UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME DE MORAES UZEJKA

MODELANDO UM CLIENTE VOIP INTELIGENTE PARA A PLATAFORMA ANDROID

Trabalho de Graduação

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, julho de 2011.

GUILHERME DE MORAES UZEJKA

MODELANDO UM CLIENTE VOIP INTELIGENTE PARA A PLATAFORMA ANDROID

Trabalho de conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação da Universidade Federal do Rio Grande do Sul

Orientação: Prof. Dr. Cláudio Fernando Resin Geyer

Porto Alegre, julho de 2011.

AGRADECIMENTOS

Agradeço à Universidade Federal do Rio Grande do Sul, pelo ensino e oportunidade.

Agradeço ao Instituto de Informática, pelo ambiente propício e pelo pessoal altamente qualificado o qual me cercou no período de desenvolvimento deste trabalho.

Agradeço ao Professor Dr. Cláudio Resin Geyer pela orientação e auxílio.

Agradeço ao colega Marcelo Bublitz Anton, pela parceria e auxílio na construção deste projeto.

Agradeço à minha família, pelo apoio e compreensão.

Agradeço à minha namorada Aline Luvielmo de Araújo, pelo apoio e incentivo.

“O melhor lugar para ter sucesso

é onde você está, com aquilo que você tem.”

(Charles Schwab)

RESUMO

Em um momento de popularização de dispositivos móveis e aumento da oferta para conexões sem fio e de alta velocidade com a Internet foi desenvolvido este projeto. Ele visa oferecer aos usuários uma solução prática e simples para economizar dinheiro com ligações telefônicas, fazendo escolhas entre chamadas VoIP e chamadas pela rede convencional de telefonia celular. Estas escolhas são realizadas de maneira transparente ao usuário, fazendo com que a experiência com o software não seja comprometida. A interface do aplicativo é um simples discador que recebe como entrada o número de destino a ser ligado. A proposta deste projeto foi realizar um estudo com o intuito de disponibilizar esta solução para a plataforma Android, escolhida devido a sua modularidade e simplicidade para substituir componentes do sistema, como é o caso do discador. As soluções deste tipo já existentes não são pró-ativas de modo a buscar informações referentes à conexão com a internet e qual a melhor forma de realizar a chamada, fazendo com que sempre seja necessário invocar o usuário para tomar as decisões. Para alcançar este diferencial, foi modelada uma arquitetura baseada em três módulos, que posteriormente foi mapeada e adaptada para utilização em conjunto com o software *open source* CSipSimple. O aplicativo citado disponibiliza algumas funcionalidades semelhantes às pretendidas e possibilitou a comprovação da estrutura modelada. O resultado final foi uma aplicação capaz de decidir o destino de uma chamada e tratar o encaminhamento correto para ela de maneira transparente, como foi proposto inicialmente.

Palavras-chave: Android. VoIP. SIP. Smartphones. Mobilidade. Ligações telefônicas. Discador.

MODELING A INTELLIGENT VOIP CLIENT FOR ANDROID PLATFORM

ABSTRACT

In a moment of mobile devices popularization and increasing supply to wireless and high speed Internet was developed this project. It aims to offer users a practical and simple solution to save money on phone calls, making choices between VoIP calls and calls over the cellular network. These choices are made transparently to user, not compromising the software experience. The application interface is a simple dialer that takes as input the destination number to be dialed. The purpose of this project was to undertake a study in order to make the solution available for the Android platform, chosen for its modularity and simplicity to replace system components, such as the dialer. The existing solutions are not proactive in order to seek information regarding the connection to the Internet and what better way to make the call, so is always necessary the user's decisions. To achieve this differential, an architecture was modeled based on three modules, which was subsequently mapped and adapted to be used with the open source software CSipSimple. The application mentioned provides some features similar to those intended allowing the comprovation of the structure modeled. The final result was an application that can decide and treat the call destination transparently, as was initially proposed.

Keywords: Android. VoIP. SIP. Smartphone. Mobility. Phone calls. Dialer.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GSM	Global System for Mobile
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
OSI	Open Systems Interconnection
RFC	Request for Comments
RTP	Real-time Transport Protocol
SDK	Software Development Kit
SIM	Subscriber Identity Module
SO	Sistema Operacional
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
XML	Extensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

Para o entendimento dos fatores que levaram à idealização do projeto, assim como para a percepção das possibilidades que o mesmo cria é necessária uma contextualização do momento tecnológico e do cenário em que o mercado se encontra. Também é de suma importância a justificativa da escolha do assunto para este trabalho. As seções posteriores têm por objetivo realizar estes esclarecimentos e tornar mais claro o acompanhamento do texto.

1.1. ABORDAGEM DO TEMA

No atual mercado de telefonia móvel nos deparamos com um cenário onde os *smartphones* estão em um desenfreado aumento de desempenho e capacidade, alcançando recursos de memória e processamento que alguns anos atrás só eram disponíveis em notebooks ou desktops. Aliadas a estes poderosos recursos estão particularidades únicas a esses aparelhos, que são a grande capacidade de comunicação e a mobilidade que possuem. Nos últimos meses, com a proliferação do sistema Android nestes dispositivos, e com a promessa de aumento deste crescimento, como pode ser visto na Tabela 1, aumentaram também as possibilidades de utilização destes recursos, já que esta plataforma possibilita um poder e uma liberdade muito grande aos desenvolvedores e usuários.

Sistema Operacional	**Participação no mercado em 2011**	**Previsão de Participação no mercado em 2015**	**Crescimento (2011 - 2015)**
Android	39.5%	45.4%	23.8%
BlackBerry	14.9%	13.7%	17.1%
iOS	15.7%	15.3%	18.8%
Symbian	20.9%	0.2%	-65.0%
Windows Mobile	5.5%	20.9%	67.1%

Outros	3.5%	4.6%	28.0%
Total	100.0%	100.0%	19.6%

Tabela 1: Previsão de crescimento para plataformas móveis(IDC, 2011)

Por outro lado, as operadoras de celulares estão vendo seu poder sobre o usuário diminuir consideravelmente, uma vez que elas possuem cada vez menos controle sobre o uso que seu cliente dá a seu dispositivo móvel, fato comprovado pela quebra da barreira do bloqueio de celulares, disponível em (ANATEL, 2011) e pelo aumento do interesse dos usuários em explorar estes dispositivos.

É neste cenário que é proposta a criação de um aplicativo capaz de oferecer uma alternativa às ligações telefônicas convencionais pela rede de telefonia móvel. O software projetado deve optar pela forma mais econômica de ligação disponível em um determinado momento, de uma forma transparente ao usuário. Caso o originador da chamada esteja conectado na internet, via rede 3G ou wi-fi, e o destinatário também possuir uma conexão, possivelmente a ligação será completada por VoIP.

Para auxiliar o controle de usuários online e o tratamento do tráfego VoIP, este trabalho será complementado pelo projeto de Marcelo Bublitz Anton (ANTON, 2011), que consiste na implementação de um servidor capaz de gerenciar estas requisições.

1.2. MOTIVAÇÃO

Este trabalho foi motivado pelo interesse do autor em novas tecnologias e dispositivos móveis, assim como pelo entendimento de que o assunto abordado pode representar uma lacuna no atual mercado de mobilidade. A sua idealização foi realizada juntamente com o também estudante de Ciência da Computação na UFRGS e colega de trabalho, Marcelo Bublitz Anton.

1.3. OBJETIVOS

Como objetivos fundamentais deste trabalho temos um estudo aprofundado da arquitetura da pilha de software Android e o projeto e implementação de um software para substituir o discador padrão de dispositivos equipados com este sistema. Este discador deverá ter a capacidade de originar chamadas pela rede de telefonia móvel para uma terminação por esta mesma rede ou por VoIP, tanto no destino quanto na origem, e decidir qual a melhor opção em uma determinada situação. Mais tarde poderão ser anexadas funcionalidades, como a descoberta de preços praticados pelas operadoras baseando-se em serviços de localização.

1.4. ORGANIZAÇÃO DO TEXTO

O texto deste trabalho está organizado em seis capítulos que abordam diferentes tópicos necessários ao projeto. Seus assuntos serão especificados abaixo, visando um melhor entendimento da estrutura utilizada, assim como uma percepção do projeto em sua totalidade.

Capítulo 1 – Contextualiza o momento tecnológico da área de mobilidade e telefonia no período da criação do projeto, introduz o tema, a motivação e o objetivo buscado com a criação do mesmo. Por fim o capítulo situa o leitor na organização do texto.

Capítulo 2 – Aborda a tecnologia Android e as suas características principais. Justifica a atual participação de mercado deste sistema operacional e realiza um estudo sobre a arquitetura da plataforma e dos softwares desenvolvidos para ela, focando nas particularidades do Android e no conhecimento que será necessário para o desenvolvimento da aplicação desejada.

Capítulo 3 – Faz uma rápida introdução à tecnologia VoIP, destacando a topologia de rede VoIP propícia ao trabalho e abordando mais tecnicamente alguns componentes de sua arquitetura que serão utilizados no decorrer do

projeto. São mostradas algumas estruturas desta tecnologia que já são utilizadas na plataforma Android.

Capítulo 4 – Neste capítulo é modelada a arquitetura a ser utilizada no projeto e são introduzidas as principais funcionalidades e casos de uso da aplicação. São esboçadas possíveis soluções para os problemas que serão encontrados e é esclarecido o funcionamento do software desenvolvido com a ajuda de diagramas.

Capítulo 5 – Aqui será introduzido o protótipo construído. Serão apresentadas as funcionalidades alcançadas e as possibilidades de crescimento para elas. Serão mostradas as tecnologias utilizadas para sua construção e discutidas as limitações e dificuldades encontradas na implementação. Os códigos referentes às principais funcionalidades do sistema e algumas particularidades da comunicação da aplicação com o servidor também serão exibidos nesta seção.

Capítulo 6 – Nele serão apresentadas as descobertas e os aprendizados adquiridos com o desenvolvimento do estudo, assim como o resultado final e as possibilidades de utilização do protótipo.

2. PLATAFORMA ANDROID

Para o desenvolvimento de uma aplicação capaz de aproveitar ao máximo todos os recursos disponíveis em uma determinada plataforma, necessita-se um conhecimento mais profundo sobre sua arquitetura e funcionamento. Este capítulo irá introduzir questões básicas referentes à construção do sistema Android e definições utilizadas na criação de aplicações para o mesmo.

2.1. VISÃO GERAL

A criação do Android representou um marco no atual mercado de mobilidade, uma vez que a partir de sua criação houve uma corrida pelas empresas fabricantes de *smartphones* para disponibilizar no mercado aparelhos com este sistema, que além de estável era gratuito. Segundo pesquisa da Canalys (CANALYS, 2010), para algumas empresas o Android representou a possibilidade de recuperação no mercado de dispositivos móveis, já que suas plataformas estavam caindo em desuso e suas vendas de aparelhos diminuindo.

Por ser uma plataforma aberta e de fácil customização pelos fabricantes, em (ANDROID SOURCE, 2011) podem ser encontradas instruções para portar o sistema, o Android conseguiu algo inédito até então, que foi ser o sistema padrão de aparelhos de diversas marcas. Apesar desta fragmentação de fabricantes, a plataforma manteve sempre um desempenho satisfatório e com isto conseguiu abocanhar uma fatia significante do mercado em pouco tempo, se tornando uma alternativa barata para a produção e interessante para os desenvolvedores que buscavam uma maior padronização neste mercado tão heterogêneo.

O kit de desenvolvimento fornecido gratuitamente pela Google (ANDROID DEVELOPERS, 2010) possui uma grande variedade de ferramentas e a plataforma possui uma série de API's que possibilitam tarefas

aparentemente complicadas, como fornecer a localização do usuário em um mapa, ser realizada em pouquíssimas linhas de código.

Segundo (WALL, 2008), a plataforma Android é considerada uma pilha de software, a qual integra um sistema operacional (baseado no kernel do linux), bibliotecas de baixo nível para execução de operações básicas, um interpretador de *bytecode* da máquina virtual, bibliotecas para desenvolvimento e aplicativos essenciais. Todas essas ferramentas possibilitam o sistema transparecer para o usuário como sendo muito prático, integrado e estável.

O usuário também possui uma liberdade muito grande utilizando o sistema, até mesmo as aplicações mais básicas para um telefone, como o discador ou o gerenciador de contatos, podem ser substituídos por aplicações de sua preferência. Isto acontece devido ao sistema de provedores de conteúdo utilizado, que será detalhado mais adiante, e fornece uma comodidade ao programador que não precisa alterar componentes de sistema para realizar uma operação deste tipo.

Para desenvolver para esta plataforma é utilizada a linguagem de programação Java, e as aplicações rodam sobre uma máquina virtual chamada Dalvik, a qual possui algumas otimizações em relação a uma JVM convencional para possibilitar um menor consumo de memória, se adequando aos padrões para um dispositivo portátil.

2.2. ARQUITETURA

Como citado em (WALL, 2008), a plataforma Android não é considerada apenas um sistema operacional, ela é um conjunto de ferramentas e bibliotecas acopladas a um kernel do Linux formando um sistema mais completo e robusto. Essas ferramentas interagem entre si em um formato de pilha, onde cada camada oferece recursos para a camada imediatamente acima, inspirando-se em um modelo já conhecido em redes de computadores chamado modelo OSI. A estrutura geral do Android pode ser encontrada em (GOOGLE DEVELOPERS, 2011) e é exibida na Figura 1.

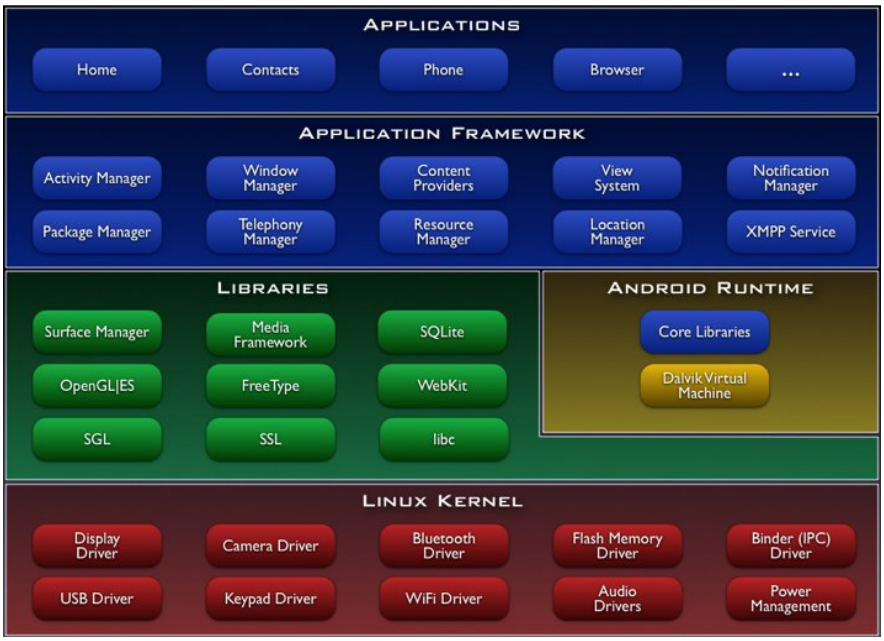


Figura 1: Arquitetura Android

No seu núcleo está um kernel do Linux, na versão 2.6 com algumas alterações, que é responsável pelo gerenciamento de memória e processos, além do sistema de arquivos. Junto a ele estão os drivers de mais baixo nível, que controlam a tela, o teclado, dispositivos de rede, gerenciamento de energia, entre outros. Estas estruturas fornecem a base sobre a qual a plataforma é construída.

Na camada logo acima encontram-se bibliotecas escritas em C e C++ que são responsáveis pela parte gráfica (Surface Manager e OpenGL), multimídia, banco de dados (SQLite), suporte a *browsers* (WebKit), além de camadas de rede e gerenciamento de fontes. Estas bibliotecas prestam serviços à camada seguinte que pode ser chamada de *framework* de aplicações. Neste nível encontram-se as API's de suporte ao desenvolvimento, que simplificam o acesso ao sistema de localização, de telefonia, de notificações e outros, além de prover a estrutura dos programas do Android, gerenciando o sistema de *views,* os *content providers,* as *activities* e o sistema de janelas.

Segundo (WALL, 2010), para comunicação entre as bibliotecas e as API's e para a execução do sistema em si existem duas alternativas:

- Bibliotecas de código – Segundo (WALL, 2008), é uma combinação de bibliotecas semelhantes às encontradas no Java Standart Edition (Java SE), porém sem algumas funcionalidades desnecessárias a um dispositivo móvel, como o Swing, assim como remoções de funcionalidades redundantes.

- Dalvik Virtual Machine – Uma JVM com diversas alterações feitas pelo Google, visando à diminuição do uso de memória e maior desempenho da mesma. A Dalvik consegue estes resultados por ser baseada em registradores e por não utilizar o *bytecode* Java. O código binário utilizado possui um formato próprio e é gerado a partir do *bytecode* Java, possuindo algumas simplificações e compactações.

Por fim encontra-se a camada de aplicativos básicos do Android. A plataforma fornece todo o software básico necessário para a boa utilização de um dispositivo móvel. Estes softwares podem ser substituídos em caso de preferência do usuário sem que seja comprometido o funcionamento do sistema. Também existe a possibilidade de serem instalados inúmeros outros softwares agregando todo o tipo de funcionalidade ao aparelho.

2.3. APLICAÇÕES NO ANDROID

Uma aplicação Android é construída a partir de quatro tipos básicos de componentes (LOMBARDO, 2009). Estes componentes possuem características próprias as quais os fazem mais apropriados para determinadas situações. Para a qualidade de uma aplicação Android é muito importante que essas partes estejam bem alinhadas e com uma comunicação adequada, caso contrário o aplicativo pode não apresentar o efeito desejado.

Essa padronização nas responsabilidades de cada modelo de componente gera uma reutilização de funcionalidades muito grande, otimizando a gerência de memória em um dispositivo com esta plataforma. Porém, algumas características particulares, como a de cada aplicativo rodar em seu próprio usuário no sistema utilizando uma instância da máquina virtual exclusiva faz com que cada um rode seu código isoladamente.

Abaixo estão listados os componentes os quais são a base para as aplicações na plataforma, assim como uma breve descrição de suas responsabilidades e alguns exemplos. Maiores detalhes podem ser obtidos em (LOMBARDO, 2009) e (ANDROID DEVELOPERS, 2011).

- *Activities* – São componentes executáveis e reutilizáveis instanciados pelo usuário ou pelo sistema operacional. Têm por objetivo interagir com o utilizador ou com outras *Activities* ou *Services* trocando serviços ou informações através de *Intents*. Cada *Activity* pode ser considerada uma *UI* e possui uma tela associada para comunicação com o usuário. O sistema operacional pode encerrar este componente para gerência de memória, caso ele não esteja ativo. A maioria dos componentes de uma aplicação Android é uma *Activity*.

- *Services* – São uma espécie de *daemons* que possuem um período longo de vida e rodam em plano de fundo. Normalmente são utilizados quando é necessário prover um serviço por um longo tempo sem ter problemas de encerramento, ou quando são componentes que não necessitam de uma interface com o usuário. Um exemplo prático de utilização de um *Service* é quando se necessita fazer um download. Uma tarefa deste tipo roda em background e não pode ser finalizada antes de completo, logo precisaríamos de um *Service* para ela.

- *Intent and Broadcast Receivers* – Pode ser considerado um gerenciador de notificações e eventos, oriundos do sistema ou de aplicativos. Quando uma aplicação requer comunicação com outros módulos do dispositivo, sendo eles aplicações (por exemplo, compartilhamento de informações de um contato) ou o próprio sistema (por exemplo, sinalizar chamada perdida), ela deve possuir um componente deste tipo. A comunicação é realizada através de uma *intent,* uma espécie de requisição que é enviada ao sistema para que ele gerencie e decida qual módulo é responsável por receber uma mensagem daquele tipo.

- *Content Providers* – É o componente responsável por compartilhar dados entre aplicativos. Funcionam através de uma URI, onde determinados

aplicativos são responsáveis por atender certos padrões de URI, possibilitando a comunicação entre serviços que não se conheçam. Também têm um papel importante no quesito de segurança, pois são responsáveis por restringir o acesso aos dados por programas não autorizados.

Uma aplicação não precisa fazer uso de todos estes componentes em sua estrutura, a única necessidade é ter pelo menos uma *Activity* para executar o aplicativo e/ou interagir com o usuário. Estes recursos de bibliotecas estão todos disponíveis para utilização no SDK disponibilizado pelo Google, assim como dispositivos virtuais para *deploy* e teste dos aplicativos, *drivers* para comunicação com aparelhos móveis e todas as ferramentas necessárias para desenvolvimento nesta plataforma.

Depois de pronta, a aplicação é empacotada no formato *.apk.* Este pacote que será instalado no dispositivo equipado com Android contém, além de outros arquivos, um chamado *classes.dex,* um chamado *AndroidManifest.xml* e uma pasta chamada *res*. A pasta contém todas as mídias utilizadas pela aplicação. O *.dex* possui o binário das classes no formato da Dalvik VM, semelhante ao *.class* do Java. O AndroidManifest é um roteiro de instalação do aplicativo, além de possuir o sistema de permissões da aplicação, que será apresentado ao usuário no momento da instalação.

2.4. CONSIDERAÇÔES FINAIS

A partir deste estudo foi possível entender melhor o funcionamento da plataforma e suas particularidades, possibilitando a percepção de que o sistema oferece todas as estruturas e componentes necessários ao desenvolvimento do projeto. Na seção subsequente seram salientados alguns destes componentes, relacionando-os com elementos da tecnologia VoIP, fazendo com que fique mais claro a importância dos mesmos para o projeto.

3. TECNOLOGIA VOIP

Para utilização dos componentes da arquitetura VoIP de uma maneira inteligente e eficaz, capaz de prover para a aplicação desenvolvida todos os recursos necessários para o correto funcionamento das chamadas, será feito um *overview* de suas características. Serão abordados os elementos mínimos necessários para a utilização desta tecnologia.

3.1. INTRODUÇÃO

A tecnologia VoIP, a qual permite a transmissão de voz em tempo real via Internet, vem em um crescimento contínuo no mercado mundial, segundo (FROST & SULLIVAN, 2011). Sua popularização iniciou-se com a chegada ao mercado do software Skype (SKYPE, 2010), em 2003. Este produto foi um dos precursores deste tipo de serviço para o usuário final e a partir dele, na velocidade que as tecnologias de rede permitiram, desencadeou-se uma migração de usuários, visando principalmente os custos baixos para ligações de longa distância encontrados em VoIP, diferentemente dos valores elevados praticados via rede móvel convencional.

Inicialmente as limitações para serviços deste tipo foram as conexões com a Internet, que possuíam baixas velocidades, tornando complicado o tráfego de voz por redes domésticas. Outro fator que impediu o rápido crescimento da tecnologia foi o desconhecimento de sua capacidade, que não era levada muito a sério por grandes empresas do setor de telecomunicações, com medo de perderem participação no mercado. Porém a difusão de conexões melhores e a percepção das grandes operadoras que poderiam ganhar dinheiro a partir deste tipo de serviço fez crescer o percentual de mercado alcançado.

Os grandes benefícios desta tecnologia, que possibilitam sua crescente utilização, são os baixos custos de operação e uma gama maior de possibilidades para os usuários. Pelo fato de não utilizar uma rede própria, ou seja, se aproveitar das já existentes redes de dados, os custos desta tecnologia

podem ser reduzidos drasticamente. Outra característica é a de não haver muita diferenciação, em termos de custos, de uma chamada que será feita para uma cidade vizinha, ou uma para o outro lado do mundo.

Além da diminuição de gastos, VoIP propicia que sejam anexadas nas ligações outros recursos facilitadores ao usuário. Uma possibilidade é a de transformar chamadas de áudio em ligações com recursos de vídeo e até mesmo localização. Também é permitido o destino de uma chamada ser roteado de acordo com a vontade do usuário, possibilitando o serviço ser utilizado independentemente de onde a pessoa esteja.

Hoje VoIP é apontado por especialistas como a tendência para substituir completamente o modelo convencional de telefonia via PSTN, segundo pesquisa da (HARRIS INTERACTIVE, 2011), nos EUA sua participação no mercado chega a quase 30% . É estimado que inicialmente a tecnologia VoIP substitua as ligações à longa distância, e que talvez, em um futuro não muito distante, toda a rede PSTN caia em desuso.

3.2. ESTRUTURAS BÁSICAS

Nesta parte do trabalho será abordado um único modelo para as chamadas, que será a topologia onde o originador e o destinatário da ligação encontram-se utilizando um cliente VoIP, ou seja, a rede PSTN não será envolvida nas chamadas VoIP, pois não é o foco do estudo. Para o cenário escolhido o *backend* necessário para se completar uma chamada não é muito complexo.

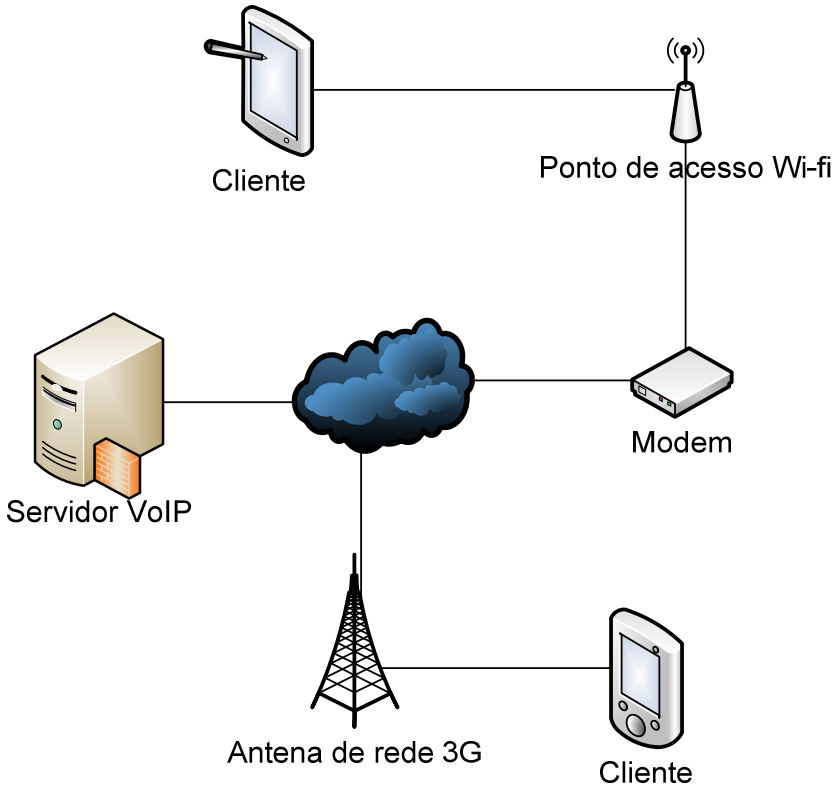


Figura 2: Estrutura VoIP básica

Considerando esta topologia, um dos principais componentes da arquitetura é o cliente, também conhecido por *softphone* ou agente. Hoje existem *softphones* para múltiplas plataformas, permitindo que qualquer um com um computador e conexão com a internet possa realizar chamadas VoIP. O agente é responsável pela transcrição da chamada para um determinado protocolo, gerando uma sinalização que será interpretada pelo cliente de destino, assim como a transmissão do áudio da ligação. O cliente também possui uma interface para interação com o usuário e funcionalidades de um telefone convencional.

Segundo (DAVIDSON, 2000), a geração de sinalização pelo cliente pode usar diversos protocolos, como SIP, XMMP e H.323, que serão analisados ainda neste capítulo. O único requisito para o estabelecimento da comunicação é que tanto *softphone* originador quanto o destinatário utilizem o mesmo protocolo, assim como o mesmo codec. O codec será responsável pela

compressão e descompressão do áudio para um melhor aproveitamento de banda, os codecs mais utilizados são o G.729 e o G.711.

Outro componente fundamental neste cenário é o Proxy. Ele é quem vai receber a sinalização enviada pelo cliente e repassá-la, ou para outro servidor Proxy, ou para a terminação. Para descobrir o destino de uma chamada ele pode utilizar DNS, consultar uma base local, ou encaminhá-la para outro servidor, o qual conheça um caminho até o destino. Além de fazer o encaminhamento da sinalização este servidor pode gerenciar o balanceamento de tráfego e colher informações e estatísticas das chamadas.

Com este modelo apresentado já é possível realizar chamadas VoIP. As tecnologias possíveis de serem utilizadas para cada módulo citado serão abordadas nos subcapítulos seguintes.

3.2.1. PROTOCOLOS VOIP

Para a comunicação entre softwares clientes e para a sinalização das chamadas VoIP existe uma série de protocolos em uso no mercado, cada um com algumas particularidades. Abaixo serão listadas algumas características dos mais utilizados.

- SIP (Protocolo de Iniciação de Sessão) – É um protocolo da camada de aplicação para a sinalização do tráfego multimídia. Funciona independentemente do tipo de mídia e de aplicação utilizados. Segue um modelo, similar ao HTTP, de requisições e respostas e foi especificado pela RFC 3261 de 2002. Sua principal características, além da independência da camada de transporte, é a simplicidade. Esse protocolo possui apenas seis métodos.

- H.323 – Este protocolo possui a mesma independência de outras camadas que o protocolo SIP possui. É o recomendado pela ITU-T, entidade regulamentadora da área. Sua complexidade é superior ao anterior, pois possui uma variedade maior de elementos, além disso, o H.323 não garante QoS (Qualidade de serviço). Para a utilização deste padrão é necessário haver

transmissão de áudio obrigatoriamente, o que impede sua utilização em serviços de mensagens instântaneas por exemplo.

- XMPP – É um protocolo baseado em XML que teve suas origens no extinto Jabber. Sua principal utilização é para mensagens instantâneas, mas também pode ser utilizado em chamadas VoIP. Uma característica importante é o fato de ser um protocolo aberto, permitindo ser estendido.

3.2.2. SERVIDOR PROXY

O servidor Proxy é o responsável por encaminhar o tráfego VoIP do cliente origem ao cliente destino. Caso o servidor conheça o destino, este encaminhamento pode ser realizado diretamente ou consultando uma base de dados a qual ele possua. Caso o servidor desconheça uma rota até o cliente destino, o tráfego pode ser repassado a outro servidor capaz de fazer a entrega.

Um dos softwares mais completos e utilizados para este propósito chama-se Opensips (OPENSIPS, 2010). Sua característica principal é a capacidade de permitir de certa forma programar cenários para o tratamento da sinalização VoIP. Esta programação pode envolver balanceamento, roteamento e inúmeros outros tratamentos para a sinalização.

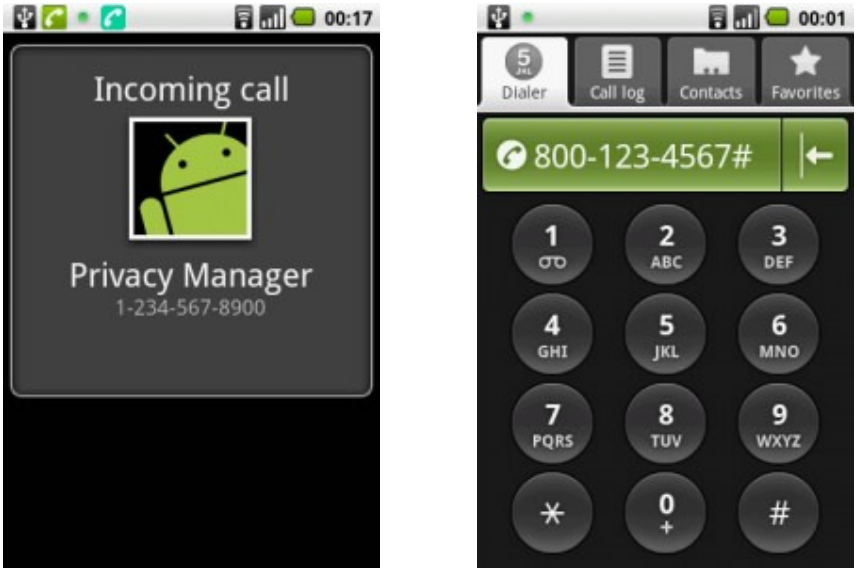
3.3. ESTRUTURAS PRESENTES NO ANDROID

O Android não oferecia suporte a VoIP nativo até a sua versão 2.2 (Froyo), que era a corrente no início deste projeto, porém juntamente com a sua versão 2.3 (Gingerbread), lançada em dezembro de 2010, foi adicionado o suporte para esta funcionalidade. Como no atual período, o mercado brasileiro ainda é pouco abastecido por aparelhos com esta versão, não serão utilizadas as facilidades criadas pelo surgimento desta nova API neste trabalho.

Existem aplicativos de terceiros capazes de oferecer chamadas de VoIP em um aparelho com sistema Android. Para suportar uma ligação deste tipo é

necessário um aplicativo capaz de interpretar algum dos protocolos citados nos capítulos anteriores, uma conexão com a internet (wi-fi ou 3G) e acesso às funcionalidades de chamadas do Android. Os recursos externos necessários ao dispositivo móvel para uma operação deste tipo, como o servidor VoIP, não serão abordados nesta seção.

Quando se fala em ligações de Voz sobre IP no Android já existe mais de um aplicativo bastante difundido entre usuários e desenvolvedores, um deles é chamado *Sipdroid* (SIPDROID, 2010), suas telas podem ser vistas nas Figuras 3 e 4. Este aplicativo é um cliente *open source* para o protocolo SIP que adiciona ao discador padrão da plataforma suporte à tecnologia VoIP. Entre as principais vantagens do *Sipdroid,* além da integração com o sistema e possuir seu código aberto, estão também as grandes possibilidades de *customizações* e configurações avançadas que ele possui, deixando o usuário livre para montar sua própria infraestrutura independente de serviços proprietários.



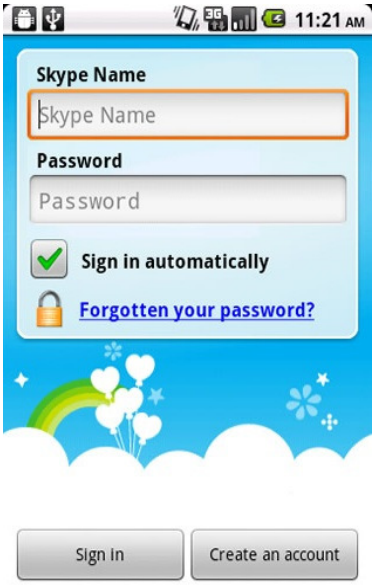
Figuras 3 e 4: Screenshots do aplicativo Sipdroid.

No mesmo nível do *SipDroid* existe uma solução chamada *CSipSimple* (CSIPSIMPLE, 2010), que prove funcionalidades parecidas com a aplicação citada anteriormente. Seu discador se integra perfeitamente com o discador nativo, tornando a experiência do usuário mais agradável. Para os tratamentos

de mais baixo nível, este aplicativo utiliza uma biblioteca *open source* chamada *PJSIP.* Esta biblioteca é escrita na linguagem C e disponibiliza as principais operações necessárias para um aplicativo que utilize o protocolo SIP para realizar chamadas VoIP. Nas Figuras 5 e 6 é possível ver algumas imagens deste software em funcionamento.

Figuras 5 e 6: Screenshots do aplicativo CSipSimple.

Outra opção muito conhecida para soluções VoIP é o software *Skype* (SKYPE, 2010), que se consagrou em outras plataformas e agora possui uma versão para Android. O *Skype* utiliza seu próprio protocolo no modelo *peer-to-peer* e restringe a comunicação com outras redes VoIP, o que simplifica as configurações necessárias para o tráfego de chamadas, mas impossibilita seu aproveitamento por outros serviços. A base de usuários registrados no *Skype* é bastante grande, mais de 600 milhões segundo (WIKIPEDIA, 2010), o que o coloca entre os principais serviços de comunicação da internet, abrangendo tanto o modelo de comunicação de dois clientes VoIP, quanto o modelo de comunicação de um cliente VoIP para um cliente de telefonia convencional. Nas imagens 7 e 8 é possível ver algumas telas da aplicação citada.



Figuras 7 e 8: Screenshots do aplicativo Skype para Android.

Além dos dois produtos citados, existe uma grande variedade de soluções VoIP no Market, loja de aplicativos do Google para o Android, porém nenhum destes *softwares* possui características inéditas aos produtos apresentados. A expectativa é que aconteça com os *smartphones* um movimento semelhante ao que houve com os PCs, ou seja, com a melhora das conexões 3G irão surgir mais usuários e mais aplicativos para a tecnologia VoIP *mobile*.

3.4. CONSIDERAÇÕES FINAIS

Todas as soluções analisadas possuem um incoveniente em comum: a necessidade de um prestador de serviço para o completamento das chamadas. A utilização de um provedor deste tipo normalmente vem atrelada a um custo, que apesar de inferior à telefonia convencional, pode ser desconsiderado se utilizada a solução proposta. No capítulo seguinte será criado um modelo de arquitetura capaz de prover esta solução.

4. MODELAGEM DA ARQUITETURA

Neste capítulo serão abordados os objetivos do projeto e as etapas e componentes necessários para a sua conclusão. Será traçado um modelo de comunicação entre os módulos *mobile* e o servidor VoIP do sistema e será planejada a estrutura do cliente para Android. Por fim, serão estabelecidas as responsabilidades deste trabalho e as responsabilidades atribuídas ao servidor desenvolvido por (ANTON, 2011).

4.1. INTRODUÇÃO

O objetivo desta modelagem é fornecer ao usuário um discador na plataforma Android, que seja capaz de realizar tanto ligações da maneira tradicional via rede móvel da operadora, quanto ligações de VoIP para VoIP de maneira transparente. Estas funcionalidades podem ser muito úteis para tomada de decisão de qual tecnologia utilizar baseada no preço ou na localização.

Para isso será necessário um módulo *mobile,* capaz de utilizar a estrutura nativa do Android para fazer chamadas pela rede convencional e adicionalmente um cliente VoIP para realizar as chamadas deste tipo. O outro grande módulo deste projeto seria um servidor externo ao dispositivo, capaz de registrar os aparelhos e prover a comunicação entre os clientes Android.

O caso de uso básico deste aplicativo seria: O cliente abre o discador, digita o número de destino e tecla *dial*. Neste caso o discador desenvolvido fará uma consulta ao sistema Android para descobrir a existência de uma conexão com a internet. Em caso negativo gerará uma *intent* ao sistema para completar a ligação de maneira convencional. Em caso positivo fará uma consulta ao servidor em busca de informações sobre o destino.

A consulta ao servidor terá por objetivo descobrir se o destino da chamada está ou não conectado na internet para a ligação ser completada por VoIP. O servidor terá esta informação armazenada em um banco de dados

local que será atualizado em um período curto de tempo por uma chamada *keepalive* dos dispositivos clientes. Caso o cliente destino não esteja conectado à internet, é aplicado o mesmo caso de quando o cliente origem não tem conexão, ou seja, a chamada é feita pela rede móvel convencional. Caso o destino também possua uma conexão com a internet, o agente VoIP assume o controle gerando a sinalização e a enviando ao servidor juntamente com o áudio comprimido.

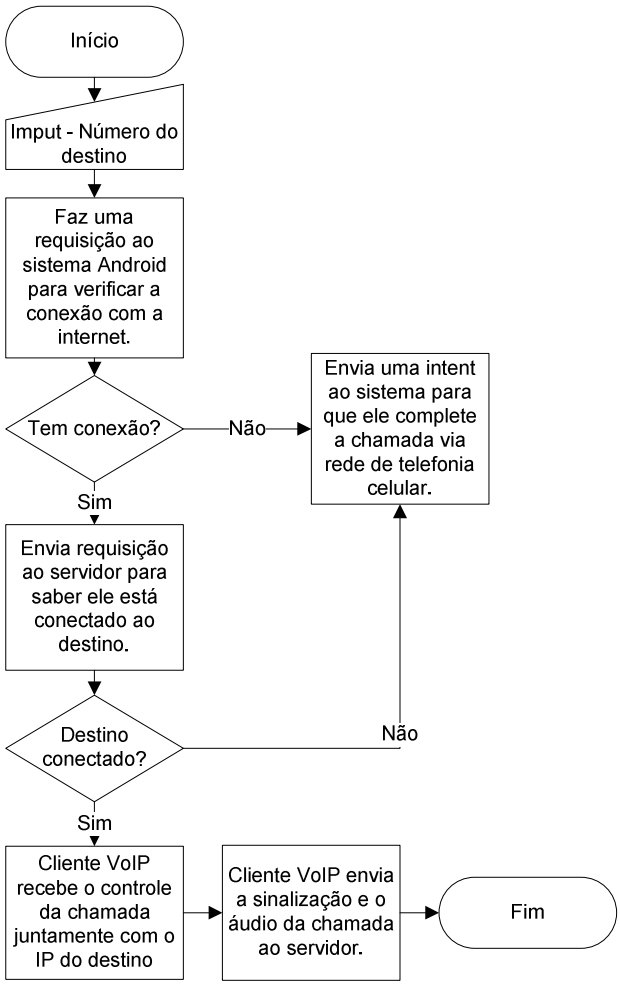


Figura 9: Fluxograma do caso de uso básico.

4.2. PLANEJAMENTO DA ARQUITETURA

O modelo de estrutura pensado, durante o estudo das tecnologias envolvidas, para a construção da parte *mobile* do projeto é baseado em três

módulos: o discador em si, um agente VoIP e um *daemon*. Todos eles teriam participações importantes no funcionamento do sistema, porém esta não seria a única abordagem possível ao problema. Questões referentes ao servidor serão tratadas sem detalhamento, pois ficarão a cargo do projeto complementar.

Neste modelo de arquitetura, ao ser iniciada a aplicação, o discador exibiria a tela principal (tela de discagem) e de forma transparente se registraria no *backend*. Para isso seria necessário informar o número do telefone, que será armazenado em um banco no servidor e servirá como identificador de uma instância da aplicação. A partir disto o discador instanciaria o agente VoIP e ficaria pronto para receber e originar chamadas. Caso recebesse um número discado pelo usuário, ele faria toda a inteligência de consultas ao servidor sobre a disponibilidade do destino e tomaria as decisões necessárias baseadas em critérios pré-definidos. Caso a opção considerada for uma chamada via VoIP, os parâmetros necessários seriam passados ao agente.

O agente se encarregaria do tratamento da chamada utilizando o protocolo definido. Toda a parte de sinalização e áudio seria tratada por este componente que se comunicaria diretamente com o proxy VoIP existente na infraestrutura. Este módulo também agiria de forma passiva quando estivesse aguardando por chamadas VoIP, onde a instância da aplicação em questão é o destinatário.

O papel do *daemon* seria realizar o registro da aplicação junto ao servidor e mantê-lo sempre atualizado sobre o estado de conectividade com a internet do dispositivo, assim como passar informações necessárias para o completamento das chamadas, como o endereço IP do cliente ou seu identificador.

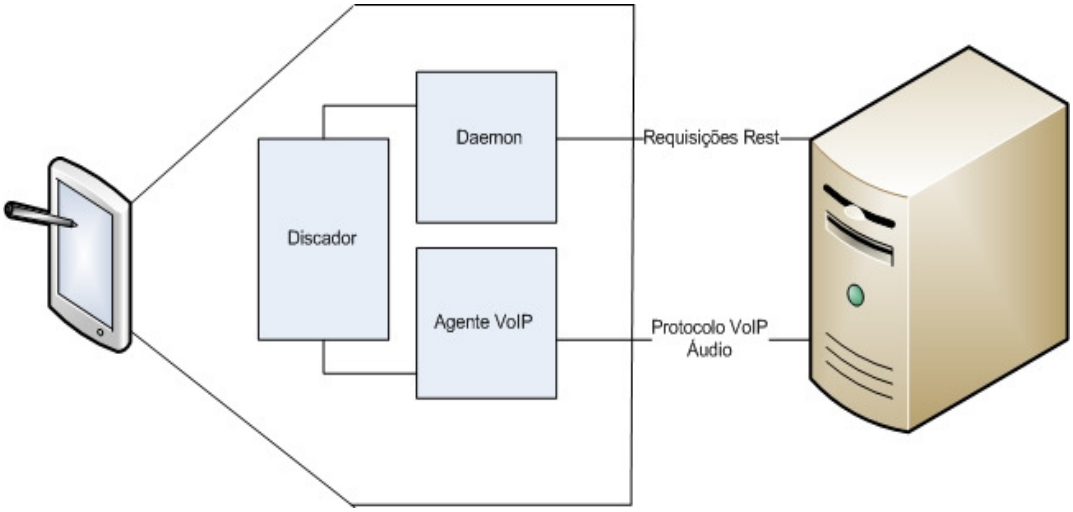


Figura 10: Modelagem aplicação mobile

Na Figura 10 encontra-se um esboço da arquitetura planejada. Cada componente citado como participante desta arquitetura será abordado com maiores detalhes nos subcapítulos seguintes.

4.2.1. PROTOCOLO VOIP

Depois da arquitetura básica, o passo seguinte para o planejamento do aplicativo foi a definição do protocolo VoIP a ser utilizado, para possibilitar, posteriormente, definições sobre outros componentes do sistema. A escolha do protocolo é uma peça fundamental para prever as possíveis características e problemas que a construção do sistema irá encontrar.

Para a escolha desta parte importante da aplicação foi considerado alguns fatores como: a maturidade da tecnologia, a sua disseminação, a facilidade de uso, as ferramentas existentes e o conhecimento do autor nesta tecnologia. Baseando-se nestes fatores, foi decidido pela adoção do protocolo SIP (*Session Iniciation Protocol*), que se destaca por estar presentes em várias ferramentas que poderão ser usadas neste trabalho (Sipdroid, CSipSimple, Opensips, Asterisk, etc.), por possuir uma grande popularidade na comunidade de desenvolvedores e por o autor possuir certa familiaridade com este protocolo.

Em termos de performance o SIP também ganhou pontos se comparado ao H.323, seu principal concorrente para ser utilizado. Devido a ele possuir uma maior simplicidade, são necessários menos pacotes para a efetivação de uma chamada, ele também possui uma maior economia de banda. Com isso, o quesito rede, que em dispositivos móveis é uma métrica bastante crítica, não é tão prejudicado.

4.2.2. COMUNICAÇÃO

Na modelagem da aplicação fica evidente a utilização de uma arquitetura cliente-servidor, onde a aplicação *mobile* requisita serviços ao servidor VoIP. Estes dois módulos estarão conectados através da internet e para tal é necessário a definição do protocolo desta comunicação.

A comunicação com o *backend* acontecerá de duas formas distintas. Em um primeiro momento a aplicação fará requisições com o objetivo de se registrar, buscar informações sobre o status do destinatário da chamada e informar se está conectado. Essas requisições serão feitas a um *web service* presente no servidor. Posteriormente a comunicação será com o proxy SIP no nível de enviar a sinalização e áudio de uma chamada VoIP.

Para a comunicação via *web service* será utilizada a tecnologia REST (*Representational State Transfer*), especificada por Roy Fielding em (FIELDING, 2000). Este modelo utiliza o conceito de URI, onde cada método possui um caminho único como identificador. Na camada de aplicação, apesar da não definição de um protocolo padrão, normalmente é utilizado o HTTP. Outra característica fundamental deste protocolo é a não manutenção de estado durante a comunicação, ou seja, cada requisição HTTP com uma URI associada possui toda a informação necessária para a transação.

O *web service* deverá prover 3 métodos básicos para a aplicação *mobile*. A partir destes métodos será possível fornecer para o software as informações necessárias para um funcionamento dos requisitos básicos. O primeiro método é o de registro no servidor:

- registrar (numeroTelefoneCliente);

Este método tem por objetivo informar ao servidor que o cliente está pronto para receber chamadas via VoIP. Internamente o *backend* deverá registrar em seu banco o número de telefone passado como parâmetro. Este registro será a identificação do cliente na rede criada.

- ativar (numeroTelefoneCliente);

Este segundo método tem por objetivo manter o status de um cliente como ativo. Ou seja, enquanto este método estiver sendo chamado periodicamente o servidor saberá que o cliente identificado pelo número passado como parâmetro permanece conectado e é capaz de receber uma chamada. Há a possibilidade deste método ser mesclado ao anterior, neste caso sempre seria chamado o "registrar" e o servidor faria a inteligência de registrar quando o identificador não estiver cadastrado e apenas manter como ativo quando já houvesse o registro.

- ligar (numeroTelefoneDestino);

O terceiro método tem por objetivo autorizar a realização de uma chamada por VoIP. Para que isto ocorra, será enviado ao servidor uma requisição passando como parâmetro o número de destino. O servidor deverá consultar em seu banco se o cliente identificado por aquele número está registrado naquele momento. Em caso positivo ele envia uma mensagem de

confirmação para o cliente, em caso negativo, ele informa a impossibilidade da realização da chamada.

A comunicação entre o agente VoIP e o Proxy seguirá os padrões SIP, definidos em (IETF, 2011), a sinalização trafegará via protocolo SIP utilizando UDP na porta 5060 e o áudio trafegará via o protocolo RTP utilizando UDP no intervalo de portas 10000 até 20000.

4.2.3. DAEMON

A estrutura de *backend* do projeto necessita sempre estar o mais atualizada possível com o status e as informações da aplicação *mobile*. Para garantir essa coesão entre os dois módulos, é necessário desenvolver uma estrutura para gerenciar esta comunicação e possibilitar um correto funcionamento do software.

Como visto nos capítulos anteriores, no Android existe um componente diferenciado para ser utilizado em situações onde há necessidade de se garantir a execução de um código ininterruptamente, este componente é o serviço. Utilizando esta estrutura, podemos ter certeza que o componente capaz de informar ao servidor o status da aplicação vai estar sempre rodando.

Um serviço no Android, além de garantir a execução de um trecho de código, permite que uma aplicação continue executando mesmo depois de todas as suas telas terem sido encerradas e disponibiliza uma interface para comunicação interna e externa a uma tarefa. Esta característica pode ser importante no desenvolvimento do software, pois pode ser uma forma segura de troca de mensagens entre os módulos discador e agente, caso eles sejam desenvolvidos de forma independente.

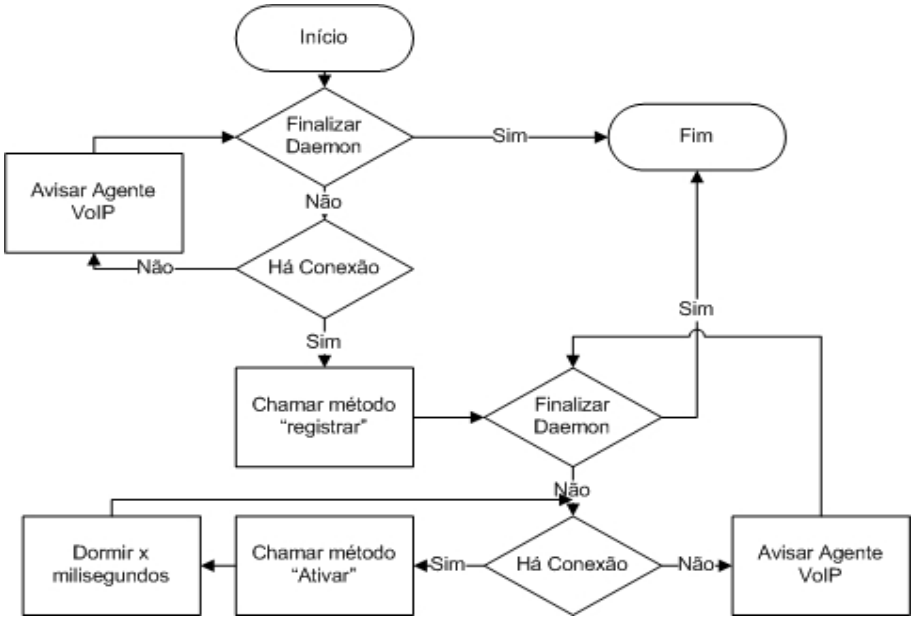


Figura 11: Fluxograma funcionamento do daemon.

A função básica deste serviço, como visto na Figura 11, será, ao se inicializar, testar a conexão com a internet e chamar o método "registrar" do *web service.* Após isso, o serviço deverá testar a conexão novamente e em caso dela estar ativa acionar o método "ativar" do servidor para informar que aplicação continua viva e capaz de efetuar chamadas utilizando a rede VoIP. Em caso de perda de conexão com a rede, acarretando impossibilidade de envio de dados, deve ser previsto um *timeout* no servidor para então considerar aquela aplicação *mobile* como não operante. Para permitir a finalização do *daemon* pelo usuário, deve ser realizado um controle sobre a necessidade de encerrar a aplicação.

Um ponto importante a ser lembrado é que o procedimento de chamada do método "ativar" terá que ser periódico. Para definir o intervalo de repetição desta tarefa terá que ser levado em conta um período que permita uma baixa taxa de erros, no sentido do servidor pensar que a aplicação está funcionando, mas na verdade ela não está, e que ao mesmo tempo não consuma em excesso recursos, como banda e processamento, do dispositivo móvel. Em termos de desempenho, este módulo deve ser o mais crítico da aplicação e que inspirará mais cuidados, por ser o único que estará rodando continuamente.

Para permitir o encerramento do *daemon* e consequentemente impossibilitar a realização e o recebimento de chamadas VoIP, poderá ser desenvolvido um ícone junto à barra de notificações do sistema operacional, o qual permita habilitar e desabilitar esta função. Para isso esta funcionalidade deve estar presente na interface de comunicação do serviço do *daemon*.

4.2.4. DISCADOR

O discador será a cara e o cérebro da aplicação, ele é o único componente que possui interface com o usuário, além de conter toda a lógica de funcionamento do software. Sua base deverá ser construída em cima de uma tela semelhante à do discador padrão da plataforma, que é mostrado na Figura 11.

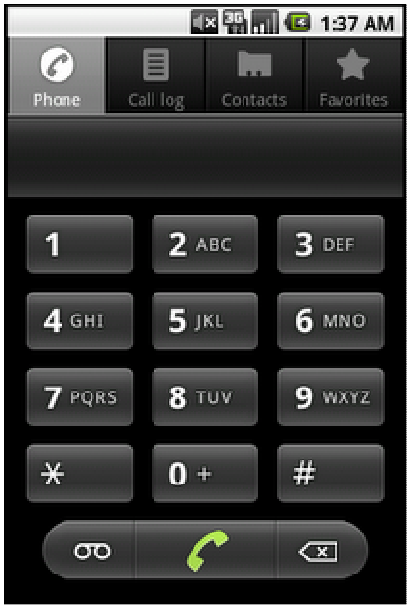


Figura 12: Discador padrão do Android Eclair (2.1).

A tela deverá disponibilizar as mesmas informações da imagem mostrada acima, porém algumas alterações de funcionamento deverão ocorrer na aba do telefone. Estas alterações serão citadas no decorrer do texto. Em caso de clique em uma aba diferente da aba de telefonia deverá ser lançada uma *intent* solicitando aquele recurso ao sistema, podendo ser o log de

chamadas, a lista de contatos ou os contatos favoritos. Há a possibilidade do botão físico de menu dos aparelhos Android chamar uma tela de configuração neste ponto da aplicação, porém esta tela só deverá existir caso haja alguma configuração não automática a ser feita, o que deve ser evitado.

Ao instalar e inicializar a aplicação pela primeira vez o usuário receberá uma pergunta do Sistema Operacional para saber se ele deseja substituir o discador atual pelo recém instalado. O usuário deve responder que sim, então o aplicativo se inicializará e verificará a conexão com a internet. Em caso de conexão presente será acionado o método registrar do *web service* e será exibida a tela mostrada acima. A partir deste momento toda vez que o aparelho for reiniciado o serviço correspondente ao software será inicializado junto fazendo as checagens necessárias.

Na aba de telefonia da tela do discador o comportamento será o padrão até a discagem do número de destino e o clique na tecla dial. A partir daí surgirão as modificações necessárias para o controle do fluxo da chamada, porém uma das funcionalidades pretendidas é que o usuário não perceba estas alterações e siga o mesmo procedimento de ligação ao qual ele está acostumado.

Inicialmente a regra para a decisão de qual rede usar será: utilizar VoIP em caso da conexão com o servidor ser estabelecida e em caso contrário utilizar a rede GSM. Então, caso não satisfeita a condição de conectividade, será sinalizado ao sistema para fazer a chamada pela rede de telefonia. Caso a condição seja satisfeita, será passado o controle ao agente VoIP, que receberá como parâmetros o número de telefone de destino e o endereço do servidor VoIP.

Como pode ser visto, além de ser a interface do sistema, o módulo discador é quem faz a interligação com os outros componentes, tendo uma importância significativa para o funcionamento do software. No futuro mais funcionalidades de decisão poderão ser embutidas, como verificação de preços e localização, construindo uma lógica mais econômica para o usuário.

4.2.5. AGENTE VOIP

O agente é o módulo que fornecerá toda a estrutura SIP necessária para o projeto, por isso pode ser considerado a parte mais complexa do sistema. Sua função será enviar, receber e tratar toda a sinalização do protocolo SIP, além de encaminhar o áudio de uma chamada VoIP.

Para cumprir todas as suas funcionalidades, o agente também deverá possuir um serviço a ser inicializado junto com o *daemon*. Este serviço deverá aguardar pelo recebimento de novas chamadas e registrar a aplicação no proxy VoIP. Vale lembrar que este registro não tem conexão com o método “registrar” do *web service,* o primeiro serve para inserir um registro do cliente no banco do servidor, este segundo, serve como autenticação no servidor SIP e está especificado na RFC do protocolo, a de número 3261.

Embutido com o agente é necessário estarem presentes todos os recursos necessários para uma chamada VoIP: métodos do protocolo (*register*, *invite*, *ack*, *bye*, etc), *codecs* para compressão de áudio (g729, g711, etc), fluxos de sinalização, camada de transporte (UDP, TCP) entre outros. Para permitir maiores cuidados no software em si, e não com a estrutura do protocolo, deverá ser utilizada alguma biblioteca *open source* que ofereça estes recursos de baixo nível.

Para disponibilizar estes componentes, há dois softwares interessantes, o PJSIP implementa uma pilha SIP, disponibilizando o núcleo do protocolo. Algumas vantagens suas são o desempenho considerável e a grande portabilidade, já que este software está disponível para a maioria das arquiteturas. Mais completo que ele, existe o PJSUA, que é um *User Agent* para VoIP que utiliza a pilha implementada pelo PJSIP. Com este último pode ser alcançada uma abstração maior das camadas de baixo nível, pois ele já possui funcionalidades mais complexas como gerenciamento de contas e fluxos de chamadas.

O conjunto do agente deverá ser formado por uma junção da biblioteca a ser usada com o serviço citado anteriormente, com isso o desenvolvimento

deste módulo poderá ser reduzido ao encaixe destes componentes somado a alguns tratamentos para as funções de baixo nível.

4.3. CONSIDERAÇÕES FINAIS

O objetivo estrutural do software é que, apesar de sua modularidade, todos estes componentes tenham um papel bem definido e possam se comunicar de uma maneira organizada, utilizando corretamente as alternativas fornecidas pelo sistema Android para prover o serviço da maneira mais adequada. Essa estrutura inicial proposta foi embasada no estudo teórico dos temas relacionados. Na etapa de implementação serão abordadas novas possibilidades e alguns empecilhos encontrados no decorrer da criação do protótipo, tais acontecimentos só serão possíveis de se analisar após os primeiros experimentos e testes de conceito.

5. IMPLEMENTAÇÃO DA ARQUITETURA

A modelagem mostrada no capítulo anterior traçou uma base para ser construído um protótipo da aplicação. A partir dela será montada uma estrutura capaz de comprovar o funcionamento da arquitetura proposta e validar o projeto. No final da fase de implementação deve-se obter um software capaz de atender as funcionalidades requeridas e tornar-se um produto.

5.1. CONSTRUÇÃO DO PROTÓTIPO

Um dos fatores críticos desse projeto é o seu tempo de desenvolvimento, o qual precisa ser bem aproveitado. Como já citado anteriormente, há componentes de baixo nível, como a implementação do protocolo, e componentes de caráter visual, como as telas necessárias, que fogem um pouco do foco do estudo. Para estes componentes existem muitas estruturas já prontas que poderiam contribuir significativamente para a execução do modelo proposto. Pensando em otimizar o aproveitamento deste tempo para obter um protótipo funcional e ao mesmo tempo conseguir dar atenção a todas as estruturas necessárias, decidiu-se estudar a utilização de alguns softwares analisados no estudo.

A análise focou-se inicialmente em uma maneira inteligente de se utilizar o *User Agent* PJSUA. Este software implementa a pilha SIP PJSIP e proveria muitas funcionalidades necessárias ao desenvolvimento. Apesar de todas as facilidades oferecidas por este projeto *open source,* ele também traria alguns aspectos difíceis de ponderar. Um exemplo disso, seriam as dificuldades de portar a solução para um correto funcionamento no *framework* do Android.

Para contornar este problema foi considerada a utilização do software CSipSimple. Este software foi mencionado no estudo e trazia funcionalidades parecidas ao SipDroid. Um detalhe importante é que ele utiliza uma versão portada do PJSUA e além disso é *open source.* A partir disso foi realizada uma

análise profunda deste software e suas principais características. Abaixo pode ser visto um modelo superficial de sua arquitetura.

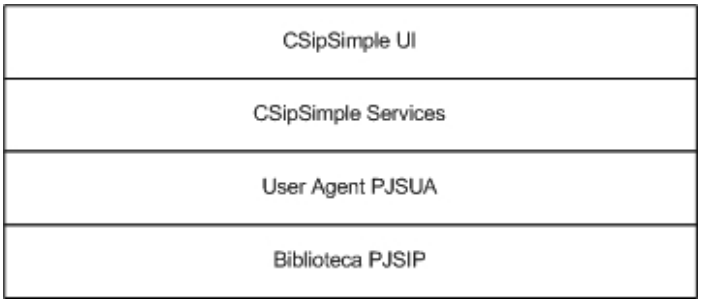


Figura 13: Arquitetura do CSipSimple.

A utilização deste software proveria duas estruturas propostas na modelagem do projeto, o discador e o agente VoIP, fornecendo uma base para o desenvolvimento das funcionalidades requeridas. Outro fator importante é que este aplicativo já fornece um modelo de comunicação entre estes componentes através de interfaces nos services, o que simplificaria o intercâmbio de contextos.

Do lado do servidor, em conjunto com o colega Marcelo Bublitz Anton, decidiu-se pela adoção do software Asterisk, com o papel de servidor SIP. Em termos de tráfego, esta decisão não afeta significativamente o cliente, porém com ela surgiu a possibilidade do próprio Asterisk fazer o controle de clientes *online* através dos métodos do protocolo SIP que são enviados periodicamente. Com este controle feito diretamente no servidor, através do Asterisk, pode ser descartada a construção do *daemon* modelado e do método “ativar” do *web service*.

5.1.1. CLIENTE

Para a adaptação do CSipSimple ao propósito desejado haverá a quebra do fluxo padrão do software em dois momentos principais. O primeiro deverá acontecer logo que o software for iniciado, tanto através de um Service como através de uma Activity, e registrará a aplicação no servidor. O segundo ponto

importante é no instante em que uma chamada for iniciada, para controle dos destinos possíveis.

O fluxo padrão da aplicação utiliza como base a Activity SipHome. Ela é declarada no AndroidManifest como a Activity inicial do aplicativo e carrega a interface básica, que é provida pela Activity Dialer e consiste em um teclado numérico e uma TextView para a realização de chamadas. A partir desta *view* inicial, além de realizar uma chamada, é possível acessar uma tela para o gerenciamento de contas VoIP e uma tela para configurações.

A tela de gerenciamento de contas traz diversas opções de configurações para se utilizar uma conta VoIP. A maioria dessas alternativas não são necessárias para prover um serviço fechado, sem integração com outras plataformas. Para o protótipo estar funcional serão necessárias apenas três informações referentes à conta em uso:

- o endereço do servidor, que virá fixo no código;

- o identificador do usuário, que será o número do telefone em uso;

- uma senha, que será utilizada uma *default.*

As opções presentes na tela de configurações também podem ser parcialmente desconsideradas, e a tela pode ser eliminada. O CSipSimple oferece muitas personalizações, as quais não possuem importância para este protótipo. A única informação que teve que ser aproveitada para a aplicação foi a utilização do codec de compreensão de áudio GSM durante as chamadas VoIP, os demais não serão suportados.

A primeira estrutura a ser incluída do zero no projeto foi o código responsável por prover os métodos necessários para realizar requisições ao *web service*. Para isso foi criada uma classe chamada RestClient, a qual possui todos os componentes para montar a requisição, adicionar parâmetros e realizar um POST ou um GET. Na figura 14 encontra-se o método que recebe um *request* pronto e a url do servidor e executa o comando.

```
119    private void executeRequest(HttpUriRequest request, String url) {
120        HttpClient client = new DefaultHttpClient();
121        HttpResponse httpResponse;
122
123        try {
124            httpResponse = client.execute(request);
125            responseCode = httpResponse.getStatusLine().getStatusCode();
126            message = httpResponse.getStatusLine().getReasonPhrase();
127
128            HttpEntity entity = httpResponse.getEntity();
129
130            if (entity != null) {
131                InputStream instream = entity.getContent();
132                response = convertStreamToString(instream);
133                instream.close();
134            }
135         } catch (ClientProtocolException e)  {
136             client.getConnectionManager().shutdown();
137        } catch (IOException e) {
138            client.getConnectionManager().shutdown();
139        }
140    }
141
```

Figura 14: Método executeRequest da classe RestClient

Esta classe será utilizada fundamentalmente nos dois momentos citados onde é necessário desviar o fluxo de chamadas do CSipSimple para permitir o gerenciamento das ligações através do servidor. A utilização do protocolo REST fez com que não haja a necessidade de armazenar nenhum estado referente às consultas, permitindo uma simplicidade maior.

O registro da aplicação no banco do servidor se dará ao inicializar a aplicação. Um cuidado que deve ser tomado é que existem dois meios diferentes deste software começar a executar. A primeira se deve à escolha do usuário de abrir a aplicação através do menu principal do aparelho, que trará o fluxo convencional e a activity SipHome será chamada, o tratamento do registro pode ser feito no *onCreate* desta classe. A segunda ocorre quando o aparelho é inicializado e a aplicação vai realizar uma auto-execução, este caso é tratado por um Broadcast Receiver do software e será abordado mais para frente.

Para efetuar o registro no servidor será necessário obter o identificador único do aparelho. Na modelagem foi decidido que a identificação dos usuários se daria pelo número do telefone, para isso é necessário solicitar ao sistema

esta informação. A figura 15 mostra o método que instancia um gerenciador de configurações do telefone e recebe o número do mesmo.

```
342    private String getMyPhoneNumber() {
343        TelephonyManager telephonyManager =
344            (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
345        String mPhoneNumber = telephonyManager.getLine1Number();
346        Log.e(SipHome.LOG_TAG, "getMyPhoneNumber: " + mPhoneNumber);
347        return mPhoneNumber;
348    }
```

Figura 15: Método getMyPhoneNumber.

O código mostrado acima será chamado pelo método que realiza o registro no servidor. Este método monta uma requisição POST utilizando como parâmetro o número adquirido com a chamada de getMyPhoneNumber *e a* submete ao servidor. Seu código pode ser visto abaixo:

```
358    private void serverRegister() {
359
360        Log.e(SipHome.LOG_TAG, "serverRegister invoked");
361        RestClient client = new RestClient(SERVER_URL_REGISTRO);
362        client.addParam("numero", getMyPhoneNumber());
363        try {
364            client.execute(RequestMethod.POST);
365        } catch (Exception e) {
366            e.printStackTrace();
367        }
368    }
```

Figura 16: Método serverRegister.

Uma vez feito este procedimento, o servidor possui as informações necessárias para controlar os clientes *online* e o fluxo da aplicação pode ser continuado. A activity SipHome possui ainda alguns tratamentos relativos à interface do dialer e o gerenciamento de algumas configurações e por fim, no seu método onResume ela cria o serviço principal da aplicação.

O serviço SipService é o núcleo do software. Ele é executado em uma thread própria e necessita estar em funcionamento nos momentos de tratamento de chamadas e alteração do status da aplicação. Nele são feitos os controles da pilha de software PJSIP, através do serviço PjSipService, e o gerenciamento de contas online. Durante toda a sua execução, o SipService

administra a autenticação da conta em uso com o servidor SIP e permite os outros componentes utilizarem o seu status através de uma interface. É através desta autenticação em nível de protocolo SIP que é possível o *agente* do cliente realizar e receber chamadas VoIP.

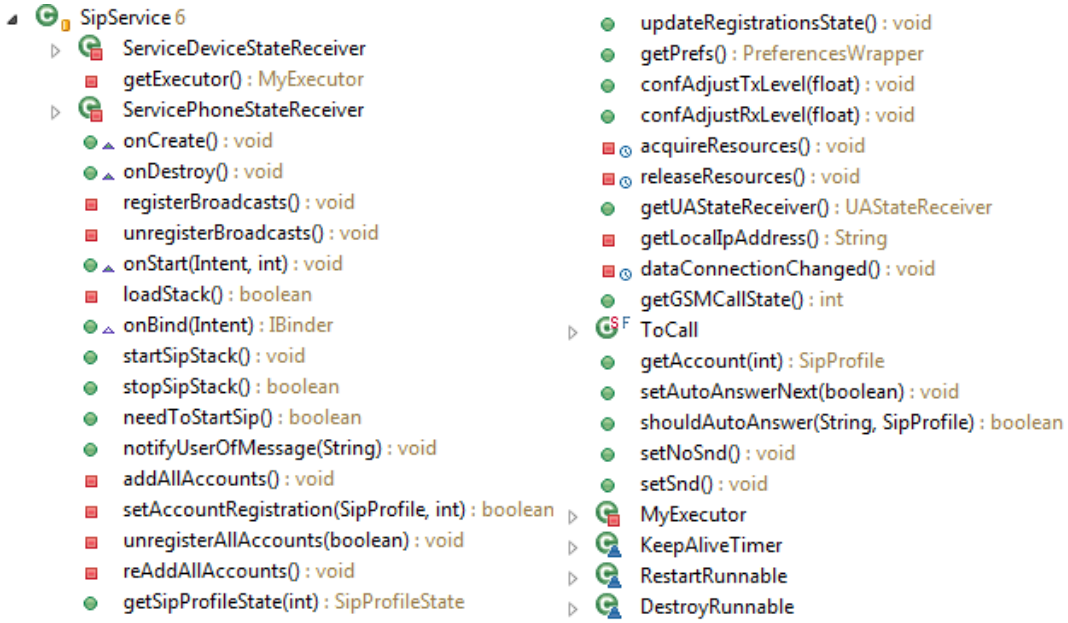


Figura 17: Componentes da classe SipService.

A complexidade deste Service é bastante alta, porém, devido à sua alta modularidade, não são necessárias alterações em seu código. Seu fluxo será mantido, e seu comportamento continuará semelhante ao *daemon* sugerido no processo de modelagem, fazendo o gerenciamento de autenticação, porém com recursos mais avançados.

Para o gerenciamento de eventos relevantes ao sistema existe um Broadcast Receiver que centraliza uma série de *listeners,* entre eles estão o de captura de inicialização do sistema e o de alteração na conectividade com a internet. O primeiro tem o papel de realizar o início da aplicação quando o aparelho é ligado. Como já citado, este ponto também é relevante por ser a forma alternativa de início da aplicação e os cuidados com o registro no servidor também se aplicam a este momento. O segundo detecta alterações na

conectividade do aparelho com a internet e informa essas alterações ao SipService através da geração de uma intent para o serviço.

Ao que diz respeito à execução passiva do software, ou seja, enquanto ele apenas aguarda o recebimento de chamadas, os aspectos fundamentais foram levantados, agora restam os tratamentos necessários para o usuário conseguir realizar as chamadas de forma transparente. Para isso foram necessárias alterações no fluxo do *dialer.*

A classe *dialer,* responsável por prover a interface para a classe SipHome, chama o método placeCall no momento que o botão discar é pressionado. Este método realizará a consulta de qual será o encaminhamento a ser dado para a chamada, através da consulta de contas ativas naquele momento. Neste ponto é necessário a utilização de mais um controle, a conferência se o número de destino também está utilizando o cliente e está disponível no momento. Para este controle foi inserida a chamada do método serverAuthorizeCall, como pode ser visto no código da figura 18.

```
474    private void placeCall(int view) {
475        if (service == null) {
476            return;
477        }
478        String toCall = "";
479        Integer accountToUse = USE_GSM;
480
481        if (view == DIGIT_VIEW) {
482            toCall = PhoneNumberUtils.
483                stripSeparators(digits.getText().toString());
484
485            SipProfile acc = accountChooserButton.getSelectedAccount();
486            //testa se ha conta voip e o server autorizou
487            if (acc != null && serverAuthorizeCall(toCall)) {
488                accountToUse = acc.id;
489            }
490        } else {
491            ToCall objToCall = sipTextUri.getValue();
492            if (objToCall == null) {
493                return;
494            }
495            toCall = objToCall.getCallee();
496            if(objToCall.getAccountId() != null
497                    && serverAuthorizeCall(toCall)) {
498                accountToUse = objToCall.getAccountId();
499            }
500        }
```

Figura 18: Trecho do método placeNewCall da classe Dialer.

O método em questão é a implementação da chamada ao segundo método do *web service,* o “ligar”. Após esta requisição o retorno esperado pode ser uma *String* “on” ou “off”, identificando se o destinatário está ou não registrado no servidor. A Figura 19 mostra o código do método.

```
450    private boolean serverAuthorizeCall(String destination) {
451
452        Log.e(SipHome.LOG_TAG, "serverAuthorizeCall destination: "
453                + destination);
454        RestClient client = new RestClient(SERVER_URL_AUTHORIZE
455                + destination);
456        try {
457            client.execute(RequestMethod.GET);
458        } catch (Exception e) {
459            e.printStackTrace();
460        }
461        String response = client.getResponse();
462
463        Log.e(SipHome.LOG_TAG, "serverAuthorizeCall response: "
464                + response);
465        if (response.compareTo(ONLINE)==0) {
466            return true;
467        } else return false;
468    }
```

Figura 19: Método serverAuthorizeCall da classe Dialer.

Feitas estas alterações, as chamadas só ocorrerão por VoIP no momento em que tanto o originador quanto o destinatário estiverem conectados na internet e autenticados no servidor, dispensando a utilização de uma terminação da rede móvel para a rede VoIP e vice-versa. As demais alterações tem caráter de otimização do código para a remoção de funcionalidades não necessárias e melhora da interação com o usuário para ficar mais claro o funcionamento deste novo software que pode ser chamado de CustomCSipSimple.

5.2. TESTES FUNCIONAIS

Com as estruturas do cliente e do servidor funcionais foi possível realizar os primeiros testes para verificar a corretude da aplicação. O aplicativo beta gerado possuiu como servidor a estrutura criada por (ANTON, 2011) e

publicada no endereço http://voip.mbanton.net/', portanto os métodos do *web service* possuiam as URIs "http://voip.mbanton.net/registrar/ e "http://voip.mbanton.net/ligar/".

A aplicação *mobile* foi instalada em quatro dispositivos diferentes, um *smartphone* Motorola Milestone, um *smartphone* Motorola Atrix, um *smartphone* Samsung Galaxy S e um *tablet* Samsung Galaxy Tab sem módulo de telefonia, além do emulador do SDK do Android que foi utilizado constantemente durante a fase de desenvolvimento.

O primeiro problema observado foi com a captura do número do telefone do cliente. Em alguns casos, dependendo da operadora, não era possível esta aquisição. Com uma breve pesquisa, percebeu-se a escolha feita por algumas operadoras de não disponibilizar o número de telefone no *SIM Card* e nem mesmo no aparelho. Uma resolução temporária para este problema foi a inserção do número no chip de forma manual, porém existem diversas possibilidades para contornar este problema, sendo a mais trivial solicitar ao usuário o número do telefone e realizar a confirmação através de uma mensagem de texto.

Com o problema da obtenção do número contornado, foi possível realizar o registro no servidor através da chamada "registrar". O funcionamento dos métodos já havia sido validado na fase de desenvolvimento, portanto não houve problemas de comunicação entre o *web service* e o dispositivo móvel. O registro dos aparelhos no servidor foi acompanhado para garantir a validade das informações no banco. Terminado o registro na estrutura, o software já busca automaticamente a comunicação com o servidor VoIP e o registro via protocolo SIP. A partir disso será mostrada uma mensagem na barra de notificações do aparelho informando que aquela conta está *online*.

Com dois aparelhos utilizando o software, um do autor e o outro do Marcelo Bublitz Anton, e registrados no servidor, já é possível testar a aplicação. Ao abrir o discador do CustomCSipSimple e digitar como destino o número do Marcelo a ligação foi efetuada utilizando a estrutura criada neste

projeto. Para garantir que a chamada não saiu pela rede de telefonia móvel, foram acompanhadas a entrada e a saída de pacotes no módulo servidor.

Posteriormente foram realizados testes com a conexão com a internet interrompida, para garantir que a opção por ligação convencional neste caso estivesse com um comportamento correto. Também foram realizadas chamadas com o módulo de telefonia do aparelho desativado, o que garante a realização de chamadas utilizando a rede de dados e consequentemente a estrutura de nossa rede VoIP.

Com a realização destes testes de funcionamento básico, foi possível validar o funcionamento do protótipo. Nesta fase final do projeto, seus recursos ainda são básicos, porém são suficientes para justificar a sua idealização e modelagem. Novas funcionalidades poderão ser facilmente agregadas ao cliente e sua inteligência de decisão poderá ser expandida sem maiores problemas.

5.3. PROTÓTIPO ALCANÇADO

Ao final da fase de prototipação alcançou-se um aplicativo capaz de exercer o papel para o qual foi planejado. Suas funcionalidades básicas, que seriam realizar chamadas através da rede VoIP quando esta rede estivesse disponível para os dois usuários e receber estas chamadas quando fosse necessário, podem ser realizadas sem problema algum.

A arquitetura traçada durante o estudo obteve algumas modificações na fase de desenvolvimento. Essas alterações se deveram principalmente à otimização do tempo para criar o protótipo e pelo descobrimento de novas tecnologias durante a construção do software, como foi o caso da utilização do software Asterisk do lado do servidor.

A opção de utilização do projeto CSipSimple trouxe inúmeras vantagens nas fases iniciais do desenvolvimento, como a abstração de questões referentes a gerenciamento de contas e interfaces. Em contrapartida em etapas

posteriores do desenvolvimento esta decisão apresentou dificultadores, como na parte de gerenciamento de codecs. O CSipSimple não possuía uma distribuição de responsabilidades bem definida entre a aplicação e a pilha SIP PJSIP.

Sua interface foi uma simplificação da já existente no software original, removendo aspectos desnecessários ao contexto do projeto. As possibilidades de personalização também foram abolidas, diminuindo as funcionalidades frente ao CSipSimple, porém mantendo-se dentro do escopo do projeto.

Por fim a aplicação mostrou-se estável e correta frente ao testes, possuindo alguns detalhes a serem observados em caso de necessidade de tornar-se um produto viável. Estes pontos em aberto são muito mais pertinentes a questões de mercado, como arranjar alguma empresa com interesse de abrigar um servidor para este cliente, do que de funcionamento do produto.

6. CONCLUSÃO

O estudo inicial da tecnologia Android possibilitou o entendimento das estruturas básicas envolvidas no sistema e de como elas se comunicam. A partir deste fato pode-se entender a arquitetura de uma aplicação e as grandes possibilidades de desenvolvimento para a plataforma. Foi então que a percepção de que realmente é possível desenvolver um componente, que interaja ou substitua uma peça padrão do sistema, como o discador, foi alcançada. Esta capacidade era pré-requisito básico para o desenvolvimento do projeto.

A partir do estudo sobre VoIP, esclareceram-se as estruturas de *backend* necessárias para que a aplicação desenvolvida para Android tivesse sua totalidade de funcionalidades trabalhando corretamente. Este estudo também teve sua importância na definição do protocolo VoIP a ser utilizado para efetivação das chamadas. Com ele foi possível atingir um entendimento mais claro do conjunto de recursos necessários para um software VoIP, antevendo as funcionalidades necessárias à aplicação.

O planejamento da estrutura geral a ser utilizada foi elaborado a partir de conversas constantes com o Marcelo Bublitz Anton, que desenvolveu a parte de *backend* necessária em (ANTON, 2011). Juntamente com ele foram definidas as melhores estruturas, os softwares mais adequados para se utilizar, e o modelo de comunicação e topologia que foi empregado.

Os componentes que não implicaram configurações ou alterações no servidor, como estrutura do discador, modelo de programação, cliente VoIP, entre outros, foram definidos unicamente a partir do estudo publicado neste trabalho. Foram levados em conta para estas decisões principalmente a simplicidade de implementação, a estabilidade do serviço e a experiência com o usuário.

A modelagem se baseou na construção de três módulos com uma divisão de responsabilidades bem definida entre eles. Essa independência de funcionamento criados nesta fase possibilitou uma liberdade maior na criação

do software, permitindo o mapeamento dos módulos para um software de terceiros e simplificando o processo de construção.

Na etapa de desenvolvimento a utilização do software CSipSimple possibilitou um melhor aproveitamento do tempo, fazendo com que diversos componentes fossem reaproveitados. Como já dito, essas vantagens ficaram bem claras nas fases iniciais do projeto, porém no decorrer do desenvolvimento surgiram problemas de configuração e compatibilização que poderiam ter sido evitados em caso de construção do protótipo a partir do zero, porém neste caso o resultado final teria que ser um modelo consideravelmente mais simples.

A partir dos testes da aplicação comprovou-se a possibilidade de criação de uma estrutura transparente de gerenciamento de chamadas para Android. O funcionamento da aplicação foi validado utilizando os dois principais cenários que devem ser encontrados: o primeiro quando todas as premissas de conectividade e registro no servidor são verdadeiros, o que resulta em uma ligação VoIP para VoIP, possivelmente sem custo algum, caso o cliente esteja usando algum ponto de acesso wi-fi; o segundo quando alguma das premissas não é verdadeira e a chamada seguirá um fluxo padrão utilizando a rede de telefonia móvel.

Em termos de funcionalidades, o software final apresenta os recursos necessários para uma ferramenta deste gênero. A tela do discador possibilita a realização de chamadas e os serviços que rodam em segundo plano permitem o recebimento de ligações. Em termos de configurações, ele possui os parâmetros necessários para seu funcionamento fixados no código. Para aprimoramento do aplicativo, podem ser melhoradas as regras de decisão para efetuar uma chamada VoIP. Nesta primeira versão, a regra é ligar sempre que possível utilizando a rede de dados.

Com o aprimoramento destas regras seria possível estabelecer padrões para efetuar uma ligação, aumentando substancialmente a importância da aplicação. Poderia ser possível, por exemplo, utilizar VoIP toda a vez que estiver utilizando uma rede *wireless,* e em caso de estar usando a rede 3G, utilizar VoIP apenas se não estiver em *roaming.*

As possibilidades de utilização deste projeto são significativas, e vão do seu aproveitamento como um serviço web, da sua utilização por empresas, e até o aproveitamento da ideia por operadoras de telefonia. Por exemplo, uma empresa privada poderia fornecer a seus colaboradores esta solução para ser utilizada juntamente com o *smartphone* corporativo. O gasto de manutenção de um servidor seria desprezível frente à economia com ligações que se tem utilizando a estrutura.

Por fim, poderia ser feito um estudo, analisando suas possibilidades de aproveitamento, sua real eficácia, a otimização de custos trazida por ele e todas as vantagens possíveis, mas o foco deste estudo, que seria provar a viabilidade técnica deste modelo, foi alcançado. Este trabalho poderá servir como base para futuras extensões do projeto, assim como a sua possível transformação em um produto com potencial no mercado.

REFERÊNCIAS

ANTON, Marcelo Bublitz. “Uma Proposta de Arquitetura VoIP para Smartphones com Sistema Operacional Android”. UFRGS, 2011.

BURK, Bill. “RESTful Java with JAX-RS”. O’Reilly, 2010

FIELDING, Roy Thomas. “Architectural Styles and the Design of Networkbased Software Architectures”. University of California, Irvine, USA, 2000.

GONÇALVES, Flavio E. “Building Telephony Systems with OpenSIPS 1.6”. Packt, 2010.

HASHIMI, Sayed and KOMATINENI, Satya and MACLEAN, Dave. “Pro Android 3”. Apress, 2010.

DAVIDSON, Jonathan and PETERS, James. “ Voice over IP Fundamentals”. Cisco Press, 2000.

LEE, Wei-Meng. “Beginning Android Application Development”. Wrox Programmer to Programmer, 2011.

LOMBARDO, John and MEDNIEKS, Zigurd and MEIKE, Blake and ROGERS, Rick. “Android Application Development”. O’Reilly, 2009.

MEIER, Reto. “Professional Android Application Development”. Wiley Publishing, 2009.

Web Mobile Magazine – DevMedia, Edição número 30, Ano 5.

Anatel - Disponível em: www.anatel.gov.br
Acessado em 29/06/2011

Android – Disponível em: www.android.com
Acessado em 04/06/2011

Android Developers - Disponível em: developer.android.com
Acessado em 04/06/2011

Android Source - Disponível em: source.android.com/
Acessado em 13/05/2011

Canalys – Disponível em: www.canalys.com
Acessado em: 03/11/2010

CSipSimple - Disponível em: code.google.com/p/csipsimple/
Acessado em 18/06/2011

Frost & Sullivan - Disponível em: www.frost.com
Acessado em 18/06/2011

Harris Interactive – Disponível em: www.harrisinteractive.com/
Acessado em 20/06/2011

IDC - Disponível em: *www.idc.com*
Acessado em 02/04/2011

IETF - Disponível em: www.ietf.org/rfc/rfc3261.txt
Acessado em 13/04/2011

Opensips - Disponível em: www.opensips.org
Acessado em 24/10/2010

PJSIP - Disponível em: www.pjsip.org/
Acessado em 27/05/2011

Sipdroid - Disponível em: sipdroid.org
Acessado em 11/11/2010

Skype - Disponível em: www.skype.com
Acessado em 11/11/2010

Wikipedia – Disponível em: pt.wikipedia.org
Acessado em 18/06/2011

WALL, Dick. “Introduction to Android Arquitecture”. 2008,
Disponível em: http://www.youtube.com/watch?v=D_pGLFyqPRo

WALL, Dick. “How to Develop Android Applications”. 2008,
Disponível em: http://www.youtube.com/watch?v=OdEcuXPhV6E