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

ACYR JOSÉ TEDESCHI

"DISPOSITIVOS MÓVEIS PARA *MARKETING* DINÂMICO"

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA

2011

ACYR JOSÉ TEDESCHI

"DISPOSITIVOS MÓVEIS PARA *MARKETING* DINÂMICO"

Monografia apresentada como requisito parcial à obtenção do grau de Especialista em Tecnologia Java. Universidade Tecnológica Federal do Paraná.
Orientador Prof. : Paulo Bordin

CURITIBA

2011

À Anastácia

AGRADECIMENTOS

Pela ajuda na conclusão deste trabalho: Ao orientador Paulo Bordin e aos Professores Nelson Kahima e João Alberto Fabro.

Pelos sábios conselhos sobre a vida e apoio sempre presente: À minha mãe.

Muito obrigado.

Fiat Lux

RESUMO

O crescente desenvolvimento da telefonia móvel, resultado de sua adoção cada vez mais ampla, tem transformado os costumes e padrões de nossa sociedade contemporânea.

O poder de processamento de um dispositivo móvel já é alto o suficiente para que seus usuários tenham diversas experiências relacionadas à ubiquidade da Internet. Neste contexto, o papel que tal recurso tecnológico assume, quase que passivamente, dado que não se dá conta desta inserção, é cada vez mais fundamental para o desempenho de suas atividades e lazer.

Se, num primeiro momento, há o deslumbre pela quantidade de coisas agora possíveis ao toque dos dedos, tais como: Procurar a melhor rota para uma viagem, receber notícias assim que são divulgadas, ver seus programas de televisão favoritos, ouvir música e até fazer ligações. No instante seguinte o consumidor toma uma postura mais pró-ativa, desejando que mais do mundo reaja à existência dos celulares.

Assim, este trabalho tem como objetivo aliar a presença dos dispositivos móveis, a Internet e os interesses dos usuários e propor uma solução que torne possível um dispositivo de mídia "reagir"de acordo com os que estão mais próximos à ele, disponibilizando publicidade segundo seus interesses.

LISTA DE FIGURAS

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

1.1 Telefonia Celular

Trinta e cinco anos depois do primeiro modelo ter sido idealizado, o Motorola DynaTAC 8000X [3], a evolução da telefonia passou por quatro gerações:

1G Sistema chamado Advanced Mobile Phone System, fruto de pesquisas da Bell Labs. [29, 34]

2G Transmissão de sinais digitais, início do SMS (Short Message Service) e de *emails*. [29]

3G Alta qualidade na transmissão de dados e voz. Dispositivos utilizados para máquina de jogos, multimídia, leitor de *e-books* e navegação web. [34]

4G Segundo a ITU, [32] as características que a definem são: Serviços móveis de alta qualidade; Dispositivos para uso em qualquer lugar no mundo; *Roaming* internacional e aumento nas taxas de transferência para atender serviços mais avançados

Segundo [33], existem cerca de 5.3 bilhões de telefones celulares pelo mundo. Incluindo 940 milhões usufruindo de serviços 3G. A figura 1.1 mostra o aumento do número de aparelhos desde o ano de 2005.

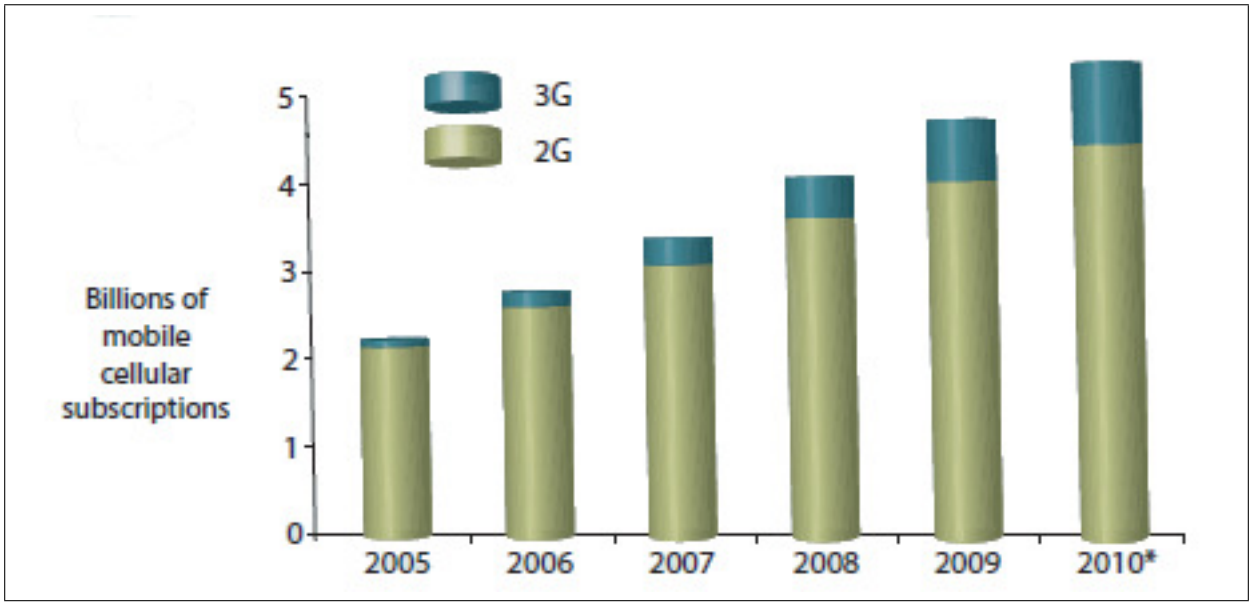


Figura 1.1: Aumento de aparelhos celulares 2005-2010 no mundo. Fonte ITU [33].

Indicadores	Unidade	2005	2006	2007	2008	2009	2010
Expansão do Setor							
Total de Telefones	Milhões	120,6	138,7	160,4	191,8	215,4	245,0
Densidades	Qtde/100 habitantes	68,1	73,9	84,3	99,5	112,1	126,4
Serviço Móvel							
Acesso Móvel Pessoal	Milhões	86,2	99,9	120,9	150,6	173,9	202,9
Densidade	Qtde/100 habitantes	46,6	53,2	63,6	78,1	90,5	104,7

Tabela 1.1: Expansão do setor de telefonia. Fonte Anatel. [11]

O acesso à redes móveis está disponível a 90% do população mundial e a 80% dos que vivem em zonas rurais. A demanda por serviços 3G aumentou, em 2007 eram 95 os países que ofereciam tais serviços comercialmente, contra 143 em 2010.

Trazendo a atenção para o Brasil, o número de celulares foi de 202,9 milhões em 2010, dados da Anatel [11]. Com uma densidade de 104,7 a cada 100 habitantes. Segundo [31], em abril deste ano, os números eram de 212,6 milhões de celulares e uma densidade de 109,3/100 habitantes. A tabela 1.1 mostra o crescimento do setor desde 2005.

Segundo Adriana de Souza e Silva [28], serão os dispositivos móveis, e não os computadores pessoais, que irão romper a barreira digital em países em desenvolvimento. Desta forma, lugares com aspectos sócio econômicos parecidos com os do Brasil, tendem a saltar a segunda onda da computação, indo diretamente à terceira: Computação Ubíqua.

Segundo Mark Weiser [36], são três as ondas da computação: *Mainframes*, um computador para muitos usuários; Computadores Pessoais, um computador para um usuário e, finalmente; Computação Ubíqua, um usuário para muitos computadores.

Da acordo com a publicação do Comitê Gestor da Internet no país, [10], 81% dos celulares vendidos no país no ano de 2009 pertenciam à planos pré-pagos, uma modalidade que popularizou o acesso à mobilidade com alguma economia.

Além da própria condição financeira, um importante fator de influência neste salto está na infraestrutura precária presente em regiões mais afastadas dos grandes centros. E mesmo no meio destes, lugares densamente populados mas informais, como favelas, que não atraem a iniciativa privada e portanto, são esquecidos. Em ambos os casos, as tecnologias sem fio tornar-se-ão as grandes responsáveis pela inclusão digital, transpondo as barreiras físicas e comerciais.

Sua versatilidade e disseminação permitiu uma utilização muito diversa da pensada para o DynaTAC 8000X em 1973, seus sucessores agora são responsáveis por receber informações sobre o clima e permitir que agricultores antecipem os cuidados com a lavoura. Fuliões acompanham seus trio elétricos favoritos através de celulares. Pais são acionados, através de mensagens, quando seus filhos tem problemas com frequência. Policiais utilizam de semelhante tecnologia para consultar a placa de carros suspeitos e finalmente, índios utilizam dispositivos móveis para medir as condições da água do mar onde cultivam suas

ostras. [10]

Este crescimento traz consigo a demanda por novos, e inovadores, serviços que atendam as especificidades dos dispositivos móveis, bem como às novas maneiras de interação que o público tem com as redes ubíquas presentes na sociedade contemporânea, [19, 27, 18].

Campbell e Park em [4], mencionam o avanço de uma sociedade da comunicação pessoal, que substitui a antiga comunicação de massa. Onde o próprio significado simbólico da tecnologia sofreu mudanças e a característica principal está na criação de nós descentralizados, baseados em interesses a priori, individualizados, que somados criam um sentido de grupo expressivo. Agora a tecnologia faz parte do vestuário e parte da socialização com seu grupo depende dela.

Wang e Wu, [35], apresentam um exemplo de um serviço neste contexto inovador. Um sistema de educação à distância que, utilizando dispositivos móveis e redes úbiquas, oferecem ao estudante recomendações de aprendizado adapatadas ao ambiente físico no qual está inserido. Tais recomendações levam em consideração seu histórico escolar e ajudam a tornar a ausência de um tutor menos significativa para o processo de ensino daquele aluno.

1.2 Objetivos gerais

1. Desenvolver um sistema que permita uma segmentação mais refinada nas mensagens de *marketing* dirigidas ao consumidor final

2. Utilizar das informações pré configuradas, pelo próprio consumidor, para adequar tais mensagens às suas preferências

1.3 Objetivos específicos

1. Desenvolver aplicativos para dispositivos móveis que, a partir da anuência de seu proprietário, sejam capazes de armazenar e transmitir suas preferências pessoais para dispositivos de publicidade de maneira automática e através de redes sem fio

2. Desenvolver o sistema que atue como uma interface com os dispositivos de publicidade para que estes recebam as informações dos dispositivos móveis

3. Estudar as potencialidades das linguagens de programação capazes de realizar a tarefa e elencar as mais abrangentes no que diz respeito à *hardware* e arquitetura

1.4 Metodologia Empregada

O sistema contará com *softwares* agentes instalados em quaisquer dispositivos móveis. Terão como funções: armazenar as preferências de seus usuários e transmiti-las utilizando-

se protocolos apropriados para redes sem fio.

Software servidores controlarão o recebimento das transmissões feitas pelos agentes inseridos naquele ambiente e, através de um algoritmo de avaliação de peso, exibirá em um dispositivo de publicidade informações relacionadas às preferências mais comuns aos usuários naquele momento.

O peso dado à cada preferência será mensurado através do número de pessoas que compartilhem mesma opinião sobre determinado assunto.

CAPÍTULO 2

REVISÃO TECNOLÓGICA

2.1 Java

Segundo [9], um projeto batizado de "*Green*"tinha como objetivo desenvolver uma linguagem de programação capaz de ser utilizada em dispositivos com baixa capacidade de processamento e memória, também precisava ser pequena em tamanho. Tais dispositivos também poderiam variar seu processador, portanto, a nova linguagem não poderia depender de uma única arquitetura.

Todas estas exigências levaram a equipe daquele projeto a uma linguagem chamada UCSD Pascal, criada por Niklaus Wirth no início da computação. O resultado de ambos os projetos era uma linguagem portável que gerava código intermediário para uma máquina hipotética, contanto que esta tivesse o interpretador correto.

A empresa, Sun, baseou seus trabalhos no C++ e orientou sua linguagem à objetos, duas diferenças do modelo adotado por Wirth, que tinha como fonte o Pascal e a orientação à procedimentos. Como resultado, chegaram até a linguagem de programação Java.

Os produtos gerados através do projeto "*Green*"não tiveram êxito comercial até que, em 1996, o Netscape 2.0 foi lançado com a capacidade de executar applets Java. Isto foi determinante para o sucesso da linguagem. Em maio daquele ano, na conferência JavaOne, o pessoal da Sun Microsystems esboçou uma série de aprimoramentos e novas bibliotecas, alterando radicalmente o destino previamente decidido pelo projeto "*Green*".

O Java Micro Edition também surgiu na conferência JavaOne, só que da que ocorreu no ano de 1999. Tinha como alvo desenvolvedores de dispositivos móveis, com baixo poder de processamento, de memória e pequena. Também precisavam de interoperabilidade [20].

Exatamente as mesmas necessidades que o projeto "*Green*"veio suprir, quase uma década antes.

Atualmente, a Oracle define a plataforma J2ME como um conjunto de tecnologias e especificações que, juntas, formam um ambiente de execução Java capaz de atender as requisições específicas de determinado dispositivo ou mercado. [26].

Três elementos são a base da tecnologia:

1. **Configuração** Provê o conjunto de bibliotecas básicas e máquina virtual para uma grande variedade de dispositivos, pode ser dividida em:

   (a) **CDLC** *Connected Limited Device Configuration* para dispositivos menores

   (b) **CDC** *Connected Device Configuration* para dispositivos com melhores configurações

2. **Perfil** Define um conjunto de classes capazes de se adaptar às necessidades dos dispositivos móveis, oferece recursos como rede e armazenamento local. É chamado de *Mobile Information Device Profile* (MIDP)

3. **Pacotes opcionais** Disponíveis como um conjunto de *Application Programming Interfaces* (API) específicas para cada tecnologia

A figura 2.1 mostra os componentes da tecnologia J2ME e como eles se relacionam ao universo Java.

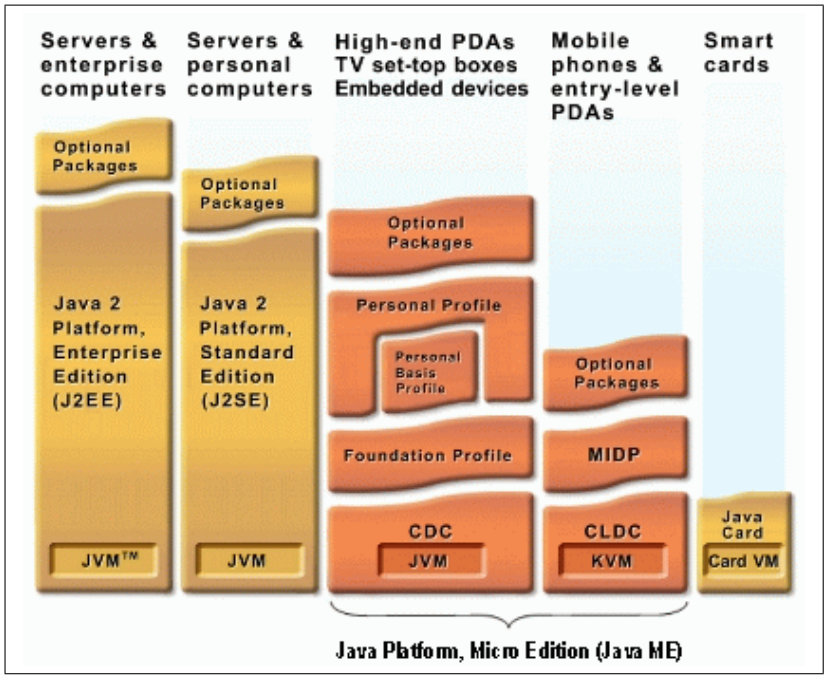


Figura 2.1: Componentes da plataforma J2ME. Fonte: Oracle, J2ME [26]

2.2 Android

Abstrair a árdua tarefa de controlar cada requisição feita à miríade de componentes de *hardware* de um dispositivo computacional é o objetivo dos sistemas operacionais. É ele quem fornece uma interface mais simples para que os programas de usuário consigam cumprir suas tarefas.

Desta forma, um sistema operacional tanto pode ser visto como uma máquina estendida, que atua escondendo toda a complexidade de atuar diretamente com o *hardware*, quanto pode ser visto como gerenciador de recursos, que fornece uma alocação ordenada e controlada de seus recursos aos programas que competem por eles.

Também os sistemas operacionais tiveram fases de evolução bem distintas: válvulas e painéis de programação; transistores e sistemas em lote; multiprogramação; computadores pessoais e, finalmente, a dos dispositivos móveis e embarcados. [30]

Como resultado, uma variedade bem ampla de sistemas operacionais tornou-se disponível. Cada qual com o seu nicho e características próprias. Dentre os que tratam de telefones celulares pode-se citar o Android, que, de acordo com a Open Handset Alliance, [1], é um conjunto completo de *softwares* para dispositivos móveis: sistema operacional, *middleware* e aplicações móveis.

Sua história começa numa empresa chamada Android Inc, que mais tarde foi comprada pela Google. Seu lançamento, feito em 05 de novembro de 2007 ocorreu quando a Open Handset Alliance foi formada, liderada pela Google.[15, 22]

Desenvolvido com base no kernel 2.6.0 do Linux, herda todos seus conceitos de administração de memória, recursos, segurança, processos e threads. Praticamente todas as partes da plataforma utilizam a licença Apache 2.0, o que mantém a plataforma aberta à contribuições por parte de fabricantes e desenvolvedores ao redor do mundo.

Tem uma interface rica e é altamente flexível, tornando possível que uma aplicação consiga interagir com qualquer funcionalidade do aparelho. Não há diferenças entre algo que tenha sido criado por terceiros e o que é nativo da plataforma.

Sendo multitarefa, cada aplicação é executada em seu próprio processo no sistema operacional, que por sua vez possui uma *thread* dedicada. Bem como um usuário lhe é criado de maneira exclusiva e, sendo assim, uma estrutura de arquivos segura para sua execução.

O Dalvik, uma máquina virtual customizada para a economia de memória e recursos de *hardware* para dispositivos, é utilizada ao invés de uma máquina virtual Java. Desta forma, mesmo a aplicação sendo desenvolvida com todos os recursos da Java, seus *bytecodes* serão convertidos para o formato .dex, executável Dalvik, e então empacotados juntamente com o restante dos recursos em um arquivo compactado .apk, Android Package File.

O núcleo da arquitetura do sistema é apresentado na figura 2.2 e algumas de suas características estão listadas a seguir. [16, 22]

Dispositivos Pode ser utilizado em *smartphones*, *netbooks*, *tablets* e na Google TV. Adaptando-se à dispositivos Video Graphics Array, VGA. Suporta bibliotecas gráficas 2D e 3D, sendo estas baseadas nas especificações da OpenGL ES 3.0.

Persistência Utiliza o banco de dados relacional SQLite para armazenar dados.

Conectividade Suporta as tecnologias GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC e WiMAX.

Mensagens SMS e MMS são formas disponíveis para envio de mensagens. O serviço Android Push Messaging agora conta com o *framework* Android Cloud To Device Messaging, C2DM.

Navegador O navegador disponível para o Androi é baseado no motor do WebKit, com a adição do motor V8 JavaScript utilizado no Chrome.

Suporte Java Classes Java são transformadas em executáveis Dalvik. Através de aplicações de terceiros o J2ME pode ser utilizado.

Multimídia Os seguintes formatos estão disponíveis para uso: WebM, H.263, H.264, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, FLAC, WAV, JPEG, PNG, GIF e BMP.

***Streaming* multimídia** Existe suporte para RTP/RTSP *streaming*, bem como para o *download* progressivo do HTML, tag <video> do HTML5. Através do *plugin* Flash podem ser utilizados o Adobe Flash Streaming, RTMP e o HTTP Dynamic Streaming. O HTTP Live Streaming da Apple através do RealPlayer para dispositivos.

Hardware A plataforma pode fazer uso de câmeras de vídeo, telas sensíveis à toque, GPS, acelerômetros, giroscópios, magnetômetros, controles de vídeo games dedicados, sensores de pressão e de proximidade e termômetros. A principal plataforma de *hardware* utilizada é a arquitetura ARM. Existe suporte para a arquitetura x86 através do projeto Android-x86 e o Google TV utiliza uma versão modificada deste último.

Multi-toque Android tem suporte nativo à multi-toque.

Bluetooth Suporte à envio de arquivos via A2DP e AVRCP, acesso à agenda de telefones, discagem através da voz. O suporte à teclados, *mouse* e a *joystick* está disponível através de aplicações de terceiros.

Vídeo chamadas Não há suporte nativo, mas alguns dispositivos têm uma versão otimizada do sistema operacional que permitem sua realização, através de rede UTMS ou sobre IP.

Acionamento através de voz Buscas através do Google estavam disponíveis desde o lançamento da primeira versão. Acionamentos de discagens, digitação e navegação estão presentes após o Android 2.2.

Tethering A arquitetura permite que o dispositivo torne-se um ponto de acesso para redes com e sem fio. Antes da versão 2.2 este suporte era disponibilizado através de terceiros

Desensolvimento O desenvolvimento de aplicações pode ser feito em Java através do Android Software Development Kit; em C e C++ através do Native Development Kit e, finalmente, através do Google App Inventor para iniciantes.

O Android Market [17] tem como proposta centralizar a distribuição de aplicativos para a plataforma. E desta maneira ajuda na divulgação da plataforma e na visibilidade de cada aplicação construída para ela, também cria uma referência onde seus usuários possam procurar *softwares* que atendam suas necessidades. O desenvolvedor precisa pagar uma taxa de inscrição para disponibilizar suas aplicações e 70% dos lucros através das vendas são repassados à quem as construiu.

Um aplicativo pré-instalado nos dispositivos, chamado Android Market, torna a busca e instalação de novas ferramentas simples para todos os usuários.

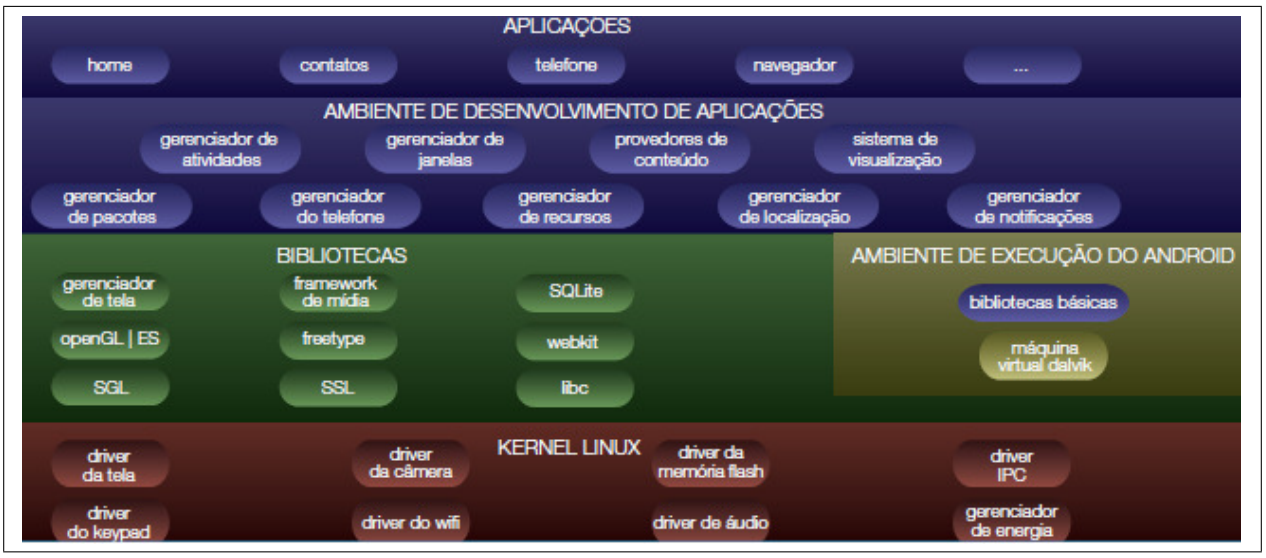


Figura 2.2: Núcleo da Arquitetura da Plataforma Android. Fonte: Developers Android [16].

Além do Android, existem outras plataformas que atuam na esfera de dispositivos móveis, à saber:

bada Mantido pela Samsung [2]. Trata-se de um ambiente de desenvolvimento no qual se pode escolher que *kernel* será utilizado

LiMo Consórcio dedicado à criação do primeiro sistema operacional verdadeiramente aberto e independente de *hardware*, baseado em Linux, para dispositivos móveis. [12]

MeeGo União dos projetos Moblin, da Intel, e do Maemo, da Nokia. Atua como sistema operacional para *netbooks*, *tablets* e *smartphones*. [23]

Symbian Primeira plataforma desenvolvida para *smartphones*, está presente em aproximadamente 2/5 de todos os que existem no mundo. Mantido pela Nokia. [25].

WebOs Sucessor do PalmOS e mantido pela HP, foi completamente re-escrito dada a transferência de direitos de código à Palm Source e também para poder concorrer com as outras plataformas disponíveis. *investorPalm*.

Windows Phone Desenvolvido pela Microsoft, sucessor da plataforma Windows Mobile, destina-se ao mercado de consumo. [24].

2.3 Simple Object Access Protocol

Segundo [7], é um protocolo leve baseado em XML para a troca de mensagens em um ambiente descentralizado e distribuído. Seus principais objetivos são simplicidade e extensibilidade e uma de suas utilizações é a de encapsular e transportar chamadas remotas à procedimentos. Faz uso do protocolo HTTP para o transporte de suas mensagens.

Segundo a especificação do protocolo, as chamadas são executadas a partir das seguintes informações:

1. Um identificador uniforme de recursos (URI) para o objeto alvo
2. O nome do método
3. Os parâmetros do método
4. Opcionalmente, uma assinatura para o método
5. Opcionalmente, um cabeçalho

Uma mensagem SOAP deve conter os seguintes elementos, conforme figura 2.3:

Envelope Elemento raiz do documento XML. Contém declarações de *namespaces* e atributos adicionais que definem a representação dos dados. O nome deste elemento deve ser Envelope

Cabeçalho Carrega informações opcionais. Tipicamente utilizado para tratar de informações referentes à autenticação e controle de transações. O nome deste elemento deve ser Header

Corpo Elemento obrigatório, carrega a informação que deve ser transportada. Opcionalmente, pode conter o elemento Fault, que transporta registros de erros gerados pelos nós que processaram aquela mensagem. O nome deste elemento deve ser Body

Todas as mensagens SOAP são codificadas usando XML e portanto, uma aplicação deve incluir o *namespace* apropriado em todos seus elementos e atributos nas documentos que gera. Bem como deve ser capaz de processar tais *namespaces* quando recebe mensagens.

Um exemplo de mensagem SOAP pode ser visto no Código Fonte 4.5, página 51. Das linhas 1 à 4 têm-se o cabeçalho, definindo o tamanho do pacote e tipos de dados transmitidos. Logo em seguida, o campo SOAPAction define qual o propósito daquela mensagem.

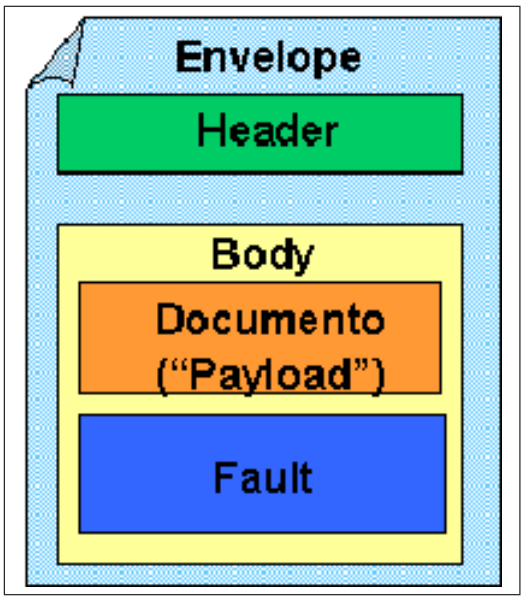


Figura 2.3: Elementos de uma mensagem SOAP

No corpo daquela mensagem, linhas 10 a 14, o método que deve ser chamado: GetLast TradePrice, incluindo o parâmetro que deve ser utilizado, no exemplo: DIS.

2.4 Web Services

Segundo [21] um *web service* é uma funcionalidade de uma aplicação, criada seguindo os padrões para Internet, que está acessível através da rede. Ou seja, se a aplicação pode ser acessada através da rede utilizando protocolos como HTTP, XML, SMTP ou Jabber, então é um *web service*.

Atrás de uma definição, aparentemente simples, tem-se uma camada de abstração que permite a interoperabilidade entre sistemas distribuídos. Do ponto de vista do cliente, que consome o Web Service, não importa em que linguagem foi implementado, ou ainda, em que plataforma está sendo executado. O inverso também é verdadeiro, não importa, para o servidor, como foi criado o cliente, ou ainda, para qual sistema operacional foi escrito.

O grupo W3C (World Wide Web Consortium) define *web services* em torno desta característica: Um Web Service é um *software* planejado para suportar a interação interoperável, máquina à máquina, através da rede. Tem uma interface descrita em um formato processável por computadores, especificadamente WSDL (Web Service Description Language). Outros sistemas interagem com ele através desta descrição utilizando mensagens SOAP (Simple Object Access Protocol), transportadas por HTTP e serialização em conjunto com outros padrões relacionados à Web. [5]

Algumas definições que gravitam ao redor de *web services*:

Agentes e Serviços Um serviço é um conceito abstrato que precisa ser implementado

através de um agente. Este último, escrito em uma linguagem de programação, de fato envia, processa e recebe mensagens. O mesmo serviço que retorna a soma de dois números pode ser implementado por dois, ou mais, agentes distintos.

Solicitantes e Provedores Uma entidade provedora, *provider entity*, é uma pessoa ou organização que provê um agente que implementa um serviço em particular. Já, uma entidade solicitante, *requester entity*, é a que deseja fazer uso de determinado Web Service e, para tal, implementa um agente, *requester agent*, para a troca de mensagens com o *provider agent*.

Descrição do Serviço A mecânica da troca de mensagens é documentada no Web Service Description (WSD), uma especificação processável por computadores, escrita em WSDL. Ela define formatos de mensagens, tipos de dados, protocolos de transporte e o formato de serialização que deverão ser utilizadas entre os agentes solicitante e provedor.

Semântica É a expectativa sobre o comportamento do serviço, em particular a respeito das respostas às mensagens que lhe são enviadas. É o contrato feito entre o solicitante e o provedor.

A invocação de um Web Service, de maneira geral, vista na figura 2.4, pode ser descrita da seguinte maneira:

1. O solicitante e o provedor descobrem-se um ao outro, ou pelo menos, um deles o faz. O que significa dizer que, de alguma maneira, têm-se o endereço um do outro. Uma das maneiras de se conseguir tal endereço é diretamente com a entidade, seja ela solicitante ou provedora; Outra é utilizando diretório de serviços, para encontrar um que atenda as especificações desejadas. Aqui a busca pode ser efetuada manualmente ou ainda, de maneira autônoma, em ambos os casos uma descrição funcional junto à uma descrição do serviço (WSD) são consultadas. Aqui cabe a utilização do UDDI.

   Universal Description, Discovery and Integration, UDDI, é um *framework* para descrição de serviços, encontrar negócios e integrar serviços de negócios utilizando a Internet. Enquanto serviço de diretório, armazena informações sobre *web services*, com uma interface descrita em WSDL e utiliza SOAP como protocolo de comunicação [8].

2. Ambas entidades acordam sobre a descrição do serviço e semântica que irá conduzir as mensagens entre seus respectivos agentes.

   O termo acordar aqui significa dizer que ambos têm acesso ao WSDL que descreve o serviço

3. Ambos agentes tornam-se cientes da descrição do serviço e semântica utilizados

   Na prática, os agentes passam a ser implementados de acordo com as descrições encontradas no WSDL. O processo pode ser feito de várias maneiras; a descrição do serviço pode estar inserida diretamente no código do agente; ou ainda, após a criação de um deles, o código do outro pode ser deduzido através de sua semântica.

4. Os agentes trocam mensagens, utilizando o protocolo SOAP.

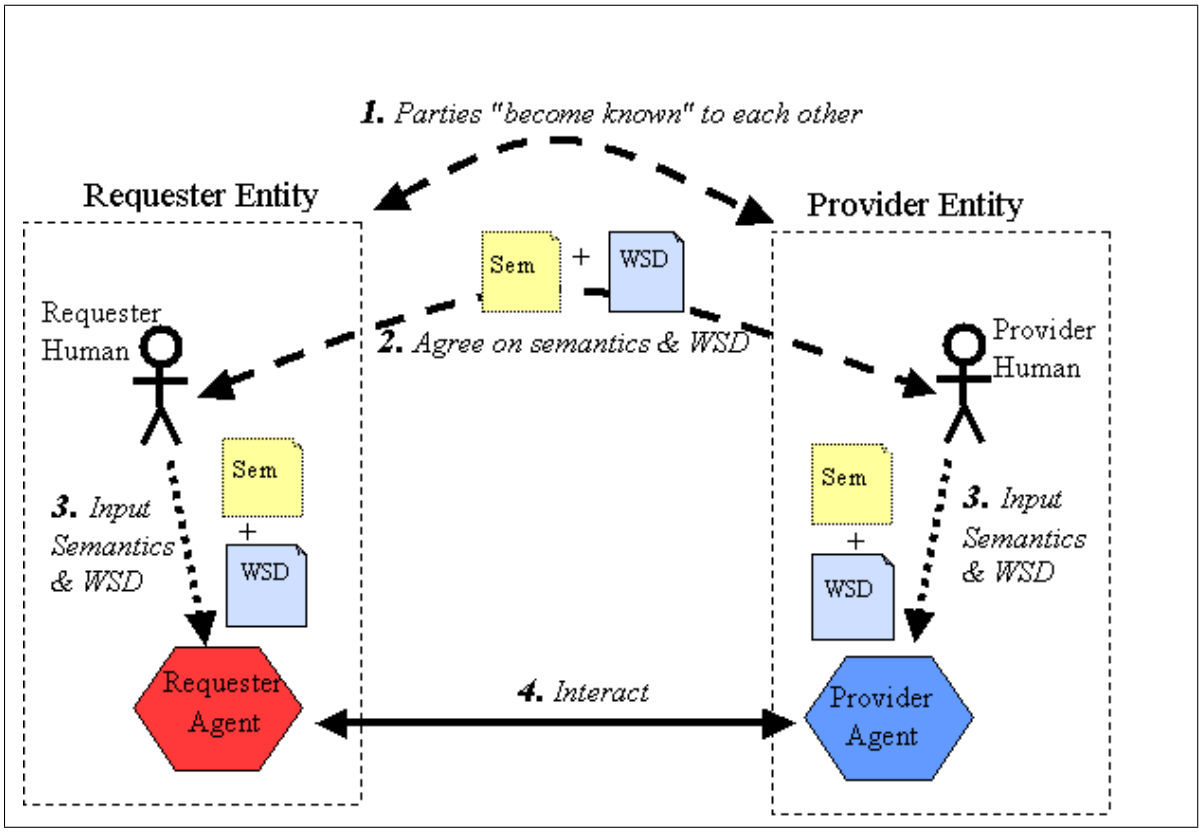


Figura 2.4: Processo geral de como utilizar um Web Service. Fonte: W3C [5]

Um exemplo de serviço implementado em linguagem Java pode ser visto em Código Fonte 4.1, página 43. Trata da porção do sistema exposta pelo servidor para que os clientes tenham acesso às funcionalidades do Web Service e basicamente, recebem os parâmetros de chamada e acionam os respectivos métodos das classes de controle. Linha 17 e 30.

O Código Fonte 4.2, página 44 mostra o WSDL do serviço.

O trecho do código que representa um cliente para este serviço está descrito em: Código Fonte 4.4, página 49. O método `transmitir(E_Dispositivo)` ali descrito faz a chamada ao Web Service e obtém sua resposta.

Para tal, um objeto SOAP é instanciado, linha 30, com a indicação de que serviço deve ser acionado, no exemplo: `oQueDivulgar`. À este objeto são adicionados os parâmetros

de invocação que serão utilizados pelo Servidor, linhas 33 à 40. Logo em seguida, um `envelope` é criado para que o objeto SOAP seja serializado, linha 41. Uma classe, definida a partir da linha 74, precisou ser criada para que objetos tipo `float` pudessem ser utilizados, conforme linhas 44 e 45. O transporte se dá através do protocolo HTTP e é realizado pelo objeto HttpTransportSE, que têm como parâmetros o endereço para o serviço e o `envelope` serializado, como pode-se ver nas linhas 47 e 48. O retorno do serviço é representado pelo objeto `resultado` e visualizado pelo `Toast`, linhas 50 à 54.

A figura 2.5 mostra o ciclo de vida de um Web Service. Alie vê-se que uma aplicação é responsável por criar a mensagem SOAP, que consiste em um documento XML contendo um cabeçalho e um corpo. Através da Internet, protocolo HTTP, é enviada ao destino. Assim que chega através da camada de transporte, a mensagem é encaminhada para a devida aplicação, possivelmente código Java. O que quer dizer, em outras palavras, que um método foi acionado no lado do servidor para tratar daquela mensagem.

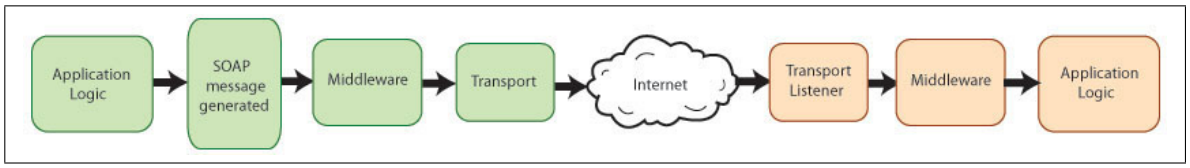


Figura 2.5: Ciclo de vida de uma mensagem Web Service. Fonte: The Apache Foundation [13]

Um ponto relevante a se considerar sobre Web Services é a segurança, a W3C define três padrões para tratar de aplicações que precisam de integridade, autenticidade e privacidade: [6]

XML Signature Esta especificação define uma assinaturas digitais para regras de processamento e de sintaxe, garantindo integridade, autenticação de mensagens e de serviços

XML Encryption Especifica o processo de criptografar dados e representá-los em um documento XML

XKMS É um padrão para registrar e distribuir chaves públicas para que sejam utilizadas com os padrões XML Signature e XML Encryption

2.5 Apache Axis2

Segundo [13], é uma implementação Java orientada à objetos com arquitetura modular que torna o desenvolvimento de itens relacionados às especificações Web Services mais ágil. Oferece suporte à:

1. Envio, recebimento e processamento de mensagens SOAP, inclusive com anexos

2. Criar Web Services a partir de uma classe Java

3. Implementar classes, tanto para servidor quanto para o cliente, a partir de um WSDL

4. Recuperar o WSDL de um serviço

O modo de operação do Axis2 está relacionado ao ciclo de vida de um Web Service, conforme visto na figura 2.5. A próxima figura, 2.6 demonstra como ele trabalha com mensagens SOAP.

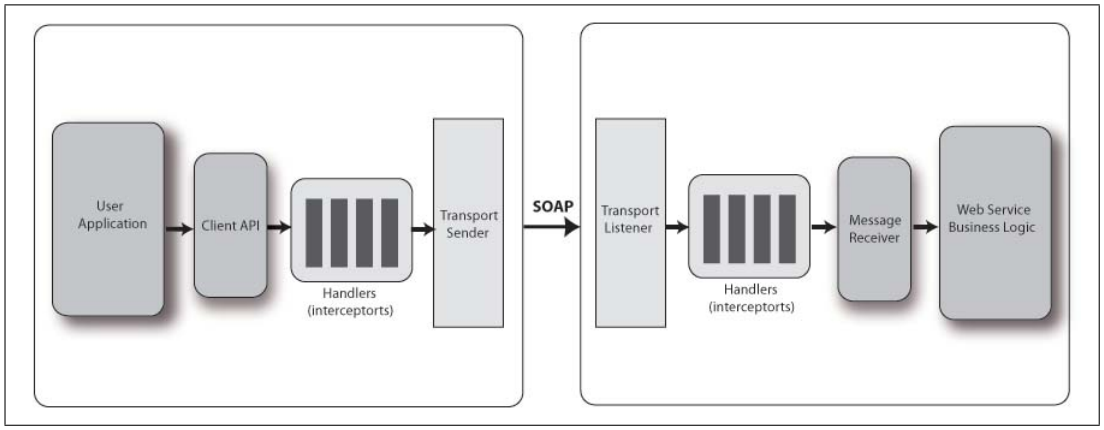


Figura 2.6: Axis2 e mensagens SOAP. Fonte: The Apache Foundation [13]

Em cada ponta existem aplicações que enviam e recebem mensagens. Assumindo que o Axis2 esteja atuando entre elas, em ambos os lados, têm-se o seguinte fluxo:

1. O remetente cria a mensagem SOAP

2. Axis2 realiza as ações necessárias que preparam a mensagem para o transporte

3. A camada de transporte realiza o envio

4. No lado do cliente, a mesma camada recebe a mensagem e a direciona para os devidos módulos

5. Assim que a mensagem for processada pelos módulos, é enviada para a aplicação apropriada

O Axis2 utiliza fases de processamento quando se trata de mensagens SOAP, com por exemplo: "*pre-dispatch*", "*dispatch*"e "*message processing*". Existe controle em cada uma delas e é possível criar as próprias.

O fluxo cita a passagem das mensagens por módulos que são componentes anexados ao Axis2 que permitem ampliar sua funcionalidade.

Para utilizá-lo em projetos que envolvem Web Services, basta copiar o arquivo `axis2.war`, disponível para *download* em [13], para o diretório de aplicações Web de seu servidor

que, por exemplo, pode ser o Apache Tomcat. Os serviços criados são empacotados em arquivos com a extensão `.aar`, basta copiá-los para o diretório `J2EE_HOME/webapps/axis2/WEB-INF/services`, isto se todos os caminhos estiverem de acordo com a instalação padrão.

Se os clientes ainda não tiverem sido implementados, é possível usar as ferramentas incluídas na distribuição do Axis2 para gerar as classes responsáveis para o envio e recebimento de mensagens. Uma delas é o *software* `wsdltojava.sh`, que se utiliza do WSDL do serviço exposto para a criação automática das classes que devem ser utilizadas na sua chamada.

2.6 Global Positioning System

Segundo [14], no início da década de 60, diversas instituições governamentais Norte Americanas tinham interesse no desenvolvimento de um sistema tridimensional de posicionamento. Tal sistema deveria ter as seguintes características:

1. Deveria ter cobertura global e alta precisão
2. Não poderia ser afetado por fatores climáticos
3. Dever ser capaz de trabalhar para observadores em altas velocidades

Em 1969, estabeleceu-se o programa *Defense Navigation Satellite System*, (DNSS), que unificava todos os esforços de criação do sistema de posicionamento e, então, o conceito do sistema NAVSTAR GPS foi criado.

Nos dias atuais, existem vinte e quatro satélites em torno do planeta. Estão divididos igualmente entre seis planos de órbita. Existe uma rede de controle, em terra, que monitora e atualiza informações de cada um deles. Possuem relógios atômicos de alta precisão, transmitem em *broadcast*, utilizando a técnica CDMA, em duas frequências distintas: L1, 1.575,42 MHz e L2, 1.227,6 MHz. Apesar de compartilharem as mesmas frequências, cada um deles utiliza um código de transmissão distinto.

Os receptores GPS operam de maneira passiva, apenas recebendo informações e, sendo assim, um número ilimitado deles pode fazer uso do sistema.

As mensagens de navegação enviadas permitem ao receptor descobrir a posição do satélite naquele momento, através do código de transmissão é possível saber o tempo de propagação do sinal e, portanto, a distância entre ambos. Esta técnica exige sincronia entre eles e quatro satélites para que a localização tridimensional seja calculada.

Se o relógio do receptor estiver sincronizado com os dos satélites e a altura já estiver previamente determinada com precisão, então menos que quatro satélites serão o suficiente para determinar o posicionamento.

Trilateração 3D é o princípio matemático no qual o cálculo de posicionamento é baseado. Se o receptor souber a distância entre ele e quatro ou mais satélites, além da própria localização destes satélites, conseguirá descobrir a sua localização em termos de latitude, longitude e altura na terra.

Além do programa Norte Americano, outros esforços são conduzidos no sentido de se obter um sistema de posicionamento:

GALILEO Em 1998, a União Europeia iniciou os trabalhos para um sistema de navegação por satélite independente do GPS e que fosse de uso civil.

GLONASS System A contra parte Russa ao GPS chama-se *Global Navigation Satellite System*, consiste em uma constelação com quatorze satélites em órbita. O projeto está sendo revitalizado pelo governo Russo.

BeiDou System Iniciativa Chinesa, em fase semi operacional com satélites em órbita geo estacionária em cima daquele país. Desenvolvido para uso civil e militar.

CAPÍTULO 3

SOLUÇÃO PROPOSTA

A solução proposta baseia-se em algumas premissas:

1. Os consumidores finais desejam assumir um papel mais ativo em relação à propaganda à que estão expostos, de modo que vejam o que lhes é interessante

2. Os estabelecimentos comerciais precisam aumentar a assertividade de sua propaganda, para que melhorem suas margens de lucros

Estas premissas estão relacionadas aos objetivos descritos no Capítulo 1.2, página 4 e a modelagem do sistema partiu do pressuposto que ambos, consumidor final e estabelecimento comercial, têm, à sua disposição, dispositivos móveis com acesso à Internet e ao GPS.

Num ensaio sem detalhes técnicos, o sistema no aparelho celular seria acionado pelo cliente num *Shopping Center* e, a partir daí, uma série de informações seriam transmitidas para um servidor central. Tais informações seriam enviadas para as fontes de propaganda de forma que estas fizessem publicidade de acordo com os interesses dos usuários que estão próximos.

O modelo centraliza todas as mensagens num único servidor, que sabe como calcular a distância entre consumidores e estabelecimentos. Estes últimos precisam acessar as informações do servidor para que possam descobrir quais são os interesses dos clientes e, então, controlar o dispositivo de publicidade de acordo.

O servidor oferecerá métodos de acesso às suas informações através de Web Services, expostos utilizando-se do Axis2, Apache Tomcat e MySql. Os dois outros utilizarão implementação Java para o sistema operacional Android, além dos recursos de acesso à Internet e ao GPS.

Os três elementos da solução podem ser descritos da seguinte maneira:

Cliente É o *software* cliente que será utilizado no dispositivo móvel do usuário que tem interesse em **comprar** algum item. Responsável pelas informações de **procura**

Mídia *Software* que será utilizado no dispositivo móvel do usuário que deseja **vender** seus itens. Será responsável pelas informações de **oferta** e pela divulgação da lista de itens desejados pelos **Clientes**

Servidor *Web services* disponíveis no servidor, é responsável por receber as informação de **procura**, do *software* **Cliente**, e armazená-las em um Sistema Gerenciador de

Banco de Dados (SGBD). Também receberá as informações sobre **oferta**, do *software* **Mídia** e efetuará o cálculo de distância geográfica dos dois. Depois comparará suas respectivas listas de **procura** e **oferta**. Se houver compatibilidade de interesses e os dispositivos estiverem próximos, o servidor retornará uma mensagem ao *software* **Mídia** contendo a lista de itens para divulgação

A figura 3.1 demonstra a relação entre os três componentes de *software* e maiores detalhes técnicos podem ser vistos à seguir:

1. O usuário aciona o *software* **Cliente** e cadastra os produtos que tem interesse. O armazenamento é feito localmente, utilizando o SGBD SqLite

2. Ainda no dispositivo **Cliente**, o usuário liga o sistema que, a partir de então, irá "perceber"cada mudança de posição geográfica. Para que isto aconteça, um serviço em segundo plano ficará executando continuamente

3. O **Cliente** faz o uso do GPS do aparelho, utilizando-se de recursos da interface `android.location.LocationManager` e da implementação do método `public void onLocationChanged(Location arg0)`, conforme descrito na listagem 4.6. Desta forma, a cada mudança na posição, uma nova `Thread` é iniciada

4. A cada execução desta `Thread`, o sistema **Cliente** lerá o cadastro de interesses gravado no SGBD local e, então:

5. Uma classe de entidade é instanciada, `E_Dispositivo`, contendo uma identificação daquele dispositivo móvel, o *timestamp*, sua localização em termos de latitude e longitude e, finalmente, os itens de interesse daquele usuário. A transmissão desta classe é feita através da Internet, utilizando-se recursos oferecidos pela API KSOAP2, que cria um envelope SOAP e aciona o método remoto `procuro(int idDispositivo, float latitude, float longitude, long timestamp, String registrosVector)`

6. O Web Service aciona, no **Servidor** os métodos necessários para que a informação do **Cliente** seja armazenada no SGBD MySQl, centralizando a informação de todos os **Clientes**. A cada mudança na posição geográfica de algum dos **Clientes**, uma nova transmissão de dados será feita, apagando o registro anterior daquele **Cliente** específico, substituindo com as novas informações

7. O *software* que faz uso dos registros gravados pelo **Cliente** é o **Mídia**. O primeiro passo para tal é o cadastro de itens ofertados pelo usuário no sistema **Mídia**. Este armazenamento também é local, utilizando o SGBD SqLIte

8. Ainda no dispositivo **Mídia**, o usuário aciona o sistema que irá ser executado a cada ciclo de tempo. Para tal, um serviço em segundo plano será executado continuamente

9. A cada intervalo de tempo, uma nova `Thread` lerá as informações gravadas no SGBD local e, então, uma classe de entidade será instanciada, `E_Dispositivo`, contendo as informações daquele dispositivo, o *timestamp*, sua localização em termos de latitude e longitude, itens ofertados e a distância máxima para **Clientes**

10. A partir daqui, através da Internet e utilizando-se dos recursos da API KSOAP2, um envelope SOAP é criado e o método remoto `public String oQueDivulgar(int idDispositivo, float latitude, float longitude, long timestamp, String registrosVector, int distanciaMetros)` é executado

11. O Web Service, no **Servidor** os métodos necessários para buscar no SGBD MySql os registros de **Clientes** que procuram o que está sendo ofertado por aquele dispositivo de **Mídia**, nas proximidades, nos últimos minutos, são executados. Conforme descrição a seguir:

    (a) Do SGBD temos uma lista com todos os **Clientes** que desejam produtos similares à oferta cuja hora de inserção na base seja menor que o tempo especificado pelo dispositivo **Midia**. Neste ponto, ainda não há qualquer cálculo de distância envolvido

    (b) De posse desta lista, para cada **Cliente** encontrado, é feito o cálculo de distância entre o dispositivo de **Mídia** e o dispositivo **Cliente**. São excluídos os que não estiverem tão perto quanto especificado pelo *software* **Mídia**

    (c) A lista de ofertas restante é ordenada seguindo o critério de maior interesse

    Finalmente, a lista é transmitida de volta para o *software* **Mídia**. O código fonte para a solução pode ser visto na listagem 4.3, página 47

12. De posse da lista, o componente **Mídia** faz a divulgação da lista de produtos

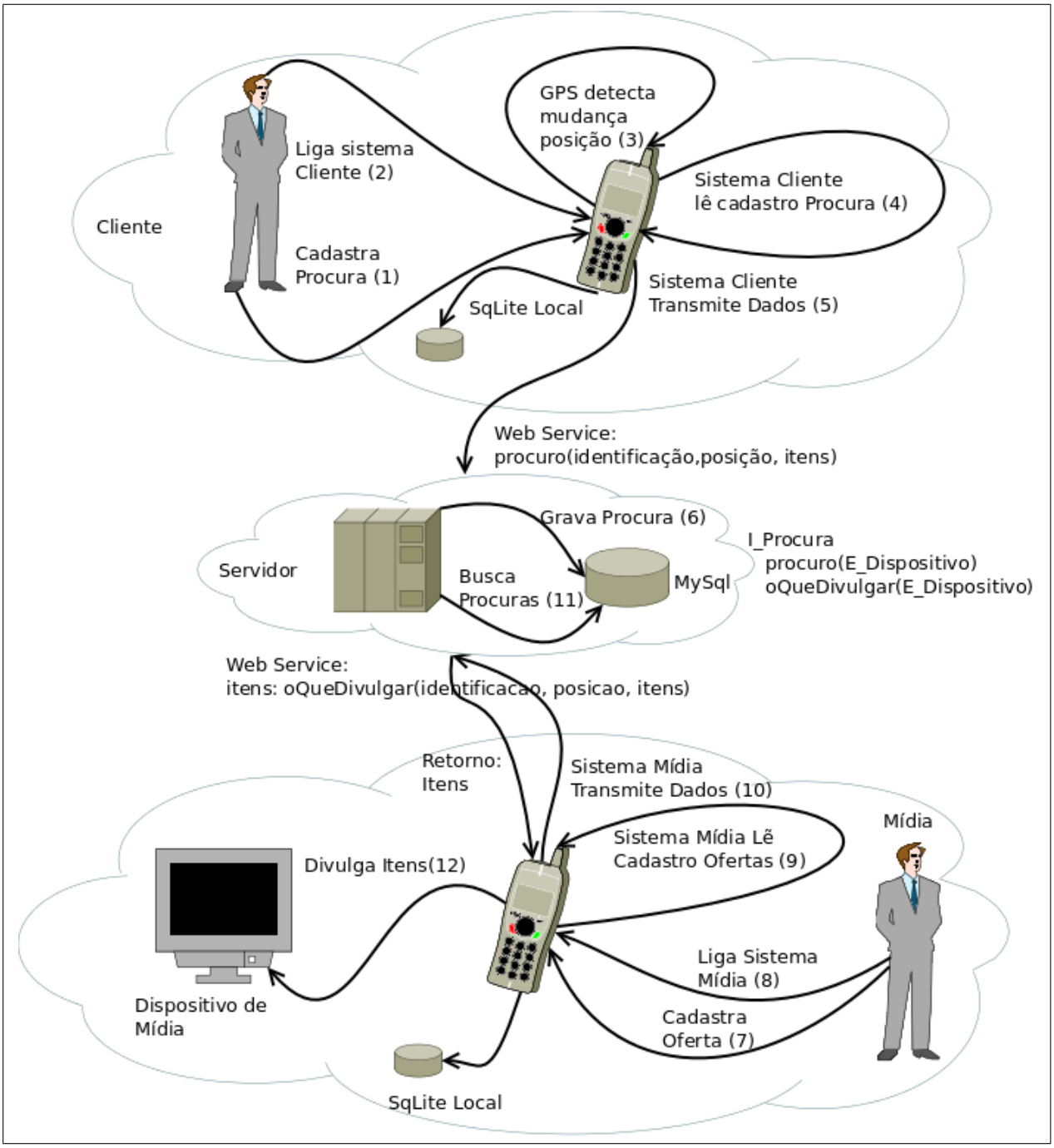


Figura 3.1: Princípio de utilização da solução proposta.

3.1 Cliente

A figura 3.2 descreve seu Diagrama de Casos de Uso. Uma ação do cliente inicia o cadastro de seus interesses, mais detalhadamente: inclusão, remoção, busca e atualização. Todos estes dependem do acesso à base de dados local. Através de um comando do usuário a transmissão das configurações é "desperta"de modo a perceber se houve alteração em sua posição física, utilizando o GPS. Se houve, uma nova transmissão é feita para o servidor.

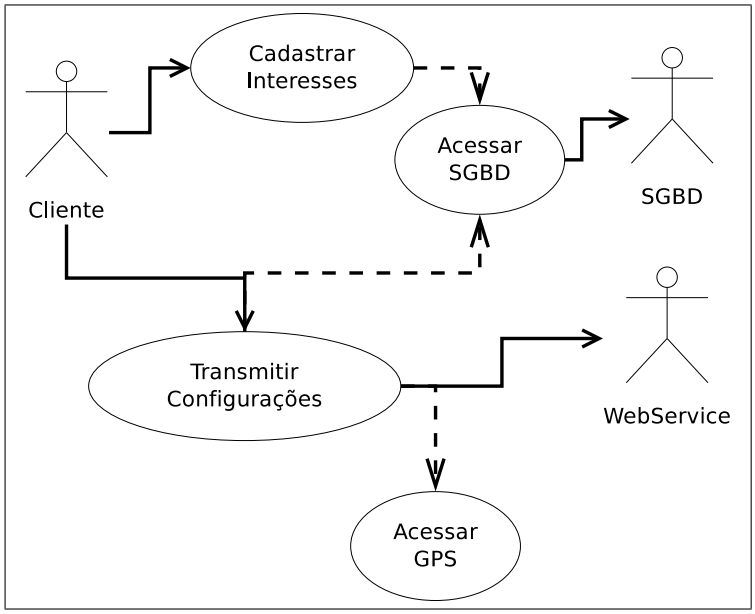


Figura 3.2: Diagrama de Casos de Uso: Cliente.

As classes principais, extraídas a partir da análise do Diagrama de Casos de Uso 3.2 podem ser vistas na figura 3.3.

`I_BancoDados` Classe que faz a comunicação com o SGBD SQLite no sistema

`E_Dispositivo` Contém as informações necessárias para representar um dispositivo, contém uma lista de `E_Registro`

`E_Registro` Contém as informações que definem um interesse do usuário. Faz parte de um `E_Dispositivo`

`C_Configurar` Classe para o controle das ações referentes à persistência dos dispositivos na base de dados

`C_ControlarTransmissoes` Controla os eventos de transmissão para o Web Service do **Servidor**, faz uso do GPS. Seu código fonte pode ser visto na listagem: 4.6, página 52

O Diagrama de Sequência, descrito a seguir, pode ser visto na figura 3.4

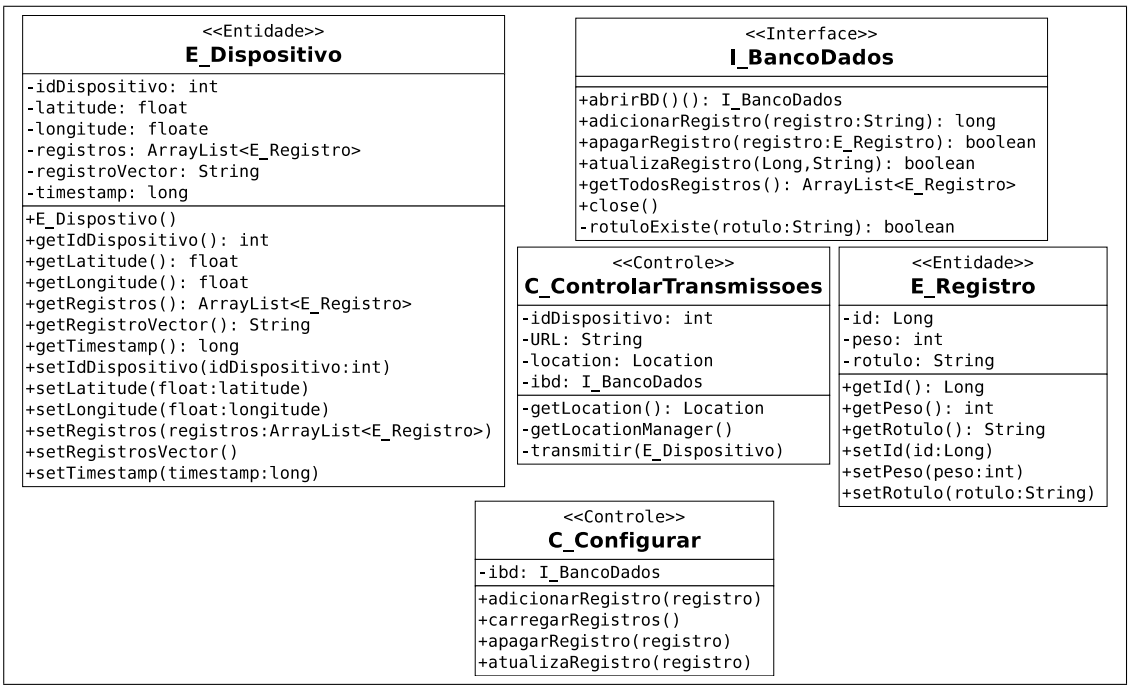


Figura 3.3: Diagrama de Classes: Cliente.

1. O cliente aciona a classe de controle `C_Controle`, que instancia `E_Registro` e, em seguida, faz uso de `I_BancoDados` para persistir os informações no SGBD

2. O cliente dispara a ação que aciona a classe `C_ControlarTransmissoes` que, uma vez em funcionamento, detecta mudanças de geo localização e recuperar as informações de longitude, latitude, identificação do dispositivo e horário para então, utilizando-se de `I_BancoDados`, recupera os interesses do usuário na forma de objetos `E_Registro`. De posse de todos estes dados, monta um objeto `E_Dispositivo` e o transmite para o **Servidor**. Ver listagem 4.6, página 52

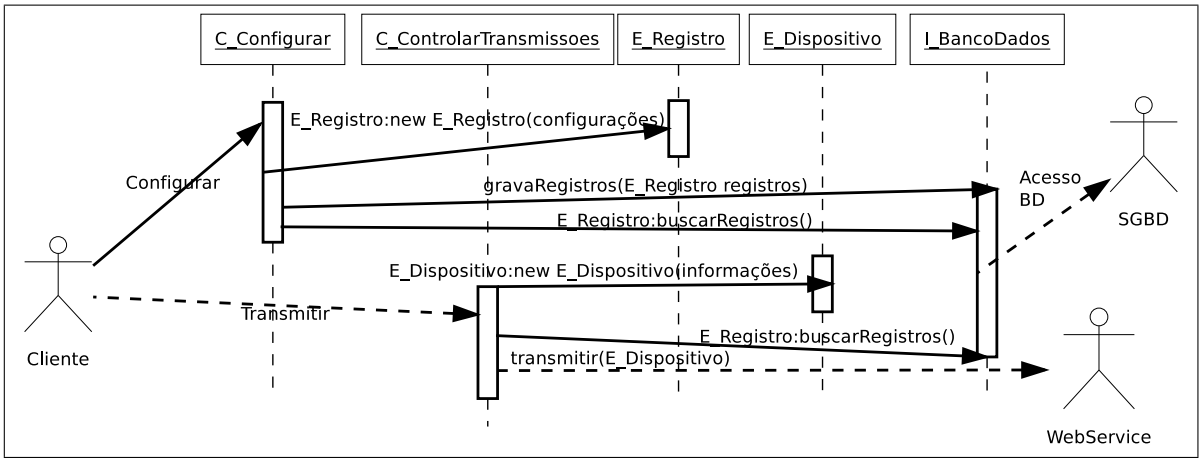


Figura 3.4: Diagrama de Sequência: Cliente.

O Diagrama de Entidade Relacionamento do Cliente é representado na figura 3.5, uma

única tabela contendo os registros de interesses do usuário é necessária para o **Cliente**.

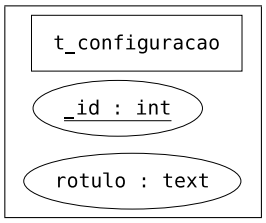


Figura 3.5: Diagrama Entidade Relacionamento: Cliente.

3.1.1 Requisitos funcionais

O sistema deverá ser capaz de:

1. Cadastrar (adicionar, editar, remover) as configurações de interesses de seu usuário
2. Transmitir tais configurações para o servidor

3.1.2 Requisitos não funcionais

1. Será desenvolvido para a plataforma Android
2. O armazenamento local das configurações dos usuários utilizará o banco de dados Sqlite
3. A transmissão das informações ocorrerá de maneira automatizada e levará em consideração a posição física do dispositivo móvel
4. O *hardware* do dispositivo deverá ser possuir um receptor GPS
5. A transmissão será implementada de modo a utilizar os WebServices disponíveis pelo servidor
6. O *hardware* do dispositivo deverá ser capaz de acessar a Internet
7. As devidas permissões de acesso à Internet e ao GPS deverão ser concedidas para que o *software* seja instalado

3.2 Midia

A figura 3.6 descreve seu Diagrama de Casos de Uso. Uma ação do usuário inicia o cadastro de seus itens em oferta, mais detalhadamente: inclusão, remoção, busca e atualização. Todos estes dependem do acesso à base de dados local. Através de um comando do usuário a transmissão das configurações é "desperta"de modo a ser executada em ciclos pré determinados de tempo. As respostas do **Servidor** são mostradas.

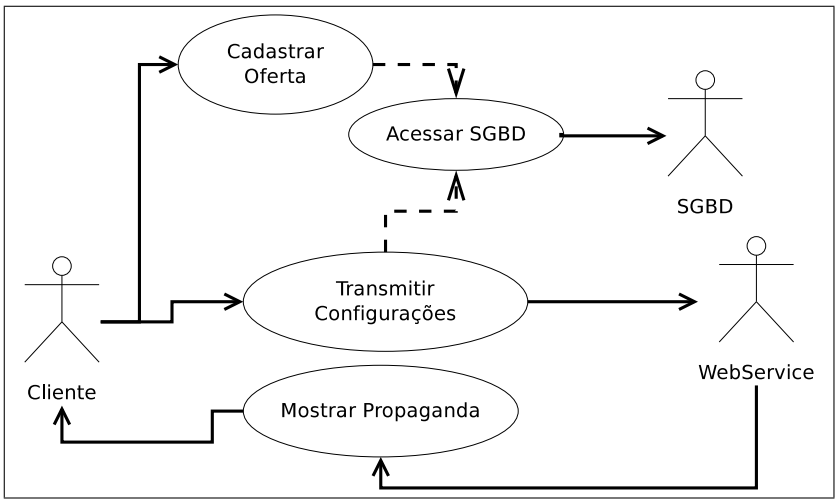


Figura 3.6: Diagrama de Casos de Uso: Mídia.

As classes principais, extraídas a partir da análise do Diagrama de Casos de Uso 3.6, podem ser vistas na figura 3.7 e estão descritas a seguir:

`I_BancoDados` Classe que faz a comunicação com o SGBD SQLite no sistema

`E_Dispositivo` Contém as informações necessárias para representar um dispositivo, contém uma lista de `E_Registro`

`E_Registro` Contém as informações que definem um interesse do usuário. Faz parte de um `E_Dispositivo`

`C_Configurar` Classe para o controle das ações referentes à persistência dos dispositivos na base de dados

`C_ControlarTransmissoes` Controla os eventos de transmissão para o Web Service do **Servidor**, faz uso do GPS. Seu código fonte pode ser visto na listagem: 4.4, página 49

`C_VisualizacaoPropaganda` Controla a visualização da resposta enviada pelo **Servidor**

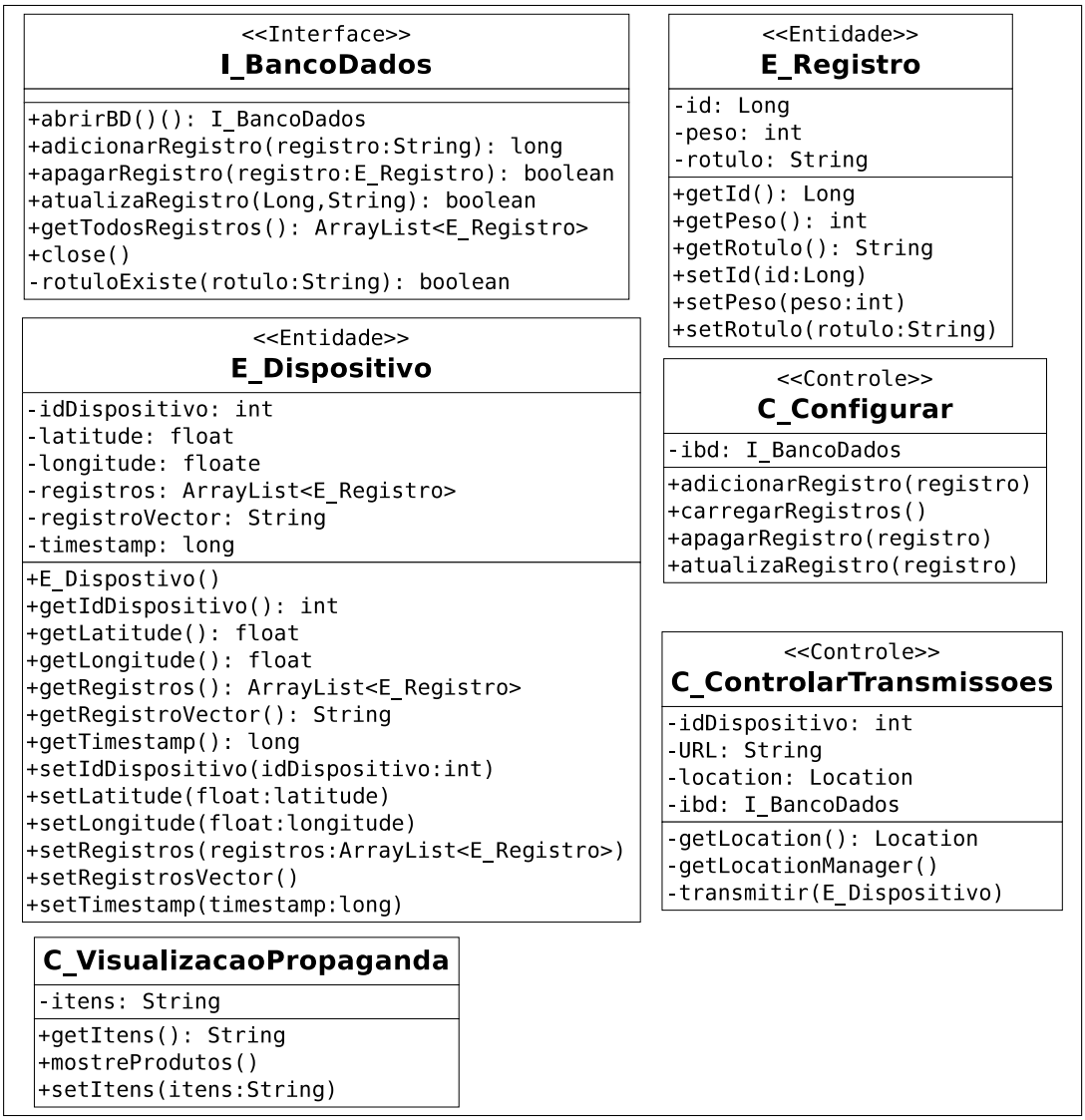


Figura 3.7: Diagrama de Classes: Mídia.

O Diagrama de Sequência, descrito a seguir, pode ser visto na figura 3.8

1. O cliente aciona a classe de controle `C_Controle`, que instancia `E_Registro` e, em seguida, faz uso de `I_BancoDados` para persistir os informações no SGBD

2. O cliente dispara a ação que aciona a classe `C_ControlarTransmissoes` que, uma vez em funcionamento e a cada ciclo de tempo recuperará as informações de longitude, latitude, identificação do dispositivo e horário para então, utilizando-se de `I_BancoDados`, buscará os interesses do usuário na forma de objetos `E_Registro`. De posse de todos estes dados, monta um objeto `E_Dispositivo` e o transmite para o **Servidor**. Ver listagem 4.4, página 49

3. Caso haja resposta do **Servidor**, esta será mostrada utilizando-se da classe `C_VisualizacaoPropaganda`

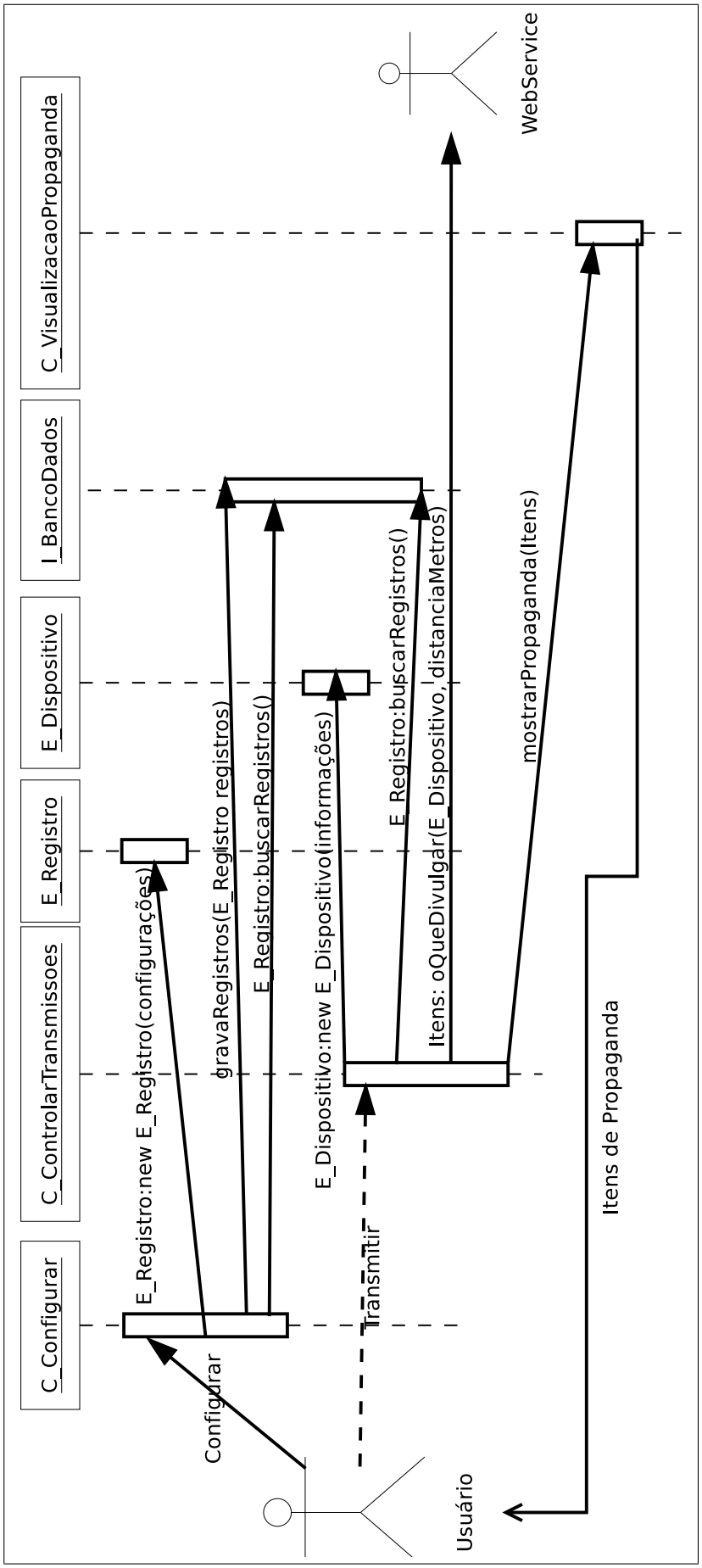


Figura 3.8: Diagrama de Sequência: Mídia.

O Diagrama de Entidade Relacionamento do Cliente é representado na figura 3.9, uma única tabela contendo os registros de interesses do usuário é necessária para o **Midia**.

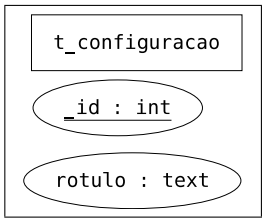


Figura 3.9: Diagrama Entidade Relacionamento: Mídia.

3.2.1 Requisitos funcionais

O sistema deverá ser capaz de:

1. Cadastrar (incluir, atualizar remover) as configurações de oferta de seu usuário
2. Transmitir tais configurações para o servidor
3. Do servidor, receber informações que deverão ser expostas
4. Expor tais informações

3.2.2 Requisitos não funcionais

1. Será desenvolvido para a plataforma Android
2. O armazenamento local das configurações das ofertas utilizará o banco de dados Sqlite
3. A transmissão das informações ocorrerá de maneira cíclica e automatizada
4. A transmissão será implementada de modo a utilizar os WebServices disponíveis pelo servidor.
5. A exposição contará com uma conexão HDMI entre o celular e um dispositivo áudio visual
6. Notificações, com os nomes dos itens procurados, serão utilizados para a exposição
7. As devidas permissões de acesso à Internet e ao GPS deverão ser concedidas para que o *software* seja instalado

3.3 Servidor

A figura 3.10 descreve seu Caso de Uso. Um **Cliente** aciona o método `procuro`, iniciando a ação que grava um dispositivo. Para tal utiliza o acesso ao SGBD. Um *software* **Midia** aciona o método `oQueDivulgar` que inicia a procura de itens para divulgação, comparando os itens de interesse dos consumidores finais com os que estão sendo ofertados pelos comerciantes. A procura de interesses faz uso do acesso ao banco e é seguida pelo cálculo de distância entre ambos e, por último, pela priorização das respostas.

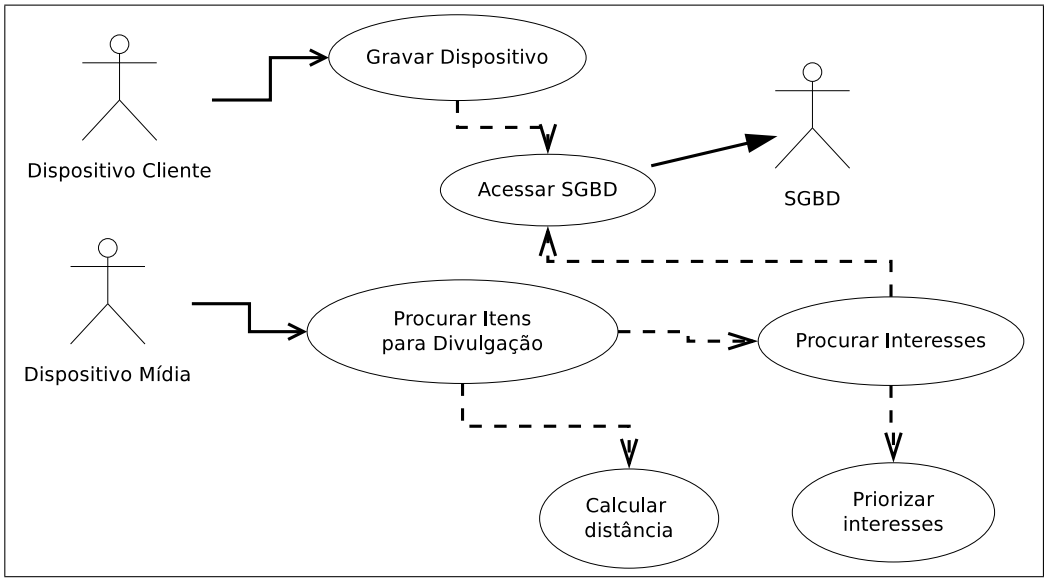


Figura 3.10: Diagrama de Casos de Uso: Servidor.

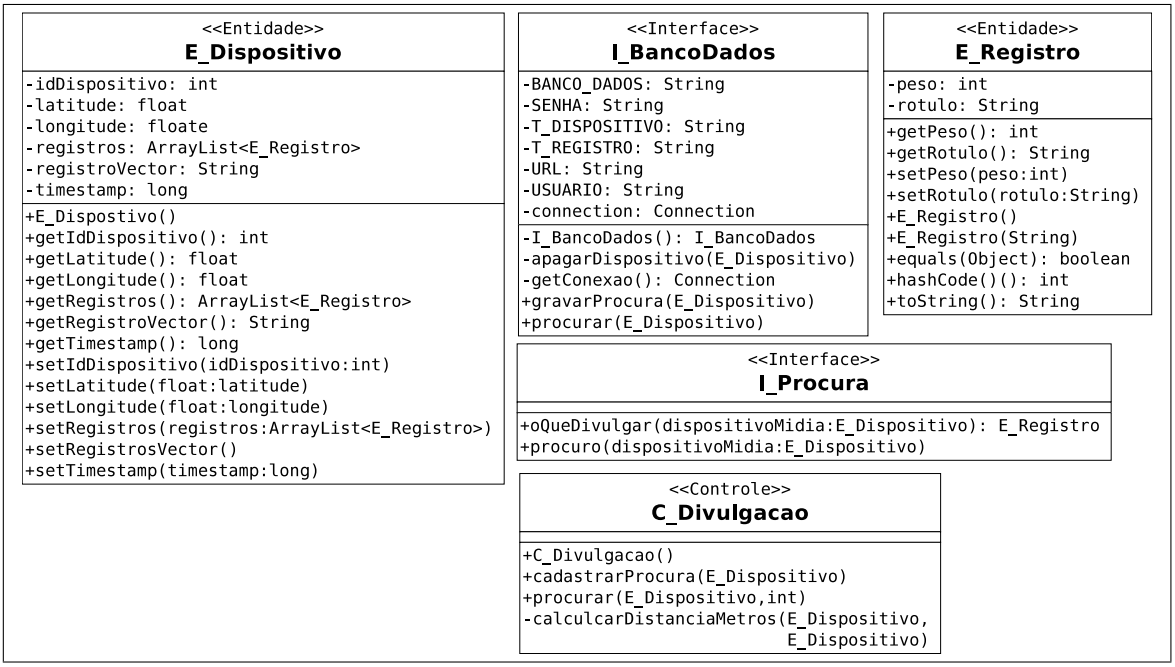


Figura 3.11: Diagrama de Classes: Servidor.

O Diagrama de Classes, representados pela figura 3.11, é descrito a seguir:

`I_BancoDados` Classe que faz a comunicação com o SGBD MySql no sistema

`E_Dispositivo` Contém as informações necessárias para representar um dispositivo, contém uma lista de `E_Registro`

`E_Registro` Contém as informações que definem um interesse do usuário. Faz parte de um `E_Dispositivo`

`I_Procura` É a interface do Web Service, acionado pelos *softwares* **Cliente** e **Midia**. É o componente que recebe e envia as mensagens para os clientes. Ver listagens do código Java em 4.1, página 43 e do documento XML que descreve seu WSDL em 4.2, página 44

`C_Divulgacao` Controla o cadastro das ofertas e procuras, faz o cálculo de distâncias e prioriza as respostas. Ver listagem 4.3, página 47

A figura 3.12 representa a relação de um para muitos entre os registros e os dispositivos. Únicas tabelas necessárias para a solução.

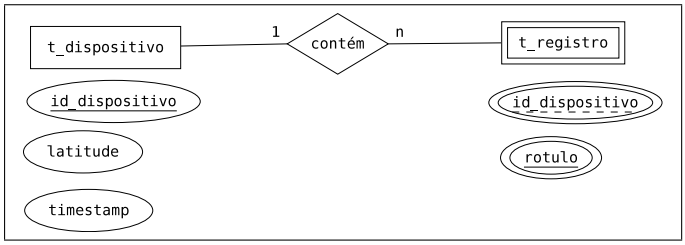


Figura 3.12: Diagrama Entidade Relacionamento: Servidor.

O Diagrama de Sequência, descrito a seguir, pode ser visualizado na figura 3.13.

1. O método `procuro` é acionado e então, um objeto da classe `C_Divulgacao` instancia os objetos `E_Dispositivo` e `E_Registro` para que, utilizando de `I_BancoDados` faça a persistência da informação de procura por parte dos clientes.

2. O método `oQueDivulgar` é acionado. Um objeto `C_Divulgacao` é instanciado, que instancia os `E_Dispositivo` e `E_Registro`. Logo em seguida utiliza-se de `I_BancoDados` para encontrar dispositivos que tenham registros de interesse compatíveis. Com o primeiro resultado, um cálculo de distância é efetuado e o resultado ordenado de maneira a privilegiar os mais buscados por clientes. Então o método retorna uma mensagem para o dispositivo que o chamou

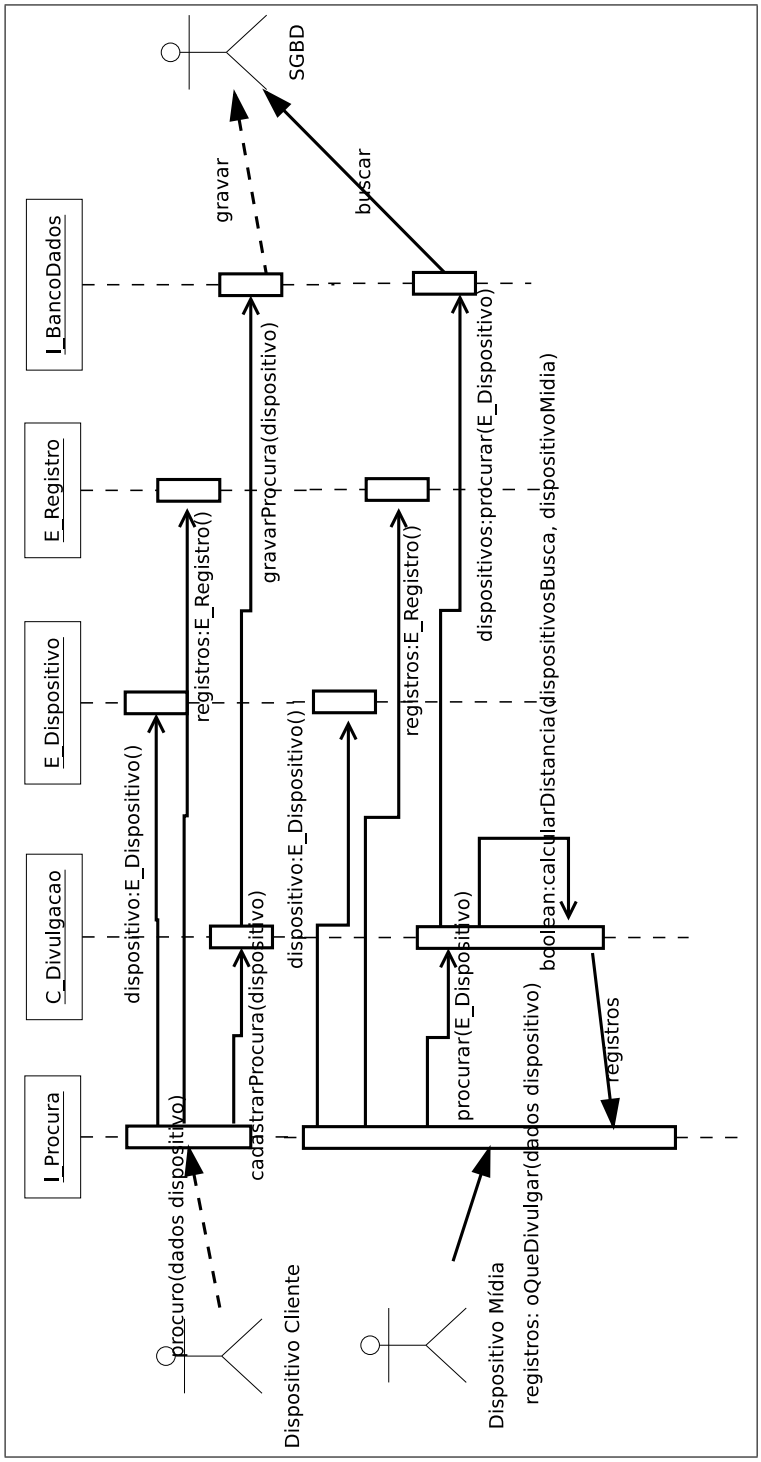


Figura 3.13: Diagrama de Sequência: Servidor.

3.3.1 Requisitos funcionais

O sistema deverá ser capaz de:

1. Receber as transmissões dos clientes e armazenar suas configurações
2. Receber as transmissões dos agentes de mídia, comparar oferta e procura para então, enviar as que são de interesse dos clientes, priorizando o maior interesse
3. Deverá levar em consideração a proximidade dos agentes de mídia e usuários clientes

3.3.2 Requisitos não funcionais

1. Será desenvolvido com a linguagem de programação Java, utilizará os servidores Apache, Tomcat e MySql
2. Fará uso, em suas transmissões, de WebServices e mensagem SOAP.

3.4 Utilização

Os dois sistemas baseados no Android dependem dos serviços oferecidos pelo **Servidor**. Este último depende dos **Servidor**es Apache, Tomcat e MySql para ser implementado. Assumindo que todos eles estão funcionando adequadamente ainda é preciso executar as tarefas relacionadas à configuração do banco de dados tais como: criação do próprio banco, das tabelas e do usuário com as devidas permissões de acesso. Com todo o ambiente pronto, basta disponibilizar o pacote referente ao **Servidor**, Servidor.ear.

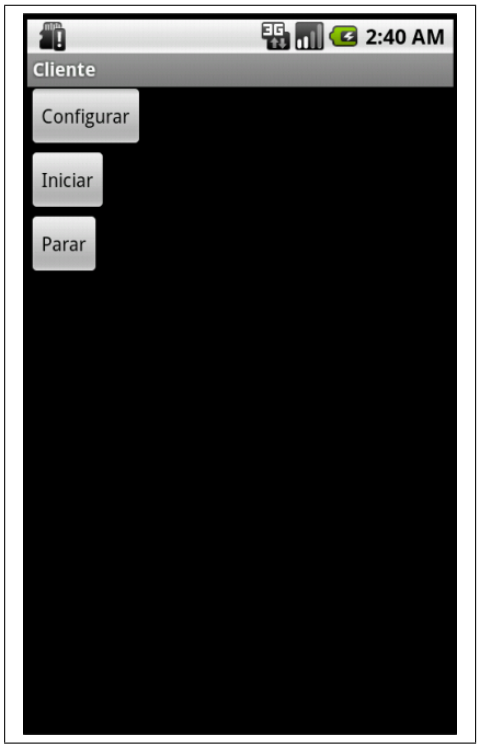
O *software* SoapUi-4.0.0 foi utilizado para enviar chamadas de teste aos dois serviços web disponibilizados pelo **Servidor**. Garantindo-se assim, seu funcionamento.

O próximo passo é o de instalar o *software* **Cliente** num dispositivo ou emulador, para tal basta copiar o pacote Cliente.apk utilizando um gerenciador de arquivos e executar os procedimentos padrão de instalação. O sistema depende das permissões de utilização da Internet e do GPS para seu correto funcionamento.

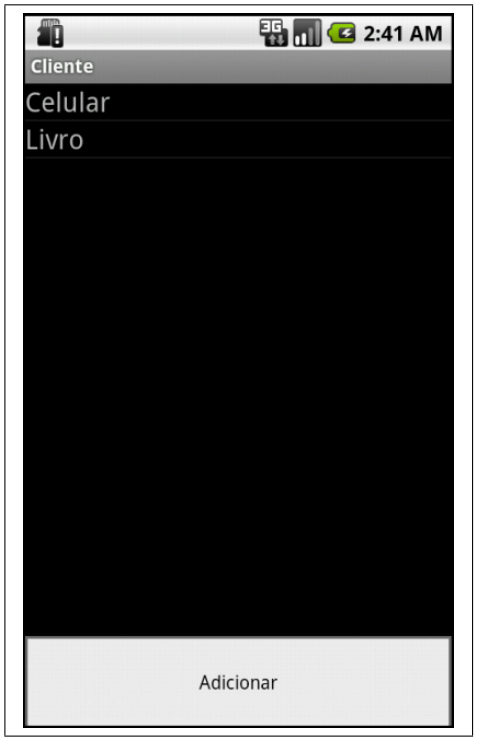
Após sua instalação a primeira ação sugerida é a de configurar sua lista de interesses. Acionando o ícone da aplicação e, em seguida, o menu de Configuração uma tela de cadastro será visualizada. Conforme figura 3.14(b)

Para inserir registros, basta acionar o menu do Android, figura 3.14(b), para corrigir algum que já exista um *click* curto levará a respectiva tela, figura 3.14(d) e finalmente, para exclusão, um *click* longo pedirá uma confirmação e, em caso positivo, o registro será apagado, figura 3.14(c). Todo o controle de banco de dados é feito pelas bibliotecas Sqlite que, em tempo de execução, re criam a tabela caso a versão do banco mude.

O menu voltar mostrará a tela inicial novamente e ali, figura 3.14(a), além do botão de configuração, existem outros dois: Iniciar e Parar.

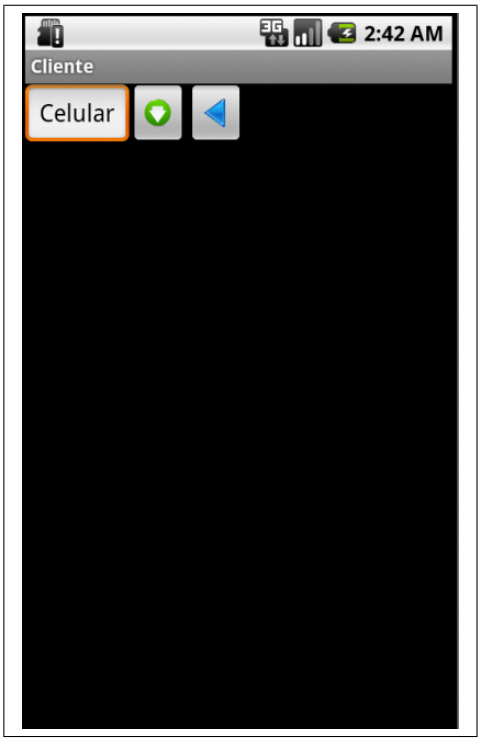


(a) Tela inicial



(b) Inclusão

(c) Exclusão



(d) Edição

Acionar o botão Iniciar aciona um serviço em segundo plano, este serviço verifica se houve alguma alteração na localidade do dispositivo móvel utilizando as bibliotecas do Google Maps e, em caso afirmativo, coleta os registros gravados no banco de dados local e, junto às informações de latitude e longitude, acionam o serviço `procuro(E_Dispositivo)` do **Servidor**.

Esta ação se repete continuamente enquanto houver mudança de localidade do dispositivo, ou até que o usuário acione o botão Parar, que causa o desligamento do serviço em segundo plano no aparelho responsável pela atualização do GPS.

O **Servidor**, por sua vez, gravará as informações referentes à localidade, registros de interesses e identificação do dispositivo em seu banco de dados. Mantendo sempre o registro da última posição gravada pelo **Cliente**.

O procedimento para instalação do sistema no dispositivo de **Mídia** segue os mesmos itens do que o de **Cliente**.

Apesar das telas serem semelhantes, as informações aqui gravadas diferem em sentido. Aqui, ao registrar um produto, o usuário estará registrando um item em oferta, ou ainda, um item que se deseja fazer publicidade. Este cadastro segue os mesmos detalhes que os do **Cliente**.

Os mesmos botões: Iniciar e Parar estão disponíveis. Com a diferença que o dispositivo de **Mídia** não acionará o serviço web caso sua localidade mude. No momento em que acionado o sistema GPS informará qual sua posição, depois disto, a cada quantidade pré determinada de tempo serão reunidos os registros referentes à publicidade e, junto aos dados do dispositivo, será acionado o serviço `oQueDivulgar(E_Dispositivo)` que retornará, se for o caso, os produtos de interesse pelos usuários do sistema **Cliente** próximos.

O **Servidor**, ao receber tal requisição, fará uma busca por dispositivos móveis **Cliente** que, nos últimos minutos, passaram por ali e que têm interesse naquele item registrado pelo dispositivo **Mídia**. A resposta é uma lista, ordenada de modo a refletir o maior interesse pelos que ali estão, com os nomes dos produtos.

De posse desta lista, o sistema **Mídia** disponibiliza informações sobre aqueles produtos à tempo dos interessados notarem.

O botão Parar termina a execução do serviço em segundo plano.

Para os testes foram utilizados o emulador Android, o já citado SoapUi, além do ambiente de desenvolvimento Eclipse para Android e aplicações WEB. Para simular a movimentação do dispositivo WEB foram inseridos comandos com posições geográficas no emulador, via interface própria do ambiente de desenvolvimento. O **Servidor** foi implementado na mesma máquina onde estavam instalados os *softwares* **Cliente** e **Mídia** e trocavam mensagens através de seu IP físico.

Do ponto de vista do cliente, uma vez configurado suas preferências, sua única preocupação está em deixar, ou não, o serviço de atualização de localidade funcionando. Todo o resto é tratado com transparência pelo *software*, que não deixa de ser executado caso o

aparelho celular precise acionar outra aplicação.

Também é transparente para o sistema **Mídia**, com o detalhe que a solução propõe um dispositivo com saída HDMI, para que as informação sejam de fato, vistas pelos usuários **Clientes**.

3.5 Ferramentas Utilizadas

No decorrer da confecção do trabalho, foram utilizadas as seguintes ferramentas:

Desenvolvimento Java, Google Maps API, Android SDK, Axis2, KSOAP2 e Eclipse

Servidores Apache Tomcat, Axis2 e MySql

Testes Android Virtual Devices, Sony Ericson XPERIA e SoapUI

CAPÍTULO 4

CONCLUSÕES

O protótipo apresentado mostrou-se capaz de transmitir as mensagens entre os dispositivos e o servidor, tornando possível assim, o Marketing Dinâmico. Um cenário onde a publicidade se adeque aos desejos dos seus usuários de forma transparente e automática.

Dos diversos sistemas operacionais para dispositivos móveis existentes, o Android se destaca pela facilidade de desenvolvimento, criação de interfaces ricas e comunidade crescente de desenvolvedores, que dá força para a tecnologia.

O modelo da solução centraliza todas as mensagens dos dispositivos em um único servidor. Uma vantagem dele é a possibilidade de armazenar os dados dos **Clientes** e de **Mídias** para utilização futura. A desvantagem do modelo é a necessidade da infra estrutura, ou seja, do servidor.

A utilização de Web Services acelerou e simplificou o processo de desenvolvimento e trouxe a possibilidade de ampliação da solução para outras plataformas e dispositivos, sem alteração da implementação já concluída.

4.1 Trabalhos Futuros

Para uma adoção mais massiva dos sistemas **Cliente** e **Mídia**, por parte dos consumidores finais, a implementação em outras plataformas deve ser estudada.

Uma proposta de trabalho futuro seria estudar uma solução que não dependa da centralização em um servidor, tornando os dispositivos capazes de se encontrarem e trocarem mensagens sozinhos.

A proposta é dependente do GPS para a obtenção da localização de seus usuários. Para os casos onde esta tecnologia não esteja presente uma alternativa baseada em triangulação de antenas utilizadas na telefonia celular pode ser estudada.

Apesar de possuir uma estrutura de banco de dados simples, ela é ponto sensível na proposta apresentada. A quantidade de requisições feitas por um dispositivo **Cliente** ao servidor poderá exigir soluções mais complexas para manter a disponibilidade. Ou o dispositivo **Mídia** só conseguirá a informação muito tempo depois que o usuário **Cliente** tiver passado.

O mesmo vale para o servidor web onde os serviços estarão disponibilizados, num ambiente produtivo, a escalabilidade e disponibilidade devem ser tratados com cuidado.

A base de informações gerada a partir da utilização dos sistemas pode ser de valia para a prospecção de padrões de consumo. No momento, a informação lá é efêmera e atende apenas a um objetivo claro: direcionar a publicidade naquele momento. Porém, como

proposta de trabalho futuro e melhoria, um aumento na amplitude desta base, tornando-a capaz de armazenar informações nos últimos dias, registrando o interesse de usuários por região, faixa econômica, para tornar possível a criação de aplicações que tratem estes dados e tornem as mensagens e produtos de marketing mais assertivos e lucrativos.

BIBLIOGRAFIA

[1] Open Handset Alliance. Android. http://www.openhandsetalliance.com/android_overview.html, 2011.

[2] bada. bada. http://www.bada.com/, 2011.

[3] Tania Teixeira BBC. Meet Marty Cooper - the inventor of the mobile phone. http://news.bbc.co.uk/2/hi/programmes/click_online/8639590.stm, 2010.

[4] Scott W. Campbell e Yong Jin Park. Social implications of mobile telephony: The rise of personal communication society. *Sociology Compass*, 2(2):371–387, 2008.

[5] World Wide Web Consortium. Web Services Architecture. http://www.w3c.org/TR/2004/NOTE-ws-arch-20040211/, 2004.

[6] World Wide Web Consortium. Security. http://www.w3.org/standards/xml/security, 2010.

[7] World Wide Web Consortium. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2011.

[8] World Wide Web Consortium. WSDL and UDDI. http://www.w3schools.com/wsdl/wsdl_uddi.asp, 2011.

[9] Cay S. Horstmann, Gary Cornell. *CoreJava 2 Volume I - Fundamentos*. Makron Books, 2001.

[10] Comitê Gestor da Internet do Brasil. Pesquisa sobre o uso das tecnologias da informação e da comunicação no Brasil 2005-2009. http://www.cgi.br, 2009.

[11] Agência Nacional de Telecomunicações. Números do Setor. http://www.anatel.gov.br/Portal/exibirPortalNivelDois.do?codItemCanal=982&nomeVisao=Cidad%E3o&nomeCanal=Informa%E7%F5es%20e%20consultas&nomeItemCanal=N%FAmeros%20do%20Setor, 2010.

[12] LiMo Foundation. LiMo Foundation. http://www.limofoundation.org/, 2011.

[13] The Apache Software Foundation. Welcome to Apache Axis2Java. http://axis.apache.org/axis2/java/core/index.html, 2011.

[14] Elliot D. Kaplan, Christopher J. Hegarty. *Understanding GPS Principals and applications*. Artech House, 2006.

[15] Google Inc. Android. `http://www.android.com/`, 2011.

[16] Google Inc. Android Developers. `http://developers.android.com/`, 2011.

[17] Google Inc. Android Market. `https://market.android.com/`, 2011.

[18] N. Itaya, S. Kasahara. Dynamic parameter adjustment for available-bandwidth estimation of tcp in wired-wireless networks. *Computer Communications*, 27:225–234, 2004.

[19] Frank H. P. Fitzek, Marcos D. Katz. *Cognitive Wireless Networks. Concepts, Methodologies and Visions Inspiring the Age of Enlightenment of Wireless Communications*. Springer Netherlands, 2007.

[20] James Keogh. *J2ME: The Complete Reference*. McGraw-Hill/Osborne, 2003.

[21] James Snell, Doug Tidwell, Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilly, 2002.

[22] Ricardo R. Lecheta. *Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK*. Novatec Editora, 2010.

[23] MeeGo. MeeGo. `http://meego.com/`, 2011.

[24] Microsoft. App Hub. `http://create.msdn.com/en-US/`, 2011.

[25] Nokia. Nokia Developer. `http://www.developer.nokia.com`, 2011.

[26] Oracle. About Java ME. `http://www.oracle.com/technetwork/java/javame/about-java-me-395899.html`, 2011.

[27] G. W. Ozanich, C. Hsu, H. W. Park. 3-G Wireless Auctions as an Economic Barrier to Entry: The Western Europe an Experience. *Telematics and Informatics*, 21:225–234, 2004.

[28] Adriana Souza e Silva. Cell phones and places: The use of mobile technologies in brazil. Max Barlow, Wolf Tietze, Paul Claval, Yehuda Gradus, Risto Laulajainen, Sam Ock Park, Herman Wusten, e Harvey J. Miller, editors, *Societies and Cities in the Age of Instant Access*, volume 88 of *GeoJournal Library*, páginas 295–310. Springer Netherlands, 2007. 10.1007/1-4020-5427-0_19.

[29] Andrew S. Tanenbaum. *Redes de Computadores*. Editora Campus, 2003.

[30] Andrew S. Tanenbaum. *Sistemas Operacionais Modernos, Segunda Edição*. Prentice Hall, 2003.

[31] Inteligência em Comunicações Teleco. Estatísticas de Celulares no Brasil. http://www.teleco.com.br/ncel.asp, 2011.

[32] International Telecommunication Union. Itu global standard for international mobile telecommunicatios IMT-Advanced. http://www.itu.int/ITU-R/index.asp?category=information\&rlink=imt-advanced, 2010.

[33] International Telecommunication Union. The World in 2010. http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf, 2010.

[34] International Telecommunication Union. About mobile technology and IMT-2000. http://www.itu.int/osg/spu/imt-2000/technology.html, 2011.

[35] Shu-Lin Wang e Chun-Yi Wu. Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Systems with Applications*, 38(9):10831 – 10838, 2011.

[36] Mark Weiser. Ubiquitous Computing. http://www.ubiq.com/hypertext/weiser/UbiHome.html, 1996.

APÊNDICE A - CÓDIGOS FONTE

```
1  package marketingDinamicoServidor;

   import java.util.ArrayList;
   import java.util.Collections;
   import java.util.Iterator;
6
   public class I_Procura {

    public void procuro(int idDispositivo, float latitude, float longitude,
      long timestamp, String registrosVector) {
11   E_Dispositivo d = new E_Dispositivo();
     d.setIdDispositivo(idDispositivo);
     d.setLatitude(latitude);
     d.setLongitude(longitude);
     d.setTimestamp(timestamp);
16   d.setRegistrosVector(registrosVector);
     new C_Divulgacao().cadastrarProcura(d);
    }

    public String oQueDivulgar(int idDispositivo, float latitude,
21    float longitude, long timestamp, String registrosVector,
      int distanciaMetros) {
     String retorno = "";
     E_Dispositivo dispositivoMidia = new E_Dispositivo();
     dispositivoMidia.setIdDispositivo(idDispositivo);
26   dispositivoMidia.setLatitude(latitude);
     dispositivoMidia.setLongitude(longitude);
     dispositivoMidia.setTimestamp(timestamp);
     dispositivoMidia.setRegistrosVector(registrosVector);
     retorno = new C_Divulgacao().procurar(
31     dispositivoMidia, distanciaMetros);
     return retorno;
    }
   }
```

Código Fonte 4.1: marketingDinamicoServidor.I_Procura.java

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns1="http://org.apache.axis2/xsd" xmlns:ns="http://marketingDinamicoServidor"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    targetNamespace="http://marketingDinamicoServidor">
    <wsdl:documentation>
  Servicos para Marketing Dinamico
 </wsdl:documentation>
    <wsdl:types>
        <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
            targetNamespace="http://marketingDinamicoServidor">
            <xs:element name="procuro">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="idDispositivo" type="xs:int"/>
                        <xs:element minOccurs="0" name="latitude" type="xs:float"/>
                        <xs:element minOccurs="0" name="longitude" type="xs:float"/>
                        <xs:element minOccurs="0" name="timestamp" type="xs:long"/>
                        <xs:element minOccurs="0" name="registrosVector" nillable="true"
                            type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="oQueDivulgar">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="idDispositivo" type="xs:int"/>
                        <xs:element minOccurs="0" name="latitude" type="xs:float"/>
                        <xs:element minOccurs="0" name="longitude" type="xs:float"/>
                        <xs:element minOccurs="0" name="timestamp" type="xs:long"/>
                        <xs:element minOccurs="0" name="registrosVector" nillable="true"
                            type="xs:string"/>
                        <xs:element minOccurs="0" name="distanciaMetros" type="xs:int"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="oQueDivulgarResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true"
                            type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="oQueDivulgarRequest">
        <wsdl:part name="parameters" element="ns:oQueDivulgar"/>
    </wsdl:message>
    <wsdl:message name="oQueDivulgarResponse">
        <wsdl:part name="parameters" element="ns:oQueDivulgarResponse"/>
    </wsdl:message>
    <wsdl:message name="procuroRequest">
```

```
        <wsdl:part name="parameters" element="ns:procuro"/>
    </wsdl:message>
    <wsdl:portType name="I_ProcuraPortType">
        <wsdl:operation name="oQueDivulgar">
            <wsdl:input message="ns:oQueDivulgarRequest" wsaw:Action="urn:oQueDivulgar"/>
            <wsdl:output message="ns:oQueDivulgarResponse"
                wsaw:Action="urn:oQueDivulgarResponse"/>
        </wsdl:operation>
        <wsdl:operation name="procuro">
            <wsdl:input message="ns:procuroRequest" wsaw:Action="urn:procuro"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="I_ProcuraSoap11Binding" type="ns:I_ProcuraPortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <wsdl:operation name="oQueDivulgar">
            <soap:operation soapAction="urn:oQueDivulgar" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="procuro">
            <soap:operation soapAction="urn:procuro" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="I_ProcuraSoap12Binding" type="ns:I_ProcuraPortType">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
            style="document"/>
        <wsdl:operation name="oQueDivulgar">
            <soap12:operation soapAction="urn:oQueDivulgar" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="procuro">
            <soap12:operation soapAction="urn:procuro" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="I_ProcuraHttpBinding" type="ns:I_ProcuraPortType">
        <http:binding verb="POST"/>
        <wsdl:operation name="oQueDivulgar">
            <http:operation location="oQueDivulgar"/>
            <wsdl:input>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type="text/xml" part="parameters"/>
            </wsdl:output>
```

```
        </wsdl:operation>
        <wsdl:operation name="procuro">
106             <http:operation location="procuro"/>
                <wsdl:input>
                    <mime:content type="text/xml" part="parameters"/>
                </wsdl:input>
        </wsdl:operation>
111     </wsdl:binding>
        <wsdl:service name="I_Procura">
            <wsdl:port name="I_ProcuraHttpSoap11Endpoint"
                binding="ns:I_ProcuraSoap11Binding">
                <soap:address location="http://localhost:8080/Servidor/services/
    I_Procura.I_ProcuraHttpSoap11Endpoint/"/>
116         </wsdl:port>
            <wsdl:port name="I_ProcuraHttpSoap12Endpoint"
                binding="ns:I_ProcuraSoap12Binding">
                <soap12:address location="http://localhost:8080/Servidor/services/
    I_Procura.I_ProcuraHttpSoap12Endpoint/"/>
            </wsdl:port>
121         <wsdl:port name="I_ProcuraHttpEndpoint" binding="ns:I_ProcuraHttpBinding">
                <http:address location="http://localhost:8080/Servidor/services/
    I_Procura.I_ProcuraHttpEndpoint/"/>
            </wsdl:port>
        </wsdl:service>
126 </wsdl:definitions>
```

Código Fonte 4.2: I_Procura.xml

```
package marketingDinamicoServidor;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class C_Divulgacao {

 public C_Divulgacao() {
 }

 public void cadastrarProcura(E_Dispositivo dispositivoProcura) {
  if (dispositivoProcura != null) {
   I_BancoDados.getInstance().gravarProcura(dispositivoProcura);
  }
 }

 public ArrayList<E_Registro> procurar(E_Dispositivo dispositivoMidia,
   int distanciaMetros) {
  if ((dispositivoMidia == null) || (distanciaMetros < 1)) {
   return null;
  }
  ArrayList<E_Registro> registros = new ArrayList<E_Registro>();
  for (E_Dispositivo d : I_BancoDados.getInstance().procurar(
    dispositivoMidia)) {
   if (this.calcularDistanciaMetros(d, dispositivoMidia) < distanciaMetros) {
    registros.addAll(d.getRegistros());
   }
  }

  ArrayList<E_Registro> registrosComPeso = new ArrayList<E_Registro>();
  for (E_Registro r : registros) {
   if (registrosComPeso.contains(r)) {
    registrosComPeso.get(registrosComPeso.indexOf(r)).setPeso(
      1 + registrosComPeso.get(registrosComPeso.indexOf(r))
        .getPeso());
   } else {
    r.setPeso(1);
    registrosComPeso.add(r);
   }
  }

  Collections.sort(registrosComPeso, new Comparator<E_Registro>() {
   public int compare(E_Registro r1, E_Registro r2) {
    return r1.getPeso() < r2.getPeso() ? +1 : (r1.getPeso() > r2
      .getPeso() ? -1 : 0);
   }
  });

  return registrosComPeso;
 }

 private float calcularDistanciaMetros(E_Dispositivo dispositivoCliente,
   E_Dispositivo dispositivoMidia) {
  double raioTerra = 3958.75;
  double latitude = Math.toRadians(dispositivoCliente.getLatitude()
    - dispositivoMidia.getLatitude());
  double longitude = Math.toRadians(dispositivoCliente.getLongitude()
```

```
59      - dispositivoMidia.getLongitude());
        double d1 = Math.sin(latitude / 2) * Math.sin(latitude / 2)
          + Math.cos(Math.toRadians(dispositivoCliente.getLatitude()))
          * Math.cos(Math.toRadians(dispositivoMidia.getLatitude()))
          * Math.sin(longitude / 2) * Math.sin(longitude / 2);
64      double d2 = 2 * Math.atan2(Math.sqrt(d1), Math.sqrt(1 - d1));
        double distanciaMetros = raioTerra * d2 * 1609;
        return new Float(distanciaMetros).floatValue();
      }
    }
```

Código Fonte 4.3: marketingDinamicoServidor.C_Divulgacao.java

```
package monografia.marketingDinamicoMidia;

import java.io.IOException;
import java.util.Date;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.Marshal;
import org.ksoap2.serialization.PropertyInfo;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
...
import android.widget.Toast;

public class C_ControlarTransmissoes extends Service implements
  LocationListener {
 private static final String URL = "http://localhost:8080/
_____Servidor/services/I_Procura";
 private Location location;
 private I_BancoDados ibd = new I_BancoDados(this);
 private static int idDispositivo;
 private Looper looper;
 private TransmitirHandler transmitirHandler;

 ...

 private void transmitir(E_Dispositivo dispositivoCliente)
  throws IOException, XmlPullParserException {

  SoapObject soap = new SoapObject("http://marketingDinamicoServidor",
    "oQueDivulgar");
  soap
    .addProperty("idDispositivo", dispositivoCliente
      .getIdDispositivo());
  soap.addProperty("latitude", dispositivoCliente.getLatitude());
  soap.addProperty("longitude", dispositivoCliente.getLongitude());
  soap.addProperty("timestamp", dispositivoCliente.getTimestamp());
  soap.addProperty("registrosVector", dispositivoCliente
    .getRegistrosVector());
  soap.addProperty("distanciaMetros", 10);
  SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
    SoapEnvelope.VER11);
  envelope.setOutputSoapObject(soap);
  C_MarshalFloat cmf = new C_MarshalFloat();
  cmf.register(envelope);
  Log.i("Monografia", "Chamando WebService " + URL);
  HttpTransportSE httpTransportSE = new viewHttpTransportSE(URL);
  httpTransportSE.call("http://marketingDinamicoServidor/oQueDivulgar",
    envelope);
  Object resultado = envelope.getResponse();
  if (resultado != null) {
   Toast.makeText(this, resultado.toString(), Toast.LENGTH_LONG)
     .show();
   }
  }
 }
}
```

```
class viewHttpTransportSE extends HttpTransportSE {
 public viewHttpTransportSE(String s) {
  super(s);
 }

 @Override
 public void call(String s, SoapEnvelope soapenvelope) throws IOException,
   XmlPullParserException {
  byte bytes[] = createRequestData(soapenvelope);
  String envelope = new String(bytes);
  Log.i("Monografia", "Envelope: " + envelope);
  super.call(s, soapenvelope);
 }
}

class C_MarshalFloat implements Marshal {

 @Override
 public Object readInstance(XmlPullParser arg0, String arg1, String arg2,
   PropertyInfo arg3) throws IOException, XmlPullParserException {
  return Float.parseFloat(arg0.nextText());
 }

 @Override
 public void register(SoapSerializationEnvelope arg0) {
  arg0.addMapping(arg0.xsd, "float", Float.class, this);
 }

 @Override
 public void writeInstance(XmlSerializer arg0, Object arg1)
   throws IOException {
  arg0.text(arg1.toString());
 }
}
```

Código Fonte 4.4: marketingDinamicoMidia.C_ControlarTransmissoes.java

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="Some-URI">
   <symbol>DIS</symbol>
  </m:GetLastTradePrice>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Código Fonte 4.5: ExemploMensagem SOAP

```
package monografia.marketingDinamicoCliente;

import java.io.IOException;
import java.util.Date;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.Marshal;
import org.ksoap2.serialization.PropertyInfo;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlSerializer;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;

public class C_ControlarTransmissoes extends Service implements
  LocationListener, Runnable {

 private static final String URL =
     "http://201.86.13.144:8080/Servidor/services/I_Procura";
 private Location location;
 private I_BancoDados ibd = new I_BancoDados(this);
 private static int idDispositivo;

 @Override
 public IBinder onBind(Intent arg0) {
  return null;
 }

 @Override
 public void onCreate() {
  idDispositivo = (int) (1 + Math.random() * 10000);
 }

 @Override
 public void onDestroy() {
  super.onDestroy();
  this.getLocationManager().removeUpdates(this);
 }

 @Override
 public void onStart(Intent intent, int startId) {
  super.onStart(intent, startId);
  this.getLocationManager().requestLocationUpdates(
    LocationManager.GPS_PROVIDER, 0, 0, this);

 }
```

```
    @Override
    public void onLocationChanged(Location arg0) {
60   Log.i("Monografia", "latitude: " + arg0.getLatitude() + " longitude: "
       + arg0.getLongitude());
     if (arg0 != null) {
      this.location = arg0;
      new Thread(this).start();
65   }
    }

    @Override
    public void onProviderDisabled(String arg0) {
70  }

    @Override
    public void onProviderEnabled(String arg0) {
    }
75
    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
    }

80  private LocationManager getLocationManager() {
     LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
     return lm;
    }

85  @Override
    public void run() {
     E_Dispositivo dispositivo;
     try {
      dispositivo = new E_Dispositivo();
90    dispositivo.setIdDispositivo(idDispositivo);
      dispositivo.setLatitude((float) this.getLocation().getLatitude());
      dispositivo.setLongitude((float) this.getLocation().getLongitude());
      dispositivo.setTimestamp(new Date().getTime());
      ibd.abrirBD();
95    dispositivo.setRegistros(ibd.getTodosRegistros());
      dispositivo.setRegistrosVector();
      this.transmitir(dispositivo);
     } catch (Exception e) {
      Log.e("Monografia", e.getMessage(), e);
100  } finally {

     }
    }

105 private Location getLocation() {
     return this.location;
    }

    private void transmitir(E_Dispositivo dispositivoCliente)
110   throws IOException, XmlPullParserException {


     SoapObject soap = new SoapObject("http://marketingDinamicoServidor",
       "procuro");
115  soap.addProperty("idDispositivo", dispositivoCliente.getIdDispositivo());
     soap.addProperty("latitude",dispositivoCliente.getLatitude());
```

```
    soap.addProperty("longitude",dispositivoCliente.getLongitude());
    soap.addProperty("timestamp",dispositivoCliente.getTimestamp());
    soap.addProperty("registrosVector",dispositivoCliente.getRegistrosVector());
120
    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
      SoapEnvelope.VER11);

    envelope.setOutputSoapObject(soap);
125 C_MarshalFloat cmf = new C_MarshalFloat();
    cmf.register(envelope);
    Log.i("Monografia", "Chamando WebService " + URL);
    HttpTransportSE httpTransportSE = new viewHttpTransportSE(URL);
    httpTransportSE.call("http://marketingDinamicoServidor/procuro", envelope);
130 Log.i("Monografia", envelope.getResponse().toString());
   }
  }

  class viewHttpTransportSE extends HttpTransportSE {
135 public viewHttpTransportSE(String s) {
    super(s);
   }

   @Override
140 public void call(String s, SoapEnvelope soapenvelope) throws IOException,
     XmlPullParserException {
    byte bytes[] = createRequestData(soapenvelope);
    String envelope = new String(bytes);
    Log.i("Monografia", "Envelope: " + envelope);
145 super.call(s, soapenvelope);
   }
  }

  class C_MarshalFloat implements Marshal {
150
   @Override
   public Object readInstance(XmlPullParser arg0, String arg1, String arg2,
     PropertyInfo arg3) throws IOException, XmlPullParserException {
    return Float.parseFloat(arg0.nextText());
155 }

   @Override
   public void register(SoapSerializationEnvelope arg0) {
    arg0.addMapping(arg0.xsd, "float", Float.class, this);
160 }

   @Override
   public void writeInstance(XmlSerializer arg0, Object arg1)
     throws IOException {
165 arg0.text(arg1.toString());
   }
  }
```

Código Fonte 4.6: marketingDinamico.Cliente.C_ControlarTransmissoes.java