregador de Oferta, Android.

ABSTRACT

AFANACI JR, Marcelo. **Agreguei:** sistema agregador de ofertas para plataforma Android. 2012. Monografia (Especialização em Tecnologia Java) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

The electronic commerce is growing every day among Brazilians consumers; it helps to increase the companies' profit. Taking advantage of the good scenario an old business model became strong once again, the group buying sites. This monograph aims to show how the group buying model has been invented, how it works on the Brazilians electronic commerce, his expansion and the new needs that it has generated. The technologies that were used to develop the proposed solution to the excessive information problem created by collective buying, the deal aggregators. Lastly but not least important, the application that has been created based on the research performed and the explanation about its architecture will be showed.

Keywords: Collective Buying, Group Buying, Deal Aggregator, Android.

LISTA DE FIGURAS

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AOP	Aspect Oriented Programming
APK	Android Application Package
DEX	Dalvik Executable
DI	Dependency Injection
FD	Functional Description
FTP	File Transfer Protocol
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
IOC	Inversion of Control
JDBC	Java Data Base Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
OpengGL ES	Open Graphic Library for Embedded Systems
ORM	Object Relational Mapping
OXM	Object XML Mappers
POM	Project Object Model
SGL	Scalable Graphics Library
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SSL	Security Socket Layer
UML	Unified Modeling Language
URI	Uniform Resource Identifier
WSD	Web Service Descriptor
WSDL	Web Service Definition Language
XML	Extensible Markup Language
XSD	XML Schema Definition

LISTA DE ACRÔNIMOS

EUA	Estados Unidos da América
OHA	Open Handset Alliance

SUMÁRIO

1 INTRODUÇÃO

1.1 SURGIMENTO DO MODELO DE NEGÓCIO PARA COMPRA COLETIVA

No final dos anos 90 começaram a surgir *sites* de leilões, que ofereciam a oportunidade de interação entre as empresas e seus clientes por meio da oferta de produtos que poderiam ser arrematados pelo melhor lance. Os pioneiros nesse mercado foram os sites *Mobshop*.com *e Mercata.com,* que geravam seu lucro intermediando as negociações entre os clientes e as empresas.

A *Mercata* começou suas atividades em Bellevue, Washington, em maio de 1999, oferecendo computadores, jóias, aparelhos domésticos entre outros artigos. Seu *slogan* principal era: “Quanto mais compradores, menor o preço” e suas ofertas de produtos ocorriam durante o que se chamava de “Ciclo de leilão” (KAUFFMAN e WANG, 2002). Entretanto, como é possível ver na figura 1, a falta de detalhes sobre os produtos ofertados causava grande insegurança nos clientes.

Figura 1 – Produto oferecido pela Mercata em seu *site* de compras coletivas
Fonte: (KAUFFMAN e WANg mavvvenG, 2002).

O *MobShop* por sua vez começou suas atividades com sede em São Francisco, Califórnia, em maio de 1999, inicialmente oferecendo computadores, PDAs, eletrônicos, *softwares,* filmes e jogos de computador. Utilizando uma abordagem chamada de “Ciclos de Compra”, em que os clientes faziam lances e havia uma certa flexibilidade em relação à quantidade de um produto específico

existente para venda. O principal diferencial entre o *MobShop* e o *Mercata* era a quantidade de informação em relação ao produto, o *MobShop* oferecia mais detalhes sobre o produto em negociação permitindo ao cliente saber se realmente a oferta atendia as suas expectativas deixando-o mais seguro para realizar a transação (KAUFFMAN e WANG, 2002).

Por serem os primeiros *sites* a atuar nesse nicho de comércio, realizaram vários testes quanto à forma de interação entre empresas e possíveis clientes, a fim de descobrir o modelo de negócio com maior apelo aos clientes e que gerasse maior lucro às empresas.

Os modelos testados durante o desenvolvimento do conceito de “compra coletiva” ou “compra em grupo”, em nada se parecem com o modelo de leilão chamado *dutch auction* ou *descending price auction* que envolvia um mecanismo em que o preço, inicialmente alto, começava a ser reduzido, até que atingisse um patamar em que algum participante do leilão se interessasse pelo item a venda e o adquirisse (KAUFFMAN e WANG, 2002).

Lai (2002) propôs um número de modelos genéricos relacionados à compra coletiva norteados por cinco dimensões chave: o iniciador, a variedade de produtos, a quantidade de vendedores, o tamanho do grupo de compra coletiva e, por fim, as condições em que determinado produto está sendo oferecido. Essas dimensões são importantes para os modelos de negócios de compra coletiva mais aplicados atualmente.

Tomando como exemplo a dimensão do “iniciador”, sabe-se que é parte interessada em satisfazer suas necessidades, sejam elas de consumo (cliente) ou de lucro (empresa). Mas, ao contrário do que comumente acontece, um cliente pode ser o agente de criação da necessidade da oferta, pois ele pode entrar em contato com a empresa que oferece o produto/serviço de seu interesse e negociar um preço abaixo do mercado para um determinado número de compradores, gerando uma demanda considerável e permitindo que a empresa possa baixar o preço de seu produto sem perder a margem de lucro, pois, conforme a quantidade aumenta, o custo de fabricação diminui. Logo, é possível oferecer um preço menor, em muitos casos, com um lucro superior. Pela perspectiva da dimensão do iniciador, a empresa consegue identificar quem foi o gerador da demanda e qual o esforço de produção necessário para suprir essa demanda (DOONG et al, 2009).

Conforme novas empresas foram entrando no mercado de compra coletiva os modelos genéricos de negócio inicialmente identificados foram tornando-se mais formais, por serem dirigidos por mecanismos bem definidos de funcionamento (DOONG et al, 2009). Dentre esses mecanismos, destacam-se os seguintes:

- Mecanismo tradicional: uma quantidade definida de um mesmo produto é ofertada até que se esgote ou o fornecedor deseje aumentar a produção para que possa suprir uma demanda crescente. Normalmente esse mecanismo implica em se ter uma quantidade mínima de produtos vendidos para que a oferta passe a ser válida, e os produtos são ofertados durante um período de tempo. Caso a quantidade mínima de vendas não seja atingida para tornar a oferta válida ela é cancelada (DOONG et al, 2009).

- Mecanismo de tempo: quanto antes o consumidor comprar um item da oferta, maior será sua taxa de desconto. Por exemplo, um cliente que comprou um produto no primeiro dia da oferta paga R$200,00 enquanto que aqueles que fizerem a compra no segundo dia pagam R$250,00. O principal objetivo desse mecanismo é gerar a chamada “compra por impulso”, pois quanto mais barato o cliente paga, maior é sua sensação de recompensa e sua percepção de valor em relação ao produto/oferta (DOONG et al, 2009).

- Mecanismo de sequência: o desconto é oferecido em relação à ordem dos compradores da oferta. Por exemplo, em cliente que foi o quinto a comprar determinada oferta se enquadra no lote dos cem primeiros compradores e, por isso, recebe um desconto de 30%, o próximo lote de compradores, do comprador 101 a 300, recebe apenas 20% de desconto. Essa abordagem é menos agressiva que a imposta pelo mecanismo de tempo, pois caso a oferta não seja tão procurada mas exista o número mínimo de compradores para torná-la válida, o momento em que o cliente realiza a compra não importa, mas sim estar no primeiro lote (DOONG et al, 2009).

- Mecanismo baseado em quantidade: nesse mecanismo o desconto oferecido é inversamente proporcional a quantidade de produtos comprados, ou seja, esse mecanismo influencia diretamente o cliente a comprar mais produtos para que o valor de desconto atingido seja maior (DOONG et al, 2009).

Atualmente os *sites* de compra coletiva encontrados na Internet se baseiam tanto nas cinco dimensões chaves de Lai (2002), quanto nos mecanismos de funcionamento expostos anteriormente.

1.2 PANORAMA DO *E-COMMERCE* BRASILEIRO

Desde a metade dos anos 90, quando o *e-commerce* começou a despontar como um modelo de negócio dinâmico, adaptável e lucrativo, os consumidores brasileiros vêm aumentando a prática de compras pela Internet.

A facilidade de encontrar o que se precisa sem sair de casa ou do trabalho se torna uma opção muito atraente. O comportamento do consumidor envolve atividades físicas, mentais e emocionais por meio das quais eles fazem compras e usos de produtos para satisfação de necessidades e desejos (GADE, 2000). Assim, com alguns poucos cliques, se encontra um leque variado de produtos que vai desde roupas, comidas e móveis, até carros. Quando essa prática de compras é positiva, ela se intensifica (TOLEDO e CARO, 2006).

Pode-se encontrar no comércio eletrônico alguns benefícios como: possibilidade e facilidade de comparação, deixando preços dos produtos e serviços mais baratos; escolha entre vários fornecedores, aumentando a diversidade das compras; comodidade de comprar a qualquer hora; obtenção de detalhes dos produtos de forma rápida; personalização de produtos e interação com outros consumidores (TURBAN et al, 2004).

A receita anual gerada pelo *e-commerce* é reflexo direto do aumento da confiança e popularidade desse modelo de negócio entre os brasileiros, como se pode observar no Gráfico 1, a seguir.

Gráfico 1 – Faturamento anual do e-commerce no Brasil (em bilhões de reais)
Fonte: www.e-commerce.org.br/stats.php

Tal crescimento, explica o fato do Brasil ser atualmente o 6º maior mercado *online* do mundo e as compras *on-line* terem avançado, em 2011, 26% em relação ao ano de 2010[1]. Um fato relevante que deve ser considerado para explicar o crescimento do comércio eletrônico, é que o Brasil ocupa a terceira posição mundial em número de usuários ativos na Internet, totalizando 78,5 milhões de pessoas conectadas em média 47 horas mensais. A classe C se tornou responsável por 23% das compras online[2].Para 2012 as projeções continuam otimistas, porém um pouco cautelosas, prevendo um faturamento de R$ 30 bilhões. Entretanto, pelo fato de o faturamento de R$ 18,7 bilhões (2011) conquistado pelo varejo eletrônico, ter deixado a taxa de crescimento abaixo dos 30% previstos[3], essa projeção pode não se sustentar. Uma previsão mais conservadora considera um faturamento de R$ 22 bilhões viável para o ano de 2012.

1.2.1 Surgimento e expansão de *sites* de compras coletivas

Devido aos benefícios apresentados pelas compras *online*, houve um aumento significativo nas vendas pela Internet. Assim, surgiu a opção de explorar esse mercado de diversas formas, dentre as quais destacam-se os *sites* de compras coletivas, conquistando consumidores e revolucionando a maneira de vender pela Internet. Esses *sites* tomaram como base os modelos de negócios criados no final dos anos 90, norteados pelos mecanismos tradicionais de funcionamento (explicados na seção 1.1) e os adaptaram às necessidades dos consumidores atuais, buscando vender o produto/serviço ao maior número de pessoas interessadas, com um preço reduzido. Vendem cupons de desconto, que só passam

---

[1] COMPUTERWORLD. **Brasil deverá ser 4º maior em *e-commerce* até 2015, diz estudo.** Disponível em: <http://computerworld.uol.com.br/negocios/2011/12/28/brasil-devera-ser-4o-maior-em-e-commerce-ate-2015-diz-estudo/>.

[2] IBOPE. **Comércio eletrônico - avaliação 360º.** Disponível em: <http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=Internet&docid=0AADE4C29DC53A458325796D0050A2DD>.

[3] E-Commerce News. **E-Commerce cresce 40% no Brasil em 2010.** Disponível em: <http://ecommercenews.com.br/noticias/pesquisas-noticias/e-commerce-cresce-40-no-brasil-em-2010>.

a ser válidos depois que atingem o número mínimo estabelecido para a oferta em questão.

Esse tipo de negócio, como conhecido hoje, foi criado nos Estados Unidos em 2008 com o lançamento do site Groupon, cujo nome é uma junção das palavras "*group*" e "*coupon*", idealizado por Andrew Mason (GALO, 2010). No Brasil, o pioneiro foi o Peixe Urbano, que iniciou suas atividades em 2010. Depois surgiram os sites "ClickOn" e "Imperdível" (GALO, 2010).

Os sites de compras coletivas tornam-se muito interessantes, ao passo que atendem desde empresas renomadas até pequenos estabelecimentos dispostos a oferecer seus produtos com desconto a um grande público. Em abril de 2010, um mês após a criação do primeiro *site* de compras coletivas no Brasil, existiam apenas 10 *sites*. Já em dezembro daquele mesmo ano, o número já chegava a 400. Em janeiro de 2011 havia 1025 *sites* e os últimos dados divulgados em abril de 2012, registravam 1890 *sites* de compras coletivas e 73 *sites* agregadores de ofertas[4], como é possível visualizar no gráfico 2, a seguir.

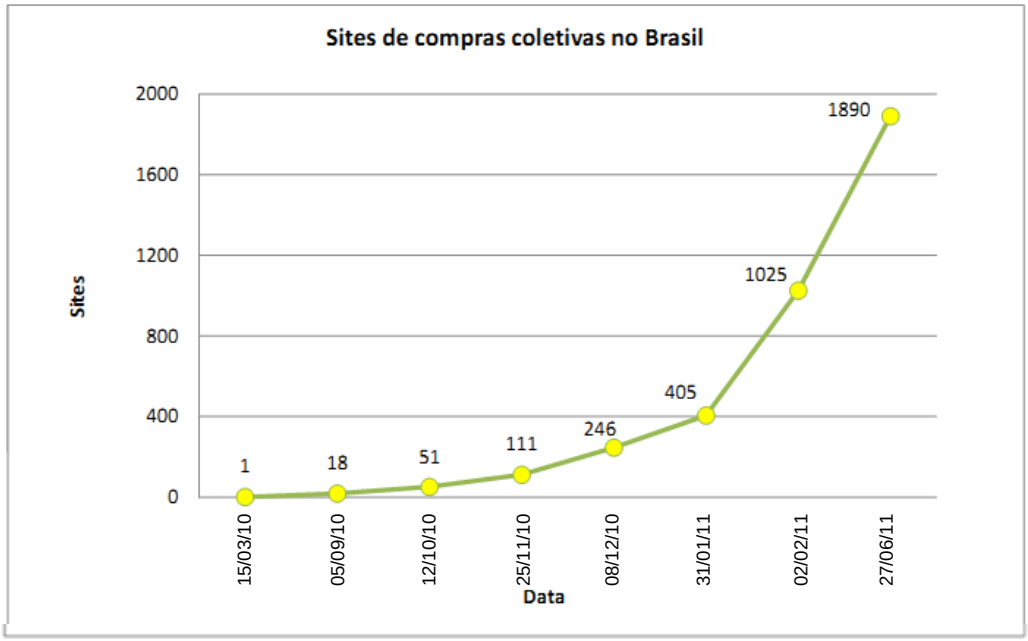

Gráfico 2 – Sites de compras coletivas no Brasil
Fonte : http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/

[4] BOLSA DE OFERTAS. **Brasil tem quase 2000 sites de compras coletivas.** Disponível em: <http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/>

Entretanto, um minucioso levantamento realizado pelo *site* Bolsa de Ofertas, que é especialista neste tipo de segmento, dividiu os *sites* de ofertas nas seguintes categorias: *sites* ativos, inativos, novos, danificados e desativados, como mostra a Tabela 1.

Status	Descrição	Total	Porcentagem
Ativo	*Site* de acesso normal, com ofertas do dia e em pleno funcionamento.	1145	61%
Inativo	*Site* que já apresentou ofertas mas está parado sem alterações a mais de 15 dias.	326	17%
Novo	*Site* em fase de registro de usuários para envio de alertas das primeiras ofertas.	212	11%
Danificado	*Site* que no acesso mostra problemas no *script* (HTML)	121	6%
Desativado	*Site* que já esteve ativo, com links no Twitter e Facebook, mas que não consegue mais o acesso.	86	5%

Tabela 1 – Levantamento realizado pelo site Bolsa de Ofertas sobre os sites de ofertas. Fonte : Bolsa de Ofertas - http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/

Ainda, em se tratando de agregadores de ofertas, existem no total setenta e três, dos quais sessenta e sete estão ativos, quatro inativos, um danificado e um desativado. Há, portanto, somando-se os *sites* de compras coletivas e os agregadores, 1963 *sites* voltados a compras coletivas, um número que retrata a rápida expansão do segmento, se considerado que a primeira ocorrência de um serviço nesses moldes no Brasil ocorreu em 2010[5].

Com esse aumento nas vendas de produtos por meio de compras coletivas abre-se a oportunidade para novas aplicações, como por exemplo: os agregadores de ofertas, cujo pioneiro no Brasil foi o "*SaveMe*", que disponibilizam de maneira

[5] BOLSA DE OFERTAS. **Brasil tem quase 2000 sites de compras coletivas.** Disponível em: <http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/>

simples as ofertas de diversos *sites* em um só lugar, visando facilitar o acesso pelos usuários, que não precisam acessar inúmeros *sites* para encontrar as ofertas disponíveis[6].

O usuário deste tipo de serviço tem a facilidade de encontrar o que precisa entre vários fornecedores em um só lugar, conseguindo chegar até a oferta principal e efetuar suas compras com descontos de maneira simples. Tendo em vista que as ofertas disponíveis tem tempo limite para compra, os agregadores de ofertas apresentam comodidade e economia de tempo para o consumidor.

1.2.2 Tendências do e-commerce no Brasil – segmento de compras coletivas

Pode-se observar no segmento de compras coletivas as seguintes tendências:

- Acomodação do mercado com a profissionalização dos *sites* já existentes, e o surgimento de novos *sites*, cada vez mais segmentados e regionalizados[7].
- Aparição de novos agentes intermediários, concentradores de ofertantes e clientes (TORRES, 2012).
- Adequação das lojas com vendas *on-line*, pois, segundo pesquisa feita nos EUA pela Forrester Research, até 2016 as vendas *online* utilizando plataformas móveis podem representar mais de 7% do total das vendas de uma loja. Além de ser um número expressivo, isto indica um perfil totalmente distinto dos consumidores que utilizam computadores ou *tablets*[8].

---

[6] BOLSA DE OFERTAS. **Brasil tem quase 2000 sites de compras coletivas.** Disponível em: <http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/>

[7] E-COMMERCE GUIDE. **Tendências do E-Commerce em 2012 no Brasil.** Disponível em: <http://www.e-commerceguide.com.br/2011/12/tendencias-do-e-commerce-em-2012-no.html>.

[8] IBOPE. **Comércio eletrônico - avaliação 360º.** Disponível em: <http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=Internet&docid=0AADE4C29DC53A458325796D0050A2DD>.

1.3 PROBLEMA

Atualmente encontramos mais de 1890 *sites* que disponibilizam ofertas diariamente, gerando uma grande quantidade de informação para seus usuários que na maioria dos casos, não será completamente absorvida, ou seja, as ofertas podem não chegar a quem as empresas ou comerciantes gostariam, simplesmente pelo fato de que os utilizadores do serviço são fiéis a três ou quatro *sites* de ofertas.

Como tentativa de contornar esse problema, surgiram os agregadores de ofertas que permitem aos usuários receberem ofertas pertinentes à localidade desejada. Existem mais de setenta agregadores de ofertas para plataforma *web* e três voltados para a plataforma móvel. Porém, as soluções existentes não impedem que o usuário receba uma grande quantidade de informações irrelevantes às suas preferências.

1.4 JUSTIFICATIVA

A grande quantidade de informações gerada pelos sites e agregadores de ofertas não são absorvidas em sua totalidade pelos usuários desses serviços, essa situação demonstra um ponto falho no que se diz respeito a como atingir exatamente o público alvo de sua oferta.

A solução proposta visa filtrar as informações que serão exibidas ao usuário de acordo com suas preferências e interesses. Dessa maneira será possível aos clientes atingirem com maior precisão seu público alvo, enquanto que aos usuários serão enviadas ofertas relevantes a suas preferências.

Da gama de sistemas operacionais existentes para dispositivos móveis o *Android* foi escolhido, pois se destaca pela facilidade de desenvolvimento e integração com diversas plataformas móveis ou não, maturidade, interfaces ricas e suporte a diversas funcionalidades desejáveis em aplicações móveis como mapas, GPS, etc.

1.5 OBJETIVO GERAL

- Desenvolver um aplicativo agregador de ofertas, para plataforma móvel que utiliza o sistema Android, que permita exibir as ofertas de *sites* de compras coletivas de maneira personalizada, clara e objetiva.

1.6 OBJETIVOS ESPECÍFICOS

- Desenvolver um conjunto de serviços, que possam automaticamente identificar, validar e disponibilizar a oferta para consulta na aplicação.
- Fornecer configurações de preferências diferentes dos modelos já existentes em aplicações agregadoras de ofertas.
- Mostrar apenas ofertas que sejam relevantes para os usuários da aplicação, baseadas em suas escolhas.
- Oferecer serviço de mapas para facilitar a compra das ofertas.

2 REVISÃO TECNOLÓGICA

2.1 SPRING FRAMEWORK

Os nichos onde se utiliza a linguagem Java são os mais variados possíveis, e vão desde tecnologia embarcada utilizada em "*Smart Cards*", até aplicações corporativas que demandam alta disponibilidade. Nesse ponto, inicia-se vislumbrar a motivação existente por trás da criação do *framework* Spring.

Para desenvolver soluções corporativas adequadas, existe um grande volume de elementos que devem ser pensados para gerar uma arquitetura robusta e ao mesmo tempo, coerente com a necessidade do negócio. Mesmo após a arquitetura ter sido criada, existe o desafio de encaixar os elementos desenvolvidos, por exemplo: como integrar a arquitetura de um controlador *web* com a interface de acesso ao banco de dados, quando eles foram desenvolvidos por times diferentes e que, em alguns casos, não mantinham comunicação entre si? (JOHNSON et al, 2011).

Vários *frameworks* conhecidos pelo termo de "*Light Containers*" surgiram para tentar resolver o problema de integração entre componentes de diferentes camadas (FOWLER, 2004). O ponto principal de atuação do *Spring* é fornecer aos desenvolvedores um bloco conciso de funcionalidades, que caso não fossem oferecidas por ele ficariam a cargo de arquitetos e desenvolvedores.

O componente de "*Inversion of Control*" (IoC) existente no *Spring,* endereça cada ponto relevante da estrutura de uma aplicação, ao módulo que conhece como tratar o elemento em questão, de forma isolada e desacoplada do resto do *container* (FOWLER, 2005). Nesse ponto é interessante saber o porquê de se usar o termo *Inversão de Controle*.

2.1.1 Inversão de controle

Para se entender *inversão de controle* em seu mais básico modelo, pode-se fazer o seguinte questionamento: Que aspecto de controle os *containers* leves estão invertendo?

Quando seu código está no controle, por exemplo, eles podem decidir quando executar determinadas atividades, quando receber entradas em relação às

atividades anteriormente executadas e, por sua vez, quando processar os resultados. Nesse modelo apresentado anteriormente não se discute inversão de controle, pois seu código está no controle do fluxo da aplicação. Entretanto, quando um *framework* que provê inversão de controle é utilizado, é ele quem decide quando seus métodos serão executados, baseado nas configurações informadas anteriormente pelo desenvolvedor. Logo, sob esse ponto de vista, existe inversão de controle. Esse conceito é explicado claramente por Johnson e Foote (2005):

> Uma importante característica de um *framework* é que os métodos definidos pelo usuário para servi-lo serão sempre chamados internamente e não através do código da aplicação do usuário. O *framework* sempre atua como o programa principal, coordenando a sequência de atividades da aplicação. Essa inversão de controle provê aos *frameworks* o poder de servirem como esqueletos extensíveis. Os métodos criados pelo usuário juntamente com os algoritmos genéricos definidos pelo framework viabilizam a criação de uma solução em particular (JOHNSON e FOOTE 2005).

A *inversão de controle* é a parte chave para tornar uma biblioteca algo muito diferente de um *framework*, pois uma biblioteca é essencialmente um conjunto de métodos, que podem ser chamados pelo desenvolvedor para que determinada tarefa seja realizada e, após seu término, ocorra retorno do controle para o chamador (FOWLER, 2005).

Uma vez entendido o conceito de inversão de controle, é necessário um nome mais específico para se referir a esse padrão. Assim, ficou convencionado estabelecer como nome do padrão discutido: "*dependency injection*" (DI).

2.1.2 Módulos

O *Spring framework* é distribuído em módulos separados interdependentes ou não. Essa abordagem permite que o desenvolvedor utilize apenas os elementos do *container* que sejam relevantes para criar a infraestrutura da qual o modelo de negócio irá depender. Na Figura 2 pode-se ver explicitamente a divisão dos módulos presentes no *Spring* e sua interdependência.

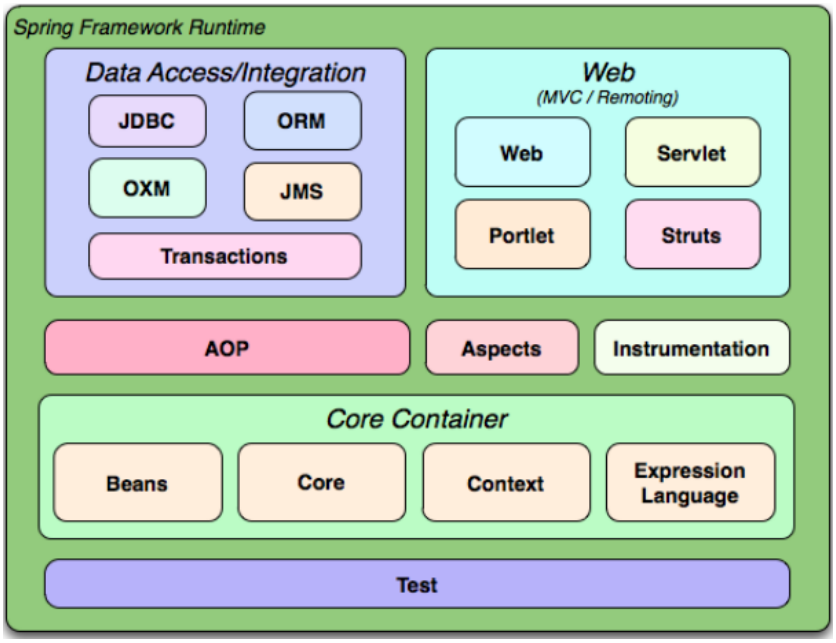

Figura 2 – Divisão dos módulos presentes no *Spring*.
Fonte: Manual de referência do Spring - http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/

Alguns módulos do *Spring* foram relevantes para o desenvolvimento do trabalho que está sendo apresentado e serão explicados a seguir:

- *Core* e *Beans*: Contém as partes fundamentais do *framework*, incluindo recursos de *inversão de controle* e *injeção de dependência.*
- *Context*: Fornece acesso aos objetos presentes no contexto do *framework* de uma maneira muito similar ao que ocorre com JNDI.
- *JDBC*: Fornece uma camada de abstração, que evita o trabalho de codificação na camada de integração com a base de dados.
- *OXM*: Camada de abstração que fornece suporte ao mapeamento Objeto/XML através de implementações como JAXB, Castor, XMLBeans, JiBX e XStream.

2.2 ANDROID

O surgimento do *Android* deu-se devido à necessidade das empresas e desenvolvedores criarem uma plataforma rápida e ágil de desenvolvimento, e à busca constante dos usuários de aparelhos móveis por um celular elegante e com um visual moderno, de fácil navegação e com inúmeros recursos (LECHETA, 2010).

O *Android* foi à resposta oferecida pelo *Google*® para ocupar esse espaço. Trata-se de uma plataforma baseada em Linux, com um ambiente de desenvolvimento bastante poderoso e flexível, que conta com aplicações pré-instaladas. É importante ressaltar que o *Android* não foi concebido exclusivamente pelo *Google*®. Importantes empresas líderes no mercado de telefonia como a LG®, Samsung®, Motorola® entre muitas outras estavam apoiando essa idéia e juntas formaram a OHA (*Open Handset Alliance*)[9].

Seu sistema é baseado no Kernel 2.6.0 do Linux, que é responsável por gerenciar a memória, processos, *threads* e a segurança dos arquivos e pastas, além de redes e *drivers*. Sendo multitarefa, cada aplicação é executada em seu próprio processo no sistema operacional, ou seja, possui uma *thread* dedicada, para evitar conflitos ou violações de segurança[10]. Existe um usuário que é criado de maneira exclusiva, com uma estrutura de arquivos dedicada para execução de suas tarefas. Todos esses recursos tornam o sistema mais seguro e menos suscetível a falhas.

O *Android* possui uma máquina virtual chamada *Dalvik*, que é otimizada para ser executada em dispositivos móveis, focada em economia de memória e recursos de *hardware*, a fim de oferecer um desempenho superior à máquina virtual Java comum. Ainda assim, a *Dalvik* fornece todos os recursos providos pela linguagem Java (LECHETA, 2010).

As aplicações desenvolvidas para o *Android* têm seus *Bytecodes* (*class*) criados normalmente, porém, após serem compiladas são convertidas no formato .dex. Após essa etapa os arquivos .dex, juntamente com arquivos adicionais de

---

[9] OPEN HANDSET ALLIANCE. **Members.** Disponível em: <http://www.openhandsetalliance.com/oha_members.html>.

[10] GOOGLE. **What is Android ?** Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>

mídia ou configurações, são compactados em um único arquivo com a extensão .apk. É nesse momento que a aplicação está pronta para ser distribuída e instalada (LECHETA, 2010).

Para realizar a distribuição de seus aplicativos os desenvolvedores contam com uma plataforma disponibilizada pelo *Google*®, chamada "*Google Play"*. O processo de distribuição através do *Google Play* é relativamente simples. Basta ter um cadastro com perfil de desenvolvedor, concordar com os termos de uso estipulados e pagar uma taxa de registro. Feito isso, o desenvolvedor terá acesso a um console administrativo onde é possível realizar o envio de aplicações, configurar as opções de publicação e monitorar os dados relativos aos aplicativos publicados, tais como: avaliação, comentários, quantidade de *downloads*, entre outros recursos adicionais[11].

2.2.1 Arquitetura

O sistema operacional Android pode ser dividido em cinco seções distintas e bem definidas[12]:

- *Kernel Linux* (v2.6.0): Responsável por oferecer os principais serviços de segurança, gerenciamento de memória e processos, serviços de rede e *drivers*. O *Kernel* atua como uma camada de abstração entre o hardware e o resto da pilha de software.
- Ambiente de execução: Inclui um conjunto de bibliotecas, que fornece a maioria das funcionalidades presentes nas principais bibliotecas da linguagem de programação Java. Todas as aplicações *Android* rodam seus processos com uma instância dedicada da máquina virtual *Dalvik* (presente nesta camada), projetada especificamente para suportar diversas instâncias com a maior economia possível de recursos.

---

[11] GOOGLE. **Publishing on Google Play.** Disponível em: <http://developer.android.com/guide/publishing/publishing.html>

[12] GOOGLE. **What is Android ?** Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>

- Bibliotecas: Conjunto de bibliotecas em C/C++ utilizada por inúmeros componentes do sistema operacional.
- *Framework* de aplicação: Elemento responsável por fornecer todos os recursos necessários para que os desenvolvedores *Android* possam explorar todas as possibilidades oferecidas pelo sistema operacional como: acesso a informações locais, execução em segundo plano de serviços, criação de alarmes, envio de notificações aos usuários, alteração da barra de *status* do aparelho e muitos outros recursos.
- Aplicações: Elementos que contêm as aplicações nativas do *Android* como: calendário, mapas, navegador, contatos, SMS e etc. Qualquer outra aplicação desenvolvida em Java, nativa ou não do sistema operacional, será comportada nessa seção.

A figura 3 representa a divisão atual dos elementos do sistema operacional *Android* e mostra os itens que estão presentes em cada um deles:

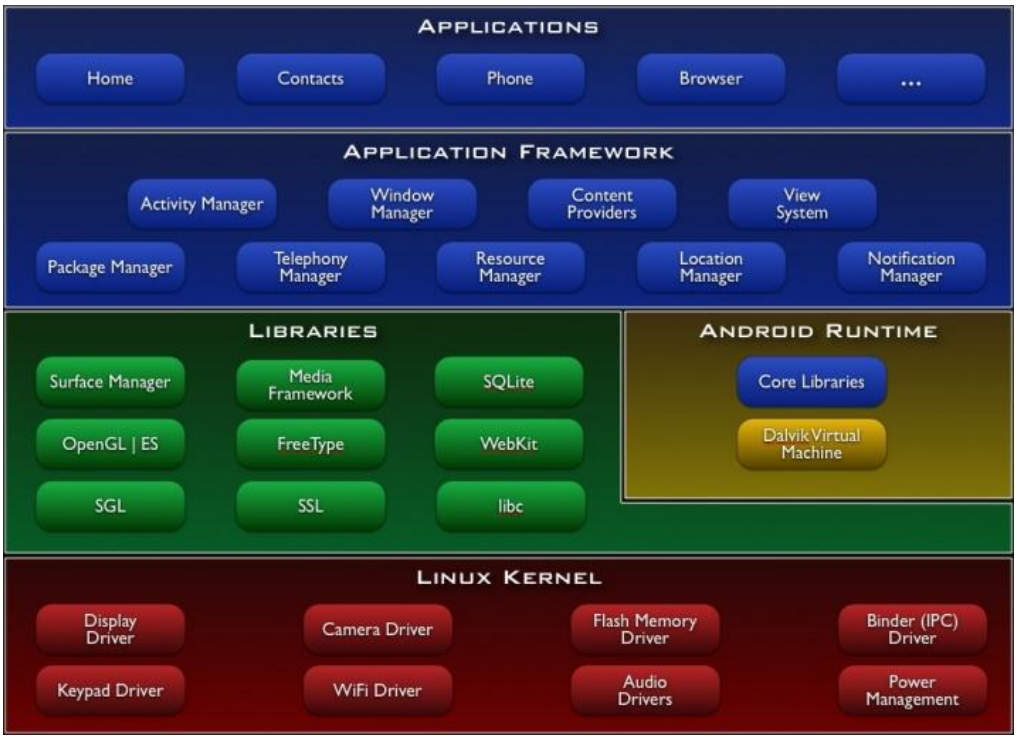

Figura 3 - Divisão atual dos elementos do sistema operacional do Android
Fonte : O que é *Android* - http://developer.android.com/guide/basics/what-is-android.html

2.2.2 Ciclo de vida

O ciclo de vida das aplicações *Android* está intimamente ligado a uma classe chamada *Activity, pois* é ela quem representa cada tela de sua aplicação e possui um ciclo de vida bem definido. Cada *Activity* iniciada é inserida no topo de uma pilha chamada de A*ctivity Stack* ou, simplesmente, pilha de atividades. Sendo assim, sempre que uma nova atividade é inserida no topo da pilha a anterior que estava em execução, fica abaixo da nova (LECHETA, 2010).

A rígida definição do ciclo de vida é importante, pois a partir do momento que uma *activity* está pausada, o sistema operacional pode decidir encerrar o processo para, por exemplo, liberar recursos de memória. Entretanto existem momentos em que as atividades do usuário são interrompidas, por exemplo: o usuário estava jogando, interrompeu suas atividades para atender uma ligação e tem a intenção de voltar a jogar retomando do ponto em que parou. Esse é um caso em que o desenvolvedor da aplicação pode se valer da etapa adequada do ciclo de vida do *Android* para salvar as informações do jogo do usuário em memória, a fim de que ele continue jogando normalmente após finalizar as atividades que apareceram com ordem maior de prioridade na pilha (LECHETA, 2010).

O ciclo de vida completo de uma aplicação *Android* é dividido em três ciclos:

- Ciclo *Entire Lifetime:* é o mais abrangente, pois ele se inicia na criação da *Activity* e se encerra quando ela é destruída. Esse ciclo ocorre apenas uma vez e define o tempo de vida completo de uma *activity*. Apenas a chamada aos métodos *onCreate*() e *onDestroy*() são relevantes. Após realizar a chamada do primeiro método a *activity* encontra-se iniciada podendo ou não estar no topo da pilha de *activities* interagindo com o usuário.
- Ciclo *Visible Lifetime:* ocorre entre os métodos *onStart*() e *onStop*(). Durante o período desse ciclo o usuário pode visualizar a *activity* na tela, mas não necessariamente em primeiro plano. O fato de ela estar na tela não quer dizer que esteja interagindo com o usuário, pois, caso exista a interrupção causada por uma ligação, a atividade ainda estará no topo da pilha, embora tenha cedido lugar a uma atividade prioritária.

- Ciclo *Foreground Lifetime:* define de fato que a *activity* está no topo da pilha e interagindo com o usuário. Mesmo que ela esteja no topo da pilha, frequentemente ela pode alternar entre os estados executando e pausada quando, por exemplo, o celular do usuário entra em modo de espera. Entretanto, assim que o celular sair desse modo, a *activity* do topo da pilha será exibida ao usuário e estará pronta para interação.

Como visto anteriormente, existem diversos métodos que são relevantes em cada ciclo de vida de uma *activity*, pois, é através deles que o desenvolvedor de aplicações *Android* pode controlar o estado de sua aplicação e escolher quais atitudes tomar, frente a diversas situações que podem ocorrer durante sua execução, como, por exemplo, necessidade de liberação de recursos no aparelho, interrupção devido a tarefa de maior prioridade, resposta a chamada de outras aplicações etc.

O ciclo de vida completo do sistema operacional bem como a explicação detalhada do papel de cada método são apresentados na próxima seção.

2.2.2.1 Activity

Como dito, o ciclo de vida do *Android* está diretamente ligado a uma classe chamada *activity*, que possui diversos métodos para lidar com diferentes estados da aplicação em momentos variados. A Figura 4 demonstra o ciclo de vida de uma *activity*.

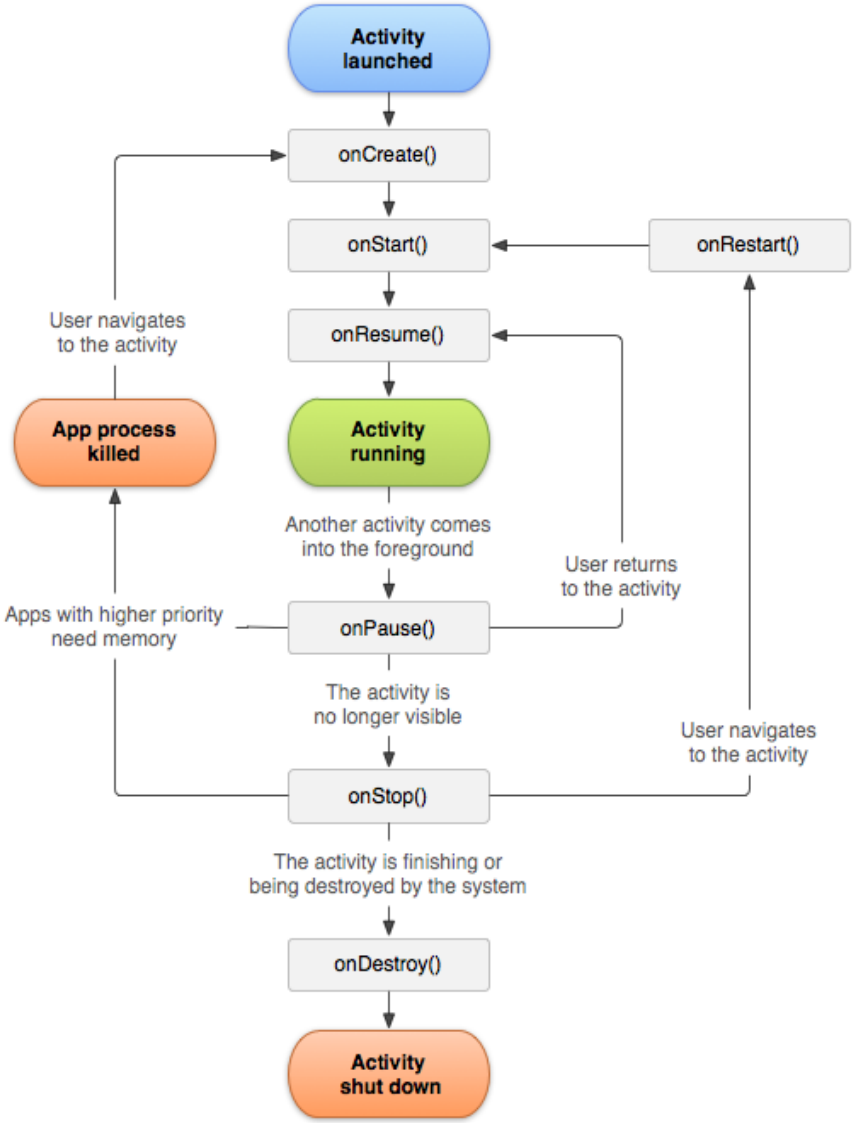

Figura 4 – Ciclo de vida de uma *activity*.
Fonte: http://developer.android.com/guide/topics/fundamentals/activities.html

Após obervar a imagem pode-se ver a descrição de cada um dos métodos atuantes durante sua execução[13].

- onCreate(): é um método obrigatório que sempre será chamado uma única vez durante o início do ciclo de vida da *activity*. É nesse método, que a *view*

[13] GOOGLE. **Activities.** Disponível em: <http://developer.android.com/guide/components/activities.html>.

deve ser criada. Também é o local onde o método *setContentView(view)* deve ser chamado para exibir a *view* na tela. Todas as configurações estáticas da aplicação devem ser executadas dentro do escopo desse método.

- onStart(): chamado no momento anterior a *activity* se tornar visível ao usuário. Pode ser chamada depois dos métodos *onCreate()* ou *onRestart()*, porém, sempre será chamada antes do método *onResume().*
- *onRestart():* quando uma uma *activity* é parada temporariamente e está sendo iniciada outra vez, ou seja, reiniciada. Esse é o método responsável por realizar uma chamada ao método *onStart().* Toda chamada ao método *onRestart()* será precedida pela execução do método *onStart()* pois é ele que conhece como iniciar a *Activity*.
- onResume(): no momento em que a *activity* está no topo da *activity stack* é quando o método *onResume()* está sendo executado. Podemos dizer que a chamada ao método *onResume()* representa o estado "em execução" da *activity*, pois ela está executando como *activity* principal e está pronta para interagir com o usuário. Sempre precedido pela chamada ao método *onStart().*
- *onPause():* caso algum evento ocorra, como por exemplo: o celular entrar em modo de espera para economizar energia ou algum serviço começar a ser processado, a *activity* que está no topo da pilha sendo executada pode ser temporariamente interrompida. Esse método é usado comumente para salvar o estado da aplicação, para que depois o usuário possa continuar de onde parou. Toda e qualquer operação feita nesse método deve ser extremamente rápida, visto que o próximo processo não será executado até que esse método retorne. Pode ser seguido tanto pelo método *onResume(),* caso a *activity* retorne para o topo da *activity stack* e seja exibida ao usuário, quanto pelo método *onDestroy()*, caso a *activity* esteja em vias de ser finalizada.
- *onStop():* quando a *activity* está sendo encerrada e não está mais visível ao usuário, o método *onStop()* é chamado. Pode ser chamado por dois motivos principais: a *activity* está prestes a ser destruída ou outra *activity* (tanto uma nova quanto uma já existente) irá ser retomada. Caso a *activity* vá ser destruída, o método *onDestroy()* será chamado, caso contrário, a *activity*

voltará ao topo da *activity stack* para interagir com o usuário. Neste caso, o método a ser chamado é *onRestart().*

- *onDestroy():* por fim, o método *onDestroy()* é chamado antes da *activity* ser destruída. Esse método pode ser chamado, pois a *activity* está terminando, ou seja, algum evento gerou a chamada ao método *finish()* ou o sistema está destruindo temporariamente a instância dessa *activity* para economizar espaço e diminuir o processamento. Após o método *onDestroy()* ter sido executado, a *activity* é removida completamente da *activity stack* e seu processo no sistema operacional é completamente encerrado.

2.3 WEBSERVICES

Um *webservice* é uma funcionalidade de uma aplicação, desenvolvida para suportar interação máquina-máquina, que pode ser acessada através da rede (intranet ou Internet), utilizando uma arquitetura formalizada e protocolos de comunicação bem definidos como, por exemplo, SOAP (ZARELLI, 2012). Essa aplicação conta com uma interface em um formato bem definido que pode ser processada por máquinas, esse formato é conhecido como WSDL (ZARELLI, 2012).

Fazendo uso dessa interface outros sistemas podem interagir com o *webservice* de uma maneira pré-definida, descrita em seu WSDL por meio de mensagens SOAP, que por sua vez utilizam XML serializado, normalmente transmitido via HTTP[14]. Visando à interoperabilidade do ponto de vista do provedor, não importa em qual linguagem de programação o cliente foi desenvolvido, nem em que sistema operacional ele está sendo executado. Também não são relevantes para o cliente as características técnicas do serviço, no que se diz respeito à linguagem de programação e sistema operacional hospedeiro. Isso se dá pelo

[14] WORLD WIDE WEB CONSORTIUM. **Web services architecture.** Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

motivo de que a interação cliente – provedor é realizada através de protocolos comuns[15].

2.3.1 Elementos

A seguir são apresentados os conceitos dos elementos que compõe um *WebService* para melhor entendê-lo[16]:

Agentes e Serviços: um serviço é um conceito abstrato que deve ser implementado por um agente concreto, ou seja, um serviço não existe por si só. O agente é uma aplicação que possui um ou mais módulos dedicados a fornecer o serviço acordado no WSDL. Normalmente, o agente é um módulo de uma aplicação que envia e recebe mensagens, enquanto o serviço é representado pelo conjunto de funcionalidades fornecidas. Um serviço de busca de cálculo de frete, por exemplo, pode ser implementado por dois agentes completamente diferentes e ainda assim retornar exatamente a mesma informação. Essa é a idéia dos arquivos de definição WSDL[17].

Solicitantes e Provedores: é comum uma pessoa ou organização querer prover um comportamento da sua aplicação para que possa ser utilizada por terceiros. Nesse momento, a pessoa ou organização que prove a implementação do serviço, está se caracterizando como o provedor de serviço (*provider entity)* ou ainda *service provider.* Por outro lado, os clientes que desejam trocar mensagens com o serviço provido são chamados de *requester agent* ou *service requester.*

Descritor de Serviço: mecanismo pelo qual é definida a troca de mensagens entre um *provider entity* e *requester entity*. O WSD é uma especificação processável por máquinas, escrita em WSDL[17]. Ele define os formatos de

---

[15] WORLD WIDE WEB CONSORTIUM. **Simple Object Access Protocol (SOAP 1.1).** Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

[16] WORLD WIDE WEB CONSORTIUM. **Web services architecture.** Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

[17] WORLD WIDE WEB CONSORTIUM. **WSDL and UUID.** Disponível em: <http://www.w3schools.com/wsdl/wsdl_uddi.asp>

mensagens, tipos de dados, protocolos de transporte e forma da mensagem ser trafegada entre o solicitante e o provedor do serviço.

Semântica: pode ser entendida como as expectativas relacionadas ao comportamento do serviço, em particular no que diz respeito às respostas relacionadas às mensagens de requisição enviadas. Com base na semântica pode-se saber qual comportamento esperar em relação a determinada ação, pois, enquanto o WSD governa os mecanismos pelos quais a interação entre solicitantes e provedores vão interagir, a semântica representa o contrato governando o significado e o motivo de determinada interação existir.

2.3.2 Processo de descoberta

Em um primeiro momento, normalmente o solicitante do serviço não sabe exatamente qual provedor pode fornecer o serviço desejado. A partir desse momento, o solicitante, precisa “descobrir” um provedor que forneça os serviços esperados. O processo de “descobrir um serviço” é o ato de encontrar uma descrição processável por máquina, que anteriormente poderia ser desconhecida, mas que corresponde a determinados critérios funcionais. Os principais passos na descoberta de um serviço são os que seguem [18]:

1. O solicitante e o provedor tornam-se conhecidos um ao outro. Isto quer dizer que, de alguma maneira, o endereço onde está disponível o serviço, juntamente com seu WSD e FD foi encontrado, e agora é de conhecimento do cliente. O FD é um arquivo processável por máquina que contém a descrição da funcionalidade (ou sua semântica parcial) do serviço que a entidade provedora está oferecendo. Esse descritor pode conter apenas poucas palavras de metadados ou uma URI. Porém, pode ser mais complexo seguindo o modelo *TModel*.

---

[18] WORLD WIDE WEB CONSORTIUM. **Web services architecture.** Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

2. Ambas as entidades (solicitante / provedor) concordam em relação à semântica utilizada. Isso quer dizer que o retorno em relação às mensagens trocadas se tornou conhecido.
3. Ambos os agentes tornaram-se cientes da descrição do serviço e semântica utilizados, ou seja, os agentes passam a ser implementados de acordo com as descrições encontradas no WSDL.
4. Os agentes trocam mensagens, utilizando protocolo SOAP.

A Figura 5 demonstra o ciclo completo de descoberta de serviço abordando em detalhes os passos descritos anteriormente:

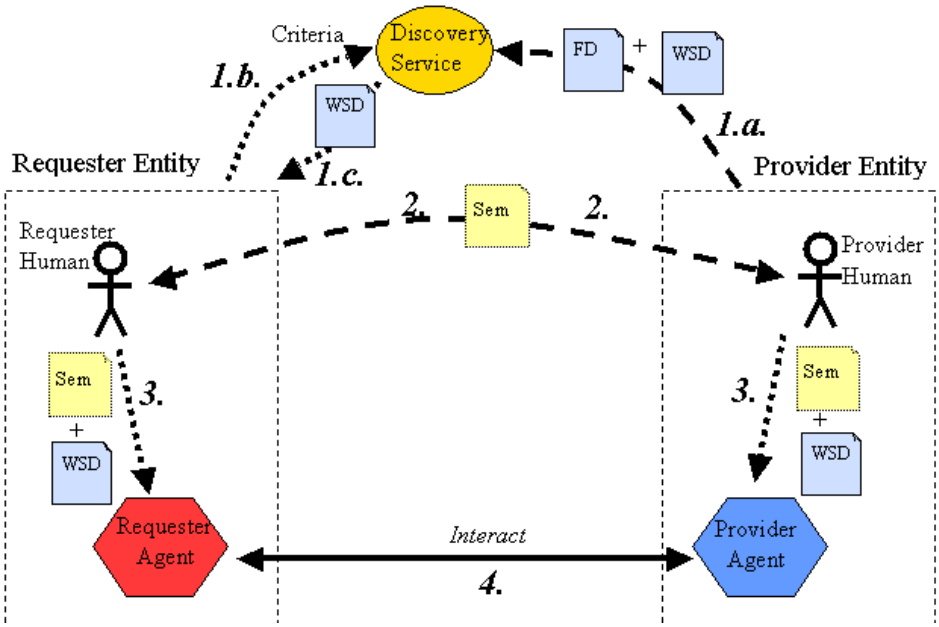

Figura 5 – Ciclo completo de descoberta de serviço.
Fonte: www.w3.org/TR/ws-arch/

2.4 SOAP

SOAP significa, em tradução livre, Protocolo Simples de Acesso a Objetos. É destinado à troca de informações em um ambiente distribuído e descentralizado

através do protocolo HTTP[19]. Seu protocolo é baseado em XML e consiste basicamente em quatro elementos essenciais: *Envelope*, *Header*, *Body* e *Fault* (ZARELLI, 2012).

A Figura 6 exemplifica a divisão de uma mensagem SOAP (ZARELLI, 2012):

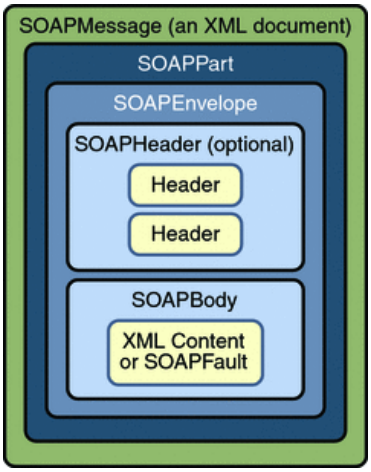

Figura 6 – Divisão de uma mensagem SOAP.
Fonte: http://zarelli.wordpress.com/2012/03/22/como-funciona-o-soap-protocolo-simples-de-acesso-a-objetos/

Envelope: é o elemento raiz da mensagem SOAP. Ele é responsável por, de fato, definir o documento XML como sendo uma mensagem SOAP. Nele se podem encontrar as declarações de *namespaces* e atributos adicionais que definem a representação dos dados.

Header*:* elemento opcional que, caso presente, deve ser o primeiro a aparecer, logo após a definição do elemento *Envelope*. Carrega informações relativas a como o destinatário deverá processar a mensagem.

Body*:* elemento obrigatório para toda e qualquer mensagem SOAP, pois é ele que define o conteúdo da mensagem e contém as *tags* que a aplicação de destino espera para processar seus valores. Opcionalmente pode conter o elemento

[19] WORLD WIDE WEB CONSORTIUM. **Simple Object Access Protocol (SOAP 1.1).** Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

fault, que transporta registros de erros. Os elementos internos de *body* podem conter *namespace*.

Fault: é o elemento de falha, que possui os erros e informações de *status* de uma mensagem SOAP. É opcional e quando estiver presente deve aparecer como um elemento filho de *Body*. Pode conter os subelementos apresentados no Quadro 1:

SubElemento	Descrição
faultcode	Código de identificação de erro
faultstring	Explicação legível da falha
faultactor	Informações sobre o que pode ter provocado a falha
detail	Informações específicas sobre o erro

Quadro 1 – Sub elementos de *Fault*
Fonte : http://zarelli.wordpress.com/2012/03/22/como-funciona-o-soap-protocolo-simples-de-acesso-a-objetos/

2.5 APACHE CXF

O CXF é um *framework* de código aberto desenvolvido pela Apache a partir da combinação de outros dois projetos, também de código aberto, *Celtix* e *XFire.*

O interesse em combinar a funcionalidade desses dois *frameworks,* foi movido pela necessidade dos engenheiros da Apache sempre os utilizarem em conjunto. O Celtix é um aglomerado de APIs, que fornece recursos de segurança e mensageria e tem por objetivo simplificar e facilitar a construção, integração e o reuso de componentes por meio de uma arquitetura orientada a serviços. O *XFire* por sua vez prove suporte a todos os padrões relacionados a *webservices* como por exemplo SOAP, WSDL, Segurança, além de ter perfeita integração com o Spring e com os protocolos HTTP, JMS e outros.

Como resultado dessa integração, o Apache CXF prove todas as funcionalidades que seus *frameworks* de origem forneciam e algumas melhorias como, por exemplo, a possibilidade de utilizar XML em conjunto com anotações para configurar os serviços expostos, bem como validadores de mensagem.

2.6 MAVEN

O Maven teve sua origem durante o desenvolvimento do projeto *Jakarta Alexandria (Sistema para versionamento de códigos fonte e documentos)* sendo apenas um módulo do sistema. Durante seu desenvolvimento foi requisitado que os componentes pertencentes ao Maven fossem retirados de dentro do projeto para se tornar um módulo externo ao *Alexandria*.

Após, ter se consolidado como um projeto independente, e tendo se mostrado maduro o suficiente para ser utilizado, seu primeiro teste funcional foi feito pelos desenvolvedores do projeto *Apache Turbine* (*framework,* baseado na especificação *servlets,* criado para facilitar e tornar mais rápido o desenvolvimento de aplicações web) que possuía três principais camadas: Persistência, Serviço e Web, essas camadas por sua vez eram mantidas em módulos separados, sempre compilados individualmente, o que tornava muito onerosa a manutenção das versões compatíveis entre os módulos do projeto.

Dentro do contexto apresentado, o Maven foi aperfeiçoado, focando em duas principais necessidades:

1. Um modelo de projeto, para que fosse fácil e possível identificar em apenas um lugar todos os artefatos e estruturas pertencentes ao projeto.
2. Uma estrutura de diretórios padrões, a fim de organizar o projeto e concentrar as bibliotecas, códigos e documentos em um só lugar.

Tendo atingido os objetivos acima citados, percebeu-se que, seria necessária a utilização de um repositório de bibliotecas para evitar a desorganização, redundância ou incompatibilidade entre bibliotecas dentro de grandes projetos, a partir desse ponto criou-se o conceito de dependência [20].

Hoje é possível definir de forma clara e objetiva o Maven como uma ferramenta de gerência e construção (*build)* de projetos.

---

[20] APACHE MAVEN PROJECT. **History of Maven by Jason van Zyl.** Disponível em: <http://maven.apache.org/background/history-of-maven.html>.

Engloba um arquivo, que define o modelo de projeto (comumente chamado de “Project Object Model”, POM), um conjunto de padrões, o ciclo de vida do projeto, um sistema de gerenciamento de dependências e a lógica necessária para executar plug-ins em determinada etapa do ciclo de vida do projeto. A arquitetura do Maven pode ser vista a seguir, na figura 7:

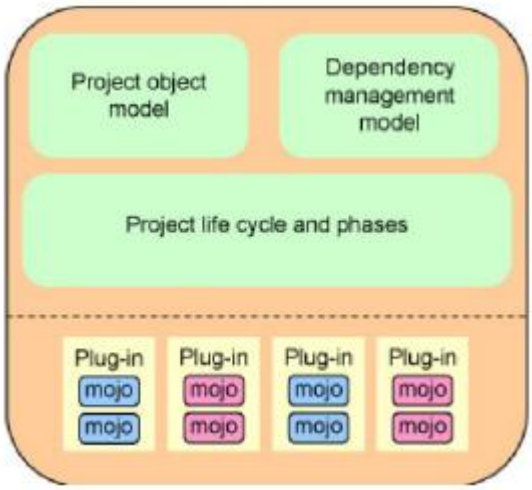

Figura 7 - Arquitetura do projeto Apache Maven
Fonte : (FILHO. W.S, 2008)

A figura 8 exemplifica as definições de projeto contidas em um arquivo POM:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>my-project</artifactId>
    <version>1.0</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.1.1.RELEASE</version>
        </dependency>
    </dependencies>
</project>
```

Figura 8 - Exemplo básico de arquivo POM
Fonte : http://maven.apache.org/pom.html#What_is_the_POM

3 DESENVOLVIMENTO

3.1 ESPECIFICAÇÃO DE REQUISITOS

3.1.1 Requisitos funcionais

- Permitir a filtragem de ofertas por nível nacional, estadual e regional.
- Visualizar detalhe da oferta selecionada.
- Redirecionar usuário ao *site* do ofertante.
- Permitir filtragem de ofertas de acordo com as preferências escolhidas pelo usuário.
- Marcar oferta como favorita.
- Visualizar todas as ofertas da cidade escolhida utilizando o Google Mapas como API.
- Permitir configuração de estado, município, porcentagem de desconto, valor da oferta, categorias de interesse.

3.1.2 Requisitos não funcionais

- A quantidade de informações enviadas pelo servidor à aplicação não deve ser demasiadamente grande, pois, uma vez que a comunicação entre cliente (celular, *tablet* etc.) e servidor (servidor web) é feita através da Internet, muitos dados trafegando poderão gerar um longo tempo de espera, indesejado pelo usuário.
- A interface deve ser limpa e intuitiva. O fácil acesso aos itens desejados deve ser provido ao usuário.

3.2 RESTRIÇÕES

- Necessidade de acesso a Internet e sistema operacional Android 2.2.

3.3 SOLUÇÃO PROPOSTA

Frente ao problema encontrado, foram propostos objetivos específicos para buscar uma solução adequada, flexível e com grande manutenibilidade. O sistema agregador de ofertas AGREGUEI, possui três módulos bem definidos, responsáveis por etapas específicas no processo de tornar uma oferta disponível ao usuário da aplicação *Android*.

Primeiramente, para tornar a abordagem genérica provê-se aos clientes (*sites* de ofertas) um XSD, que define, quais são as *tags* esperadas dentro do arquivo XML que contém os dados sobre a oferta a ser publicada no agregador de ofertas.

Uma vez que o cliente tenha conhecimento do XSD, será necessário que o XML gerado pelo cliente seja colocado em uma pasta específica de seu servidor *web* para que o módulo responsável pela validação e importação desse arquivo possa encontrar o arquivo disponibilizado. Após realizar sua validação os dados são persistidos na base de dados do sistema AGREGUEI.

Concluída a etapa de importação e persistência da oferta, pensando em tornar o sistema uma fonte de dados flexível a qualquer que seja a implementação do cliente, disponibilizar-se-ão as ofertas através de um *WebService* principal que contém vários métodos responsáveis por oferecer as informações de acordo com o filtro utilizado pela aplicação cliente. Utilizando essa abordagem, toda e qualquer informação trafegada entre cliente/servidor será um XML utilizado para definir uma mensagem SOAP.

Por fim, tendo passado pelas etapas de importação, persistência e validação, a aplicação cliente, nesse caso uma aplicação Android, poderá requisitar através de uma mensagem SOAP as informações relevantes a serem apresentadas ao usuário, baseadas nos filtros escolhidos por ele.

A solução proposta é flexível, pois a forma de importação da oferta pode ser estendida assim como sua plataforma de exibição. Utiliza-se uma conexão FTP para importar o XML que contém a oferta diretamente do servidor do cliente cadastrado. Entretanto, como um módulo de validação é oferecido, a oferta pode ocorrer por meio de uma interface *web*, sendo enviada pelo cliente ao sistema.

Uma vez que as informações são disponibilizadas via *WebService*, é totalmente viável implementar uma interface *web* para exibição das ofertas,

oferecendo assim o sistema em duas plataformas distintas, *Web* e *Mobile*, sem necessidade de alterar a forma de funcionamento do núcleo do sistema AGREGUEI.

3.3.1 Módulos

Pode-se observar abaixo o detalhamento de cada módulo desenvolvido:

Core → Importação, validação e persistência da oferta disponibilizada pelo cliente em seu FTP. Plataforma *desktop*.

WebService → Disponibilização das ofertas cadastradas na base do sistema AGREGUEI por meio de mensagens SOAP. Plataforma *Web*.

Mobile → Sistema cliente da aplicação AGREGUEI, responsável por realizar chamadas ao WebService, interpretar a resposta e disponibilizar o resultado ao usuário. Plataforma *Mobile* (Android 2.2).

A Figura 8 mostra a interação entre os três módulos desenvolvidos.

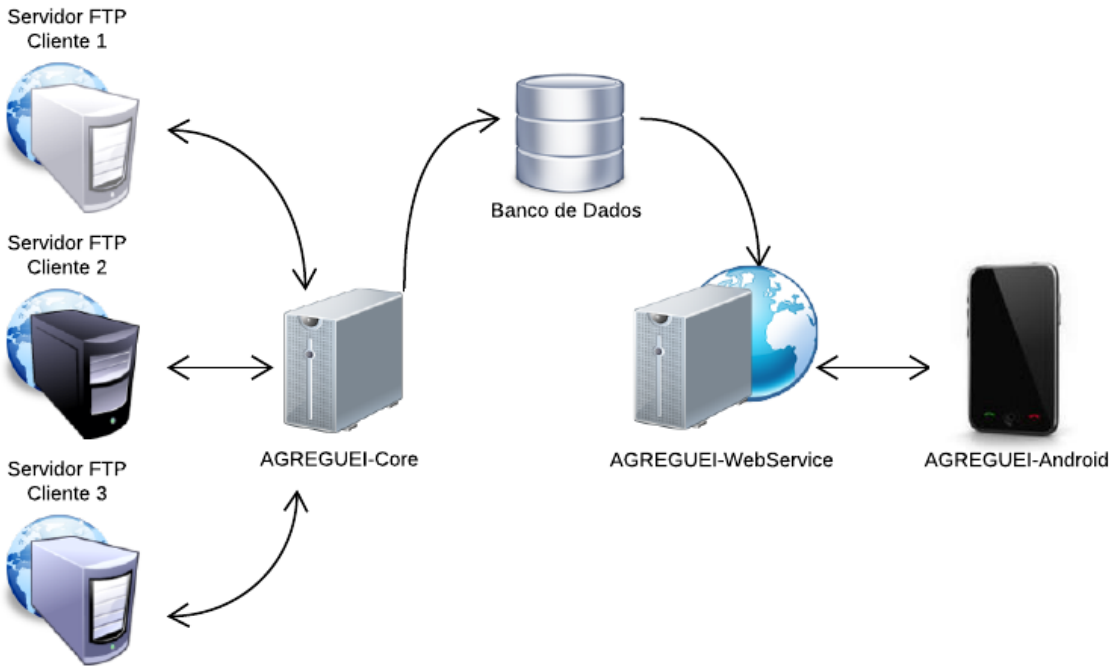

Figura 9 – Interação entre os 3 módulos desenvolvidos
Fonte: O autor.

3.4 MODELAGEM DO SISTEMA

3.4.1 Diagrama de casos de uso

As funcionalidades apresentadas na aplicação serão descritas a seguir através da utilização da linguagem de modelagem UML, que fornece diagramas que ajudam a descrever e projetar funcionalidades e comportamentos de um sistema, principalmente sistemas orientados a objetos. Na Figura 9 observa-se o diagrama de casos de uso do sistema AGREGUEI.

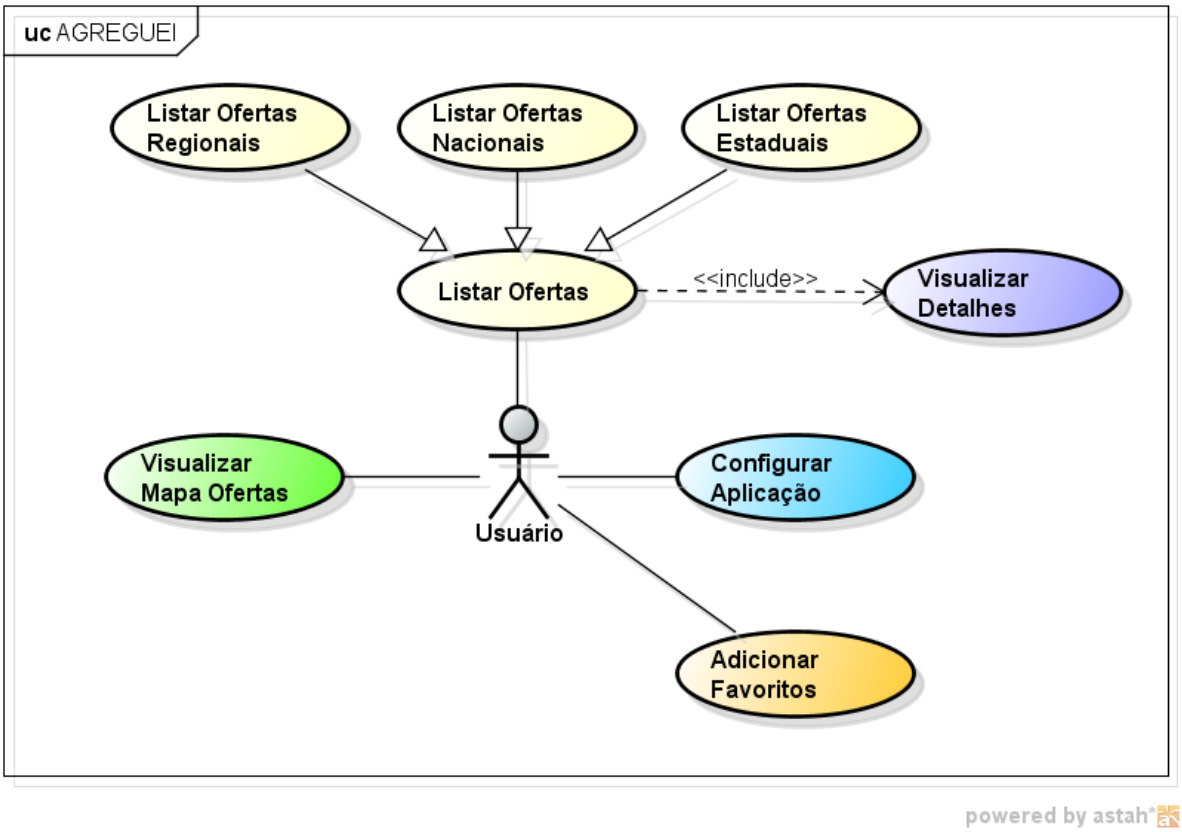

Figura 10 – Diagrama de casos de uso do sistema AGREGUEI.
Fonte: O autor.

3.4.2 Diagrama de classes

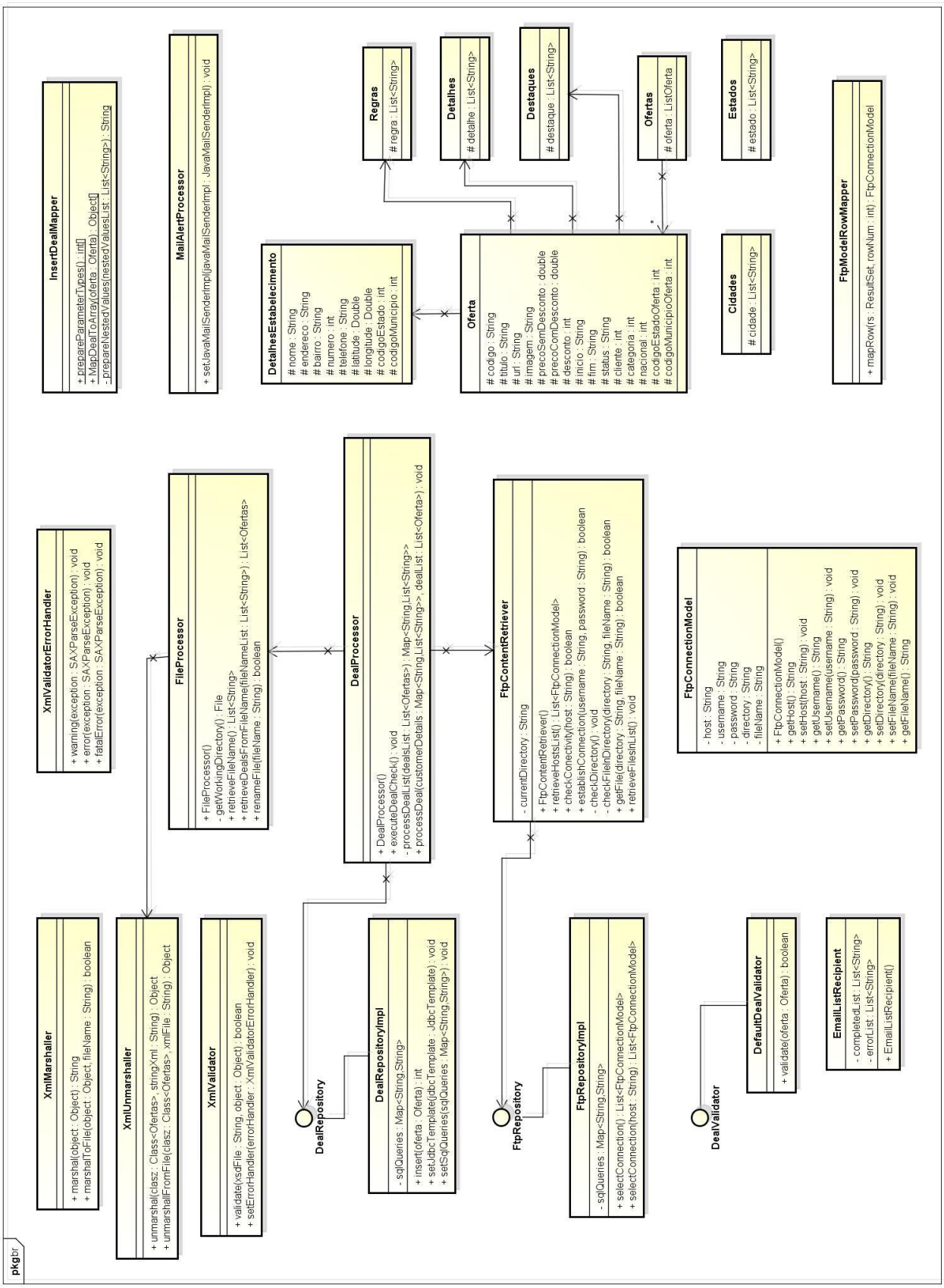

Figura 11 - Diagrama de Classes do *Core*
Fonte: O autor

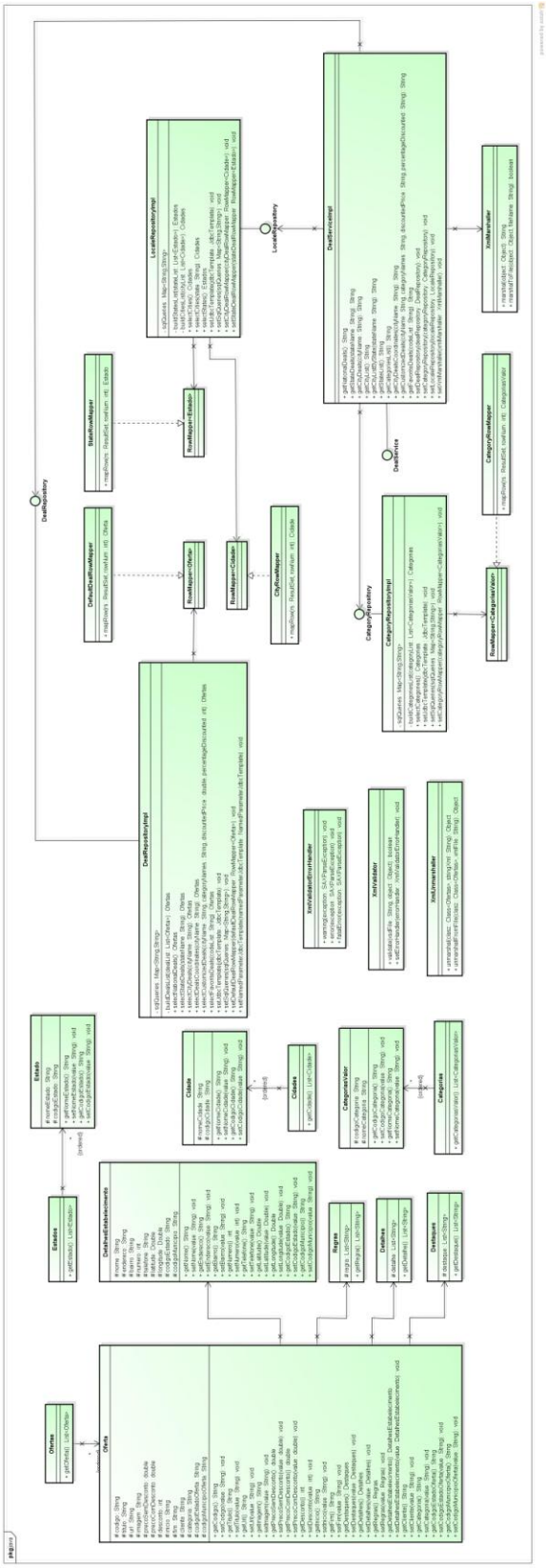

Figura 12 - Diagrama de Classes do *Webservice*
Fonte: O autor

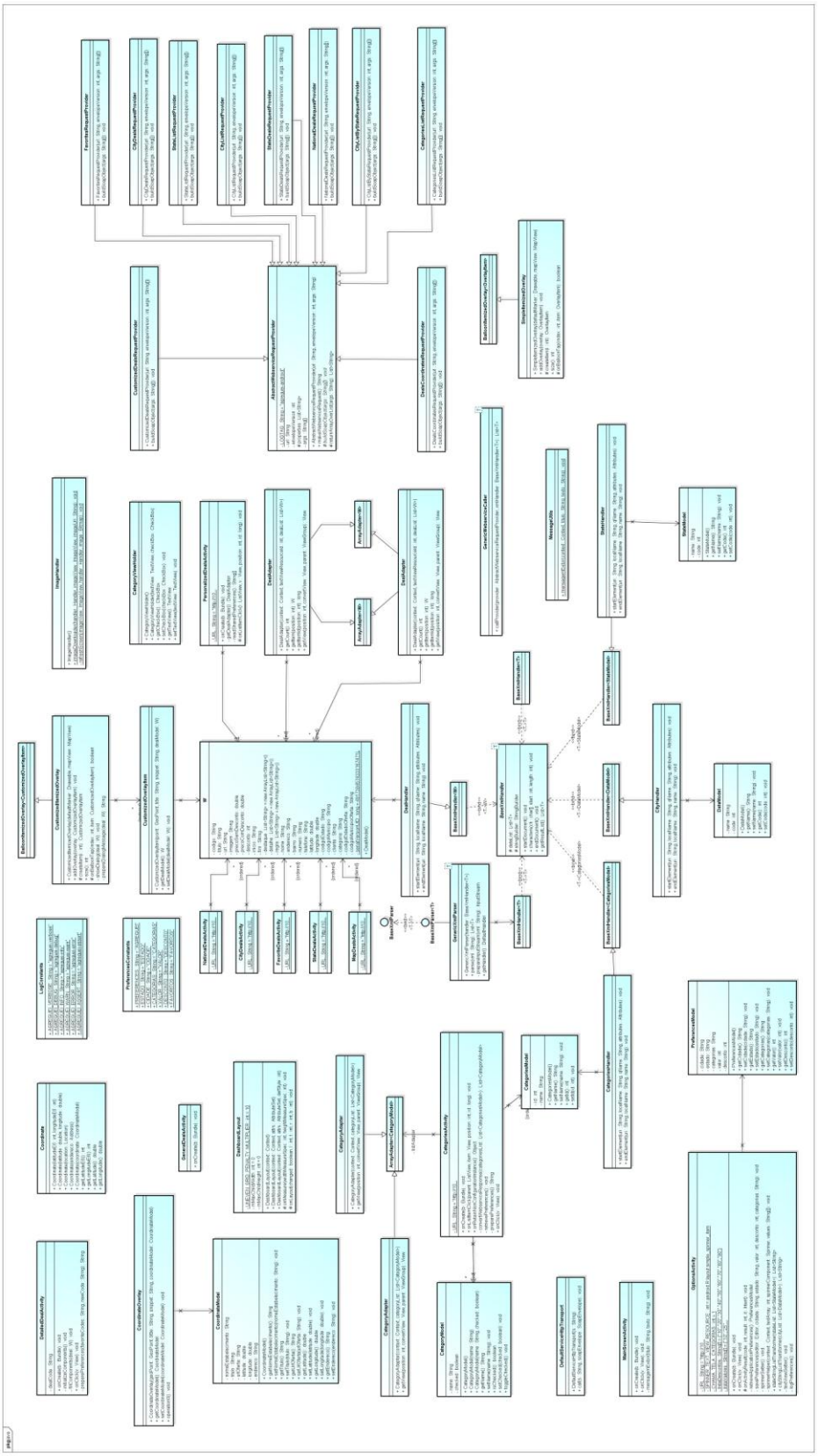

Figura 13 - Diagrama de Classes do *Android*
Fonte: O autor

3.5 IMPLEMENTAÇÃO DA SOLUÇÃO PROPOSTA

Visando o melhor entendimento da proposta dos módulos que compõe o Agreguei, a partir de uma visão técnica expondo de forma segregada e independente, mas, seguindo a ordem lógica de interação entre os módulos, cada um deles terá suas funcionalidades, relevantes, expostas em um subitem exclusivo.

3.5.1 Agreguei – Core

A necessidade de desenvolvimento de um módulo dedicado as principais funcionalidades da aplicação (*Pooling* de conexões FTP, Importação de arquivos, Validação XML, etc.) se deu pela motivação de evitar um alto acoplamento entre os módulos, ou seja, o core não precisa conhecer como os dados coletados serão disponibilizados ao *Android.* Por sua vez o módulo Android assim como o módulo WebService, não sabe como os dados foram coletados.

Essa abordagem torna a exposição do serviço independente da forma de coleta das ofertas, o que se torna interessante, pois uma vez que os dados (arquivos) não sejam mais disponibilizados em um servidor web para serem recuperados via FTP, mas sim através de e-mail, o módulo de serviço não será afetado.

A implementação desse módulo foi completamente norteada pelos princípios de programação orientada a interfaces apoiada pelos recursos de inversão de controle e injeção de dependência providos pelo Spring e suportada pela utilização do framework Junit para testes unitário.

Na figura 14 é possível verificar como foi construído o arquivo POM, responsável por definir o modelo do projeto. Um diferencial em suas configurações foi a utilização do *plug-in* da biblioteca JAXB para realizar a geração de todas as classes de modelo baseado nos arquivos XSD criados para esse fim.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.edu.utfpr.agreguei</groupId>
    <artifactId>agreguei-core</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <name>agreguei-core</name>
    <description>
        Projeto que contem os principais recursos do projeto agreguei.
    </description>
    <build>
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <filtering>false</filtering>
            </resource>
        </resources>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.0.2</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.jvnet.jaxb2.maven2</groupId>
                <artifactId>maven-jaxb2-plugin</artifactId>
                <version>0.8.0</version>
                <executions>
                    <execution>
                        <id>generate-java-classes-agreguei-core</id>
                        <goals>
                            <goal>generate</goal>
                        </goals>
                        <phase>generate-sources</phase>
                        <configuration>
                            <schemaIncludes>
                                <include>agregueiSchema.xsd</include>
                            </schemaIncludes>
                            <generatePackage>
                                br.edu.utfpr.agreguei.xml.model
                            </generatePackage>
                            <schemaDirectory>src/main/schema</schemaDirectory>
                        </configuration>
                    </execution>
                </executions>
                <configuration>
                    <generateDirectory>src/main/java</generateDirectory>
                    <forceRegenerate>true</forceRegenerate>
                    <cleanPackageDirectory>false</cleanPackageDirectory>
                    <verbose>true</verbose>
                    <episode>false</episode>
                </configuration>
                <dependencies>
                    <dependency>
                        <groupId>com.sun.xml.bind</groupId>
                        <artifactId>jaxb-xjc</artifactId>
                        <version>2.2.4-1</version>
                    </dependency>
                </dependencies>
            </plugin>
        </plugins>
    </build>
```

Figura 14 - Parte do arquivo POM do modelo do módulo Agreguei-Core
Fonte: O autor

Visando tornar a camada de persistência mais leve e funcional foram definidas configurações de repositórios para domínios relevantes da aplicação, dessa forma, é possível centralizar o acesso a dados e por meio da injeção de dependência utilizar esses recursos nas classes de serviço. Na figura 15 vemos a definição da forma de conexão a fonte de dados utilizada pela aplicação, a figura 16 contem todas as *queries* utilizadas na implementação do repositório, como pode ser visto na figura 17, que conhece os dados referentes a conexão FTP fornecida por cada cliente da aplicação. Por fim a figura 18 apresenta a utilização do repositório pela classe de serviço via injeção de dependência.

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.postgresql.Driver"/>
    <property name="url" value="jdbc:postgresql://localhost:5432/agregueiCustomers"/>
    <property name="username" value="postgres"/>
    <property name="password" value="root"/>
</bean>

<bean id="defaultJdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

Figura 15 - Configuração de acesso a fonte de dados
Fonte : O autor

```
<bean id="ftpRepository" class="br.edu.utfpr.agreguei.repository.FtpRepositoryImpl">
    <property name="jdbcTemplate" ref="defaultJdbcTemplate" />
    <property name="sqlQueries" ref="ftpRepositoryQueries" />
    <property name="defaultRowMapper" ref="defaultFtpRowMapper" />
</bean>

<bean id="defaultFtpRowMapper" class="br.edu.utfpr.agreguei.ftp.mapper.FtpModelRowMapper" />

<util:map id="ftpRepositoryQueries">
    <description>Lista contendo as queries acessíveis pelo repositório</description>
    <entry key="selectConnectionlookup">
        <value>
            <![CDATA[select host,usuario,senha,diretorio,arquivo from tb_conexaoftp]]>
        </value>
    </entry>
    <entry key="selectSpecificConnectionLookup">
        <value>
            <![CDATA[select host,usuario,senha,diretorio,arquivo from tb_conexaoftp where host = ?]]>
        </value>
    </entry>
</util:map>
```

Figura 16 - Configuração das consultas a serem realizadas na fonte de dados
Fonte : O autor

```
public class FtpRepositoryImpl implements FtpRepository {

    private JdbcTemplate jdbcTemplate;
    private Map<String,String> sqlQueries;
    private RowMapper<FtpConnectionModel> defaultRowMapper;

    public List<FtpConnectionModel> selectConnection() {
        String sqlString = sqlQueries.get("selectConnectionlookup").toString();
        return jdbcTemplate.query(sqlString, defaultRowMapper);
    }

    public List<FtpConnectionModel> selectConnection(String host) {
        String sqlString = sqlQueries.get("selectSpecificConnectionlookup").toString();
        return jdbcTemplate.query(sqlString, defaultRowMapper);
    }

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public void setSqlQueries(Map<String, String> sqlQueries) {
        this.sqlQueries = sqlQueries;
    }

    public void setDefaultRowMapper(RowMapper<FtpConnectionModel> defaultRowMapper) {
        this.defaultRowMapper = defaultRowMapper;
    }

}
```

Figura 17 - Implementação do repositório que conhece dados sobre os clientes e seus FTPs
Fonte : O autor

```
public class FtpContentRetriever {

    private FtpRepository ftpRepository;

    public FtpContentRetriever() {
        super();
    }

    public List<FtpConnectionModel> retrieveHostsList() {
        return ftpRepository.selectConnection();
    }
```

Figura 18 - Exemplo de utilização de repositório pela classe de serviço via injeção de dependência
Fonte : O autor

A funcionalidade de *polling* das conexões FTP foi implementada utilizando uma solução de *cron* fornecida pelo Spring, como pode ser visto a seguir, na figura 19.

```
<task:scheduled-tasks scheduler="taskScheduler">
    <task:scheduled ref="dealProcessor" method="executeDealCheck" fixed-delay="60000" />
</task:scheduled-tasks>

<task:scheduler id="taskScheduler" pool-size="1" />
```

Figura 19 - Configuração do scheduler responsável pelo pooling de conexões FTP
Fonte : O autor

3.5.2 Agreguei – Webservice

Buscando reduzir ao máximo o acoplamento entre o *back end.* da aplicação e seu *front end.* surgiu a necessidade de desenvolver o módulo provedor de serviço chamado Agreguei *WebService*, através deste é possível tornar transparente à aplicação *Android* a forma como os dados são obtidos e disponibilizados para consumo.

Vale ressaltar nessa implementação a utilização da tecnologia Apache CXF em conjunto com o *Spring* para implementar o *WebService* de maneira muito simples, apenas é necessário a configuração do *servlet* do Apache CXF responsável por interceptar as requisições enviadas ao servidor web e utilizar anotações nas interfaces e classes responsáveis pelas definições dos serviços, como pode ser visto na figura 20 e 21. Os atributos da classe “*DealServiceImpl*”, mostrada na figura 22, são utilizados através da injeção realizada pelo *Spring*.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="services" version="2.5">

    <context-param>
        <param-name>webAppRootKey</param-name>
        <param-value>cxf.rest.example.root</param-value>
    </context-param>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Figura 20 - Configuração do arquivo web.xml para utilização do Apache CXF e Spring
Fonte : O autor

```
@WebService(name="AgregueiService", targetNamespace="http://agreguei_webservice/")
public interface DealService {

    @WebMethod(operationName="getNationalDeals")
    public String getNationalDeals();

    @WebMethod(operationName="getStateDeals")
    public String getStateDeals(@WebParam(name="stateName") String stateName);

    @WebMethod(operationName="getCityDeals")
    public String getCityDeals(@WebParam(name="cityName") String cityName);

    @WebMethod(operationName="getCityList")
    public String getCityList();
```

Figura 21 - Exposição das interfaces de serviço
Fonte : O autor

```
@WebService(
        portName="AgregueiWebservicePort",
        serviceName = "AgregueiService",
        targetNamespace="http://agreguei_webservice/",
        endpointInterface = "br.edu.utfpr.agreguei.service.api.DealService" )
public class DealServiceImpl implements DealService {

    private DealRepository dealRepository;
    private CategoryRepository categoryRepository;
    private LocaleRepository localeRepository;
    private XmlMarshaller xmlMarshaller;

    public String getNationalDeals() {
        return xmlMarshaller.marshal(dealRepository.selectNationalDeals());
    }

    public String getStateDeals(String stateName) {
        return xmlMarshaller.marshal(dealRepository.selectStateDeals(stateName));
    }

    public String getCityDeals(String cityName) {
        return xmlMarshaller.marshal(dealRepository.selectCityDeals(cityName));
    }

    public String getCityList() {
        return xmlMarshaller.marshal(localeRepository.selectCities());
    }
```

Figura 22 - Exposição da implementação dos serviços definidos com anotações do apache CXF
Fonte : O autor

Na figura 23 constam as configurações do modelo de projeto referentes ao módulo Agreguei Webservice, vale ressaltar a utilização do *plug-in* do Apache Tomcat que permite realizar o deploy do projeto diretamente no servidor.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.edu.utfpr.agreguei</groupId>
    <artifactId>agreguei-webservice</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>agreguei-webservice</name>
    <description>Webservice project that will supply mobile module.</description>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                    <encoding>ISO-8859-1</encoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>tomcat-maven-plugin</artifactId>
                <configuration>
                    <url>http://localhost:8080/manager</url>
                    <server>Tomcat-6.0</server>
                    <path>/agreguei-webservice</path>
                    <username>Marcelo</username>
                    <password>afanaci</password>
                    <update>true</update>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <inherited>true</inherited>
            </plugin>
        </plugins>
    </build>
</project>
```

Figura 23 - Definição do arquivo de modelo do módulo Agreguei - Webservice
Fonte : O autor

3.5.3 Agreguei – Android

O módulo Android é responsável por mostrar ao usuário da aplicação os dados coletados pelo core e disponibilizado pelo WebService. As principais funcionalidades implementadas para tornar possível a comunicação entre cliente e servidor foram a conversão das mensagens enviadas pelo Webservice ao Android e a utilização da biblioteca *KSoap* para comunicação.

3.6 APRESENTAÇÃO DO SISTEMA DESENVOLVIDO

Na tela inicial são apresentadas as principais opções da aplicação. Uma vez que o usuário esteja visualizando, é possível escolher dentre seis opções distintas, como mostrado na Figura 24:

Figura 24 – Tela principal do aplicativo
Fonte: O autor.

Os seis ícones na tela representam as seguintes funcionalidades da aplicação:

1. Listar ofertas gerais (sacola de compras).
2. Listar ofertas personalizadas (alvo).
3. Visualizar localização de ofertas (mapa).
4. Visualizar ofertas favoritas (pasta com estrela).
5. Configurações (engrenagem).
6. Informações sobre aplicação/desenvolvedor (letra “i”).

3.6.1 Listar ofertas gerais

Essa tela é responsável pela listagem das ofertas disponíveis sem considerar limite de preço ou porcentagem de desconto, porém, divididas em categorias. As três abas presentes no topo da janela dividem as ofertas em nacionais, que não demandam configuração adicional alguma por parte do usuário da aplicação; estaduais e regionais, que dependem diretamente das opções de localidade configuradas pelo usuário na tela de configurações.

A listagem de ofertas oferece aos usuários, em um primeiro momento, a possibilidade de visualizar informações básicas como o título da oferta, valor sem desconto, valor com desconto, porcentagem de desconto e o *site* que está disponibilizando a oferta. Essas informações podem ser visualizadas independentemente da categoria escolhida para listagem, como pode ser visto na Figura 25:

Figura 25 – Listagem geral de ofertas
Fonte: O autor.

Além das informações resumidas visualizadas na imagem anterior, uma vez que o usuário clique na oferta desejada, uma tela contendo as informações detalhadas sobre a oferta é apresentada. Essa tela (figura 25) apresenta os seguintes dados: categoria, título, valor sem desconto, valor com desconto, porcentagem de desconto, nome do estabelecimento, endereço, *site* ofertante, data e hora de expiração da oferta.

Também é possível adicionar a oferta que está sendo visualizada aos favoritos, gerando assim um acesso rápido e fácil às ofertas escolhidas. Caso o usuário deseje comprar a oferta, basta clicar no botão “Comprar” que a aplicação redireciona o usuário ao *site* da oferta. O *layout* dos detalhes pode ser visto na Figura 26:

Figura 26 – Detalhes da oferta escolhida
Fonte: O autor.

3.6.2 Listar ofertas personalizadas

Os dados apresentados nessa tela são obtidos considerando as configurações, porcentagem mínima de desconto e valor máximo da oferta, conforme as escolhas do usuário. As categorias de interesse escolhidas pelo usuário também são consideradas no momento de buscar pelas ofertas que mais se encaixam no perfil do usuário. As informações presentes nessa tela são as mesmas existentes na tela de listagem de ofertas gerais, como pode ser visualizado na Figura 27, a seguir:

Figura 27 – Listagem de ofertas personalizadas
Fonte: O autor.

3.6.3 Listar localização ofertas

Com base nas configurações de localidade definidas pelo usuário é possível visualizar todas as ofertas disponíveis na cidade escolhida. O usuário pode escolher entre duas visões do mapa: mapa e satélite, além do nível de zoom em que o mapa será visualizado. Quando uma oferta é clicada, seu título e endereço são

apresentados em um balão logo acima da imagem que representa a localização. Uma vez que o balão tenha sido clicado, todas as informações referentes à oferta escolhida serão apresentadas ao usuário, dando a ele a opção de comprar ou não a oferta. Caso deseje comprá-la, ele será redirecionado automaticamente ao *site* da oferta, caso contrário, voltará para o mapa contendo as ofertas. Essas funcionalidades podem ser visualizadas nas figuras 28, 29, 30 e 31 que seguem:

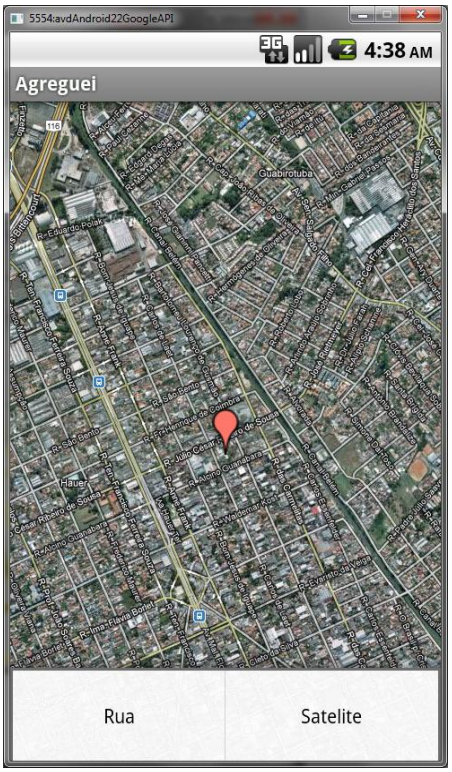

Figura 28 – Opções de exibição do mapa
Fonte: O autor

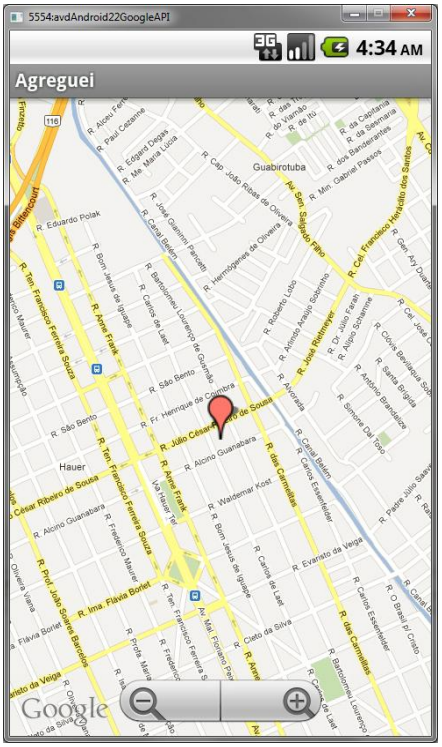

Figura 29 – Opção de zoom
Fonte: O autor.

Figura 30 – Informações gerais sobre a oferta
Fonte: O autor.

Figura 31 – Oferta detalhada
Fonte: O autor.

3.6.4 Visualizar ofertas favoritas

Quando apresentada a tela de detalhes da oferta, existe a opção de adicionar a oferta aos seus favoritos. Uma vez adicionada aos favoritos, ela ficará disponível até sua data de expiração, após essa data será automaticamente removida, pois não há sentido em manter uma oferta já vencida. Na Figura 32 pode-se visualizar a tela de listagem de ofertas favoritas:

Figura 32 – Ofertas adicionadas aos favoritos
Fonte: O autor.

3.6.5 Configurações

O usuário poderá configurar suas categorias preferidas, o estado e cidade pelos quais as ofertas serão filtradas, bem como a porcentagem mínima de desconto e o valor máximo da oferta. Esses filtros visam a diminuir a quantidade de ofertas não desejadas recebidas pelo usuário, principalmente na listagem de ofertas personalizadas. As figuras 33, 34, 35, 36, 37 e 38, a seguir, exemplificam as configurações disponíveis:

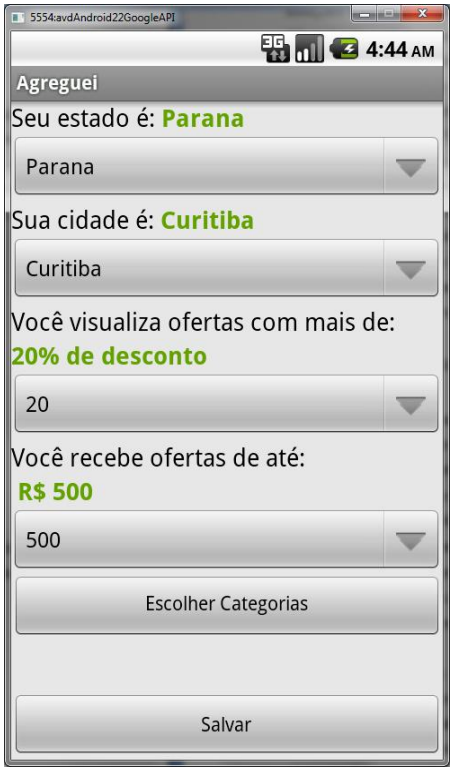

Figura 33 – Tela de configurações
Fonte: O autor.

Figura 34 – Lista de cidades
Fonte: O autor.

Figura 35 – Lista de estados
Fonte: O autor.

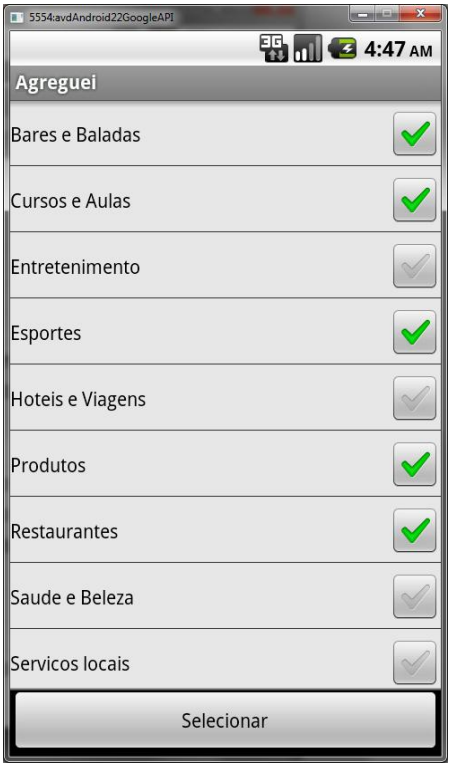

Figura 36 – Lista de categorias
Fonte: O autor.

Figura 37 – Faixa de valores
Fonte: O autor.

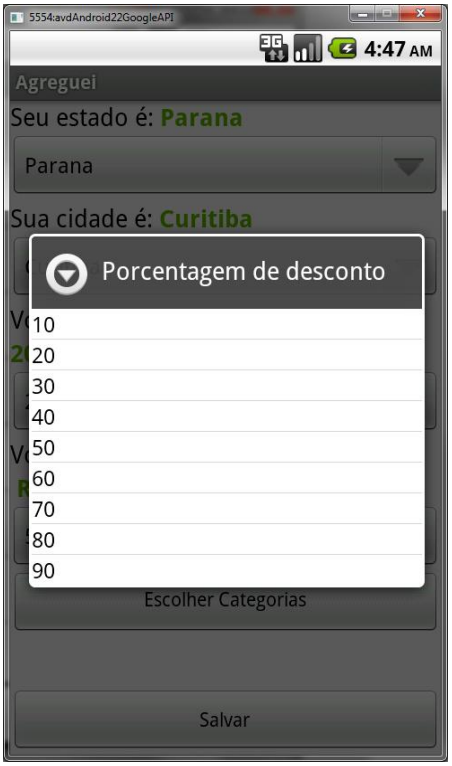

Figura 38 – Faixa de porcentagens
Fonte: O autor.

3.6.6 Informações sobre a aplicação/desenvolvedor

O usuário poderá visualizar um alerta, disparado com as informações sobre a aplicação e seu desenvolvedor assim como o motivo do desenvolvimento do aplicativo, como mostra a Figura 39 que se segue:

Figura 39 – Informações sobre a aplicação
Fonte: O autor.

4 CONSIDERAÇÕES FINAIS

A aplicação apresentada cumpriu os objetivos propostos tais como diminuir o fluxo de informação enviada ao cliente, apresentar diferenciais em relação às soluções encontradas atualmente no mercado e servir como uma integradora de diferentes tecnologias pertencentes a diferentes plataformas.

O modelo da solução baseou-se na integração entre diferentes plataformas como *web*, *desktop* e móvel para criar uma aplicação que atendesse de forma flexível às necessidades impostas pelo modelo de negócio adotado para estudo.

Como consequência direta da necessidade de integração, a decisão por utilizar *Web Services* se fez válida e acertada, pois acelerou o desenvolvimento e expandiu as possibilidades de incorporação de novas funcionalidades sem alteração da implementação já concluída.

4.1 TRABALHOS FUTUROS

Existem algumas funcionalidades interessantes que poderiam ser implementadas, a fim de tornar a aplicação ainda mais singular se comparada a outras disponíveis no mercado.

- Histórico de ofertas visualizadas pelo usuário a fim de poder apresentar aos ofertantes as características das ofertas que mais chamam atenção dos usuários.
- Posicionamento em tempo real do usuário via GPS para que em determinado raio de alcance fossem mostradas as ofertas disponíveis.
- Geração de alerta a cada nova oferta cadastrada que se encaixe nos filtros de suas ofertas personalizadas.
- Criação de uma interface *web* para que o cliente possa realizar o envio e validação de sua oferta por meio de uma interface rica e intuitiva.

Alem dos itens relativos a melhorias diretamente na aplicação, seria interessante realizar um estudo com os maiores *sites* de compra coletiva para mensurar suas expectativas em relação à evolução do mercado de compras

coletivas, procurando identificar se essa evolução demanda uma nova solução em agregador de oferta para plataforma *mobile*.

Seria interessante ouvir os clientes para saber quais pontos de melhoria, diferentes dos já expostos, eles consideram importantes, a fim de agregar mais valor à aplicação, o que se refletiria diretamente no lucro do cliente.

REFERÊNCIAS

BOLSA DE OFERTAS. **Brasil tem quase 2000 sites de compras coletivas.** Disponível em: <http://www.bolsadeofertas.com.br/brasil-tem-1963-sites-voltados-para-compras-coletivas/>. Acesso em: 07 jun 2012.

COMPUTERWORLD. **Brasil deverá ser 4º maior em *e-commerce* até 2015, diz estudo.** Disponível em: <http://computerworld.uol.com.br/negocios/2011/12/28/brasil-devera-ser-4o-maior-em-e-commerce-ate-2015-diz-estudo/>. Acesso em: 07 jun 2012.

DOONG, H., KAUFFMAN R. J., LAI, H., ZHUANG, Y. Empirical design of incentive mechanisms in group-buying auctions. In: KAUFFMAN R. J., TALLON, P. P. (Eds.) Economics, information system, and electronic commerce: empirical research. New York: AMIS, 2009. p. 181-220.

E-COMMERCE GUIDE. **Tendências do E-Commerce em 2012 no Brasil.** Disponível em: <http://www.e-commerceguide.com.br/2011/12/tendencias-do-e-commerce-em-2012-no.html>. Acesso em: 08 jun 2012.

E-Commerce News. **E-Commerce cresce 40% no Brasil em 2010.** Disponível em: <http://ecommercenews.com.br/noticias/pesquisas-noticias/e-commerce-cresce-40-no-brasil-em-2010>. Acesso em: 07 jun 2012.

FILHO, W. S. Introdução ao Apache Maven. Sergipe, p.7, ago. 2008.

FOWLER, M. 2005. **Inversion of control.** Disponível em: <http://martinfowler.com/bliki/InversionOfControl.html>. Acesso em: 10 jun 2012.

FOWLER, M. 2005. **Inversion of control.** Disponível em: <http://martinfowler.com/bliki/InversionOfControl.html>. Acesso em: 10 jun 2012.

GADE, C. Psicologia do consumidor e da propaganda. São Paulo, Editora Pedagógica Universitária, 2004.

GALO, B. 2010. **Este garoto é o novo rei da Internet.** Disponível em: <http://www.istoedinheiro.com.br/noticias/36191_ESTE+GAROTO+E+O+NOVO+REI+DA+INTERNET>. Acesso em: 07 jun 2012.

GOOGLE. **Activities.** Disponível em: <http://developer.android.com/guide/components/activities.html>. Acesso em: 14 jun 2012.

GOOGLE. **Publishing on Google Play.** Disponível em: <http://developer.android.com/guide/publishing/publishing.html>. Acesso em: 14 jun 2012.

GOOGLE. **What is Android ?** Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 14 jun 2012.

IBOPE. **Comércio eletrônico - avaliação 360º.** Disponível em: <http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=Internet&docid=0AADE4C29DC53A458325796D0050A2DD>. Acesso em: 07 jun 2012.

JOHNSON, R.; et al. 2011. **Spring Framework. Reference Documentation.** Disponível em: <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/>. Acesso em: 11 jun 2012.

JOHNSON. R.E.; FOOTE, B. **Designing Reusable Classes.** Journal of Object-Oriented Programming, June/July 1988, Vol 1 - nº2. p. 22-35.

KAUFFMAN, J. R.; WANG, B. 2002. Bid together, buy together: on the efficacy of group-buying business models in Internet-based selling. In P.B. LOWRY, J. O. CHERRINGTON, and R. R. WATSON, eds., Handbook of Electronic Commerce in Business and Society. Boca Raton, FL: CRC Press.

LAI, H. 2002. Collective bargaining models on the Internet. International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, e-Medicine on the Internet, L'Aquila, Italy, July 29–August 4.

LECHETA, R. R. Google Android: aprenda a criar aplicações para dispositivo móvel com google sdk. São Paulo: Novatec, 2010.

OPEN HANDSET ALLIANCE. **Members.** Disponível em: <http://www.openhandsetalliance.com/oha_members.html>. Acesso em: 13 jun 2012.

TOLEDO, L. A.; CARO, A. 2006. **Fatores críticos na adoção da compra pela Internet:** uma análise multivariada. Gestão & Regionabilidade. Vol 22 – nº 64. São Paulo.

TORRES, N, A. 2012. **O crescimento do e-commerce no Brasil e suas implicações.** Disponível em: <http://ecommercenews.com.br/artigos/cases/o-crescimento-do-e-commerce-no-brasil-e-suas-implicacoes>. Acesso em: 08 jun 2012.

TURBAN, E.; MCLEAN, E.; WETHERBE, J. **Tecnologia da informação para gestão:** transformando os negócios na economia digital. 3ed. Porto Alegre, Bookman, 2004.

WORLD WIDE WEB CONSORTIUM. **Simple Object Access Protocol (SOAP 1.1).** Disponível em: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Acesso em: 15 jun 2012.

WORLD WIDE WEB CONSORTIUM. **Web services architecture.** Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em: 15 jun 2012.

WORLD WIDE WEB CONSORTIUM. **WSDL and UUID.** Disponível em: <http://www.w3schools.com/wsdl/wsdl_uddi.asp>. Acesso em: 15 jun 2012.

ZARELLI, G, B. 2012. **Como funciona o soap. Protocolo simples de acesso a objetos.** Disponível em: <http://zarelli.wordpress.com/2012/03/22/como-funciona-o-soap-protocolo-simples-de-acesso-a-objetos/>. Acesso em: 15 jun 2012.