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

UMA PROPOSTA DE ARQUITETURA VOIP PARA SMARTPHONES COM SISTEMA OPERACIONAL ANDROID

MARCELO BUBLITZ ANTON

PORTO ALEGRE
JULHO/2011

MARCELO BUBLITZ ANTON

UMA PROPOSTA DE ARQUITETURA VOIP PARA SMARTPHONES COM SISTEMA OPERACIONAL 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/2011

AGRADECIMENTOS

À minha família pela criação que tive e apoio incondicional na minha caminhada até aqui, em especial ao meu pai que me mostrou o caminho da minha profissão; à minha namorada pelo apoio e companheirismo em todos os momentos; ao colega e amigo Guilherme de Moraes Uzejka, pela parceria no projeto e motivação constante; à UFRGS e em especial ao Instituto de Informática, pelo ensino de qualidade prestado; ao Prof. Dr. Cláudio Resin Geyer pela orientação e auxílio no projeto; e a todos que de alguma forma contribuem disseminando conhecimento através do mundo do Software Livre.

“O sucesso é uma consequência e não um objetivo”
(Gustave Flaubert)

RESUMO

Este trabalho tem por objetivo estudar tecnologias necessárias para propor uma arquitetura Voip que permita o uso dessas tecnologias em Smartphones com o sistema operacional Android. Primeiramente serão estudadas as possibilidades de Voip já existentes para essa plataforma, e após será proposta uma nova abordagem com alguns diferenciais. O principal objetivo desta nova abordagem é ser uma ferramenta inteligente e transparente para o usuário. Deve permitir ao mesmo utilizar um software em seu telefone que opte entre telefonia convencional e Voip. O segundo deve ser opção somente quando o aparelho tiver rede de dados disponível. Este trabalho será realizado em parceria com o trabalho de conclusão de curso do aluno Guilherme de Moraes Uzejka. Guilherme irá planejar o projeto no lado do cliente, que rodará no Android. Já o presente trabalho versará sobre a parte do servidor, que proverá as estruturas necessárias para que o cliente funcione.

Palavras chave: Smartphones, Android, SIP, Voip, Web Services, REST, 3G, Skype.

A PROPOSED VOIP ARCHITECTURE FOR SMARTPHONES WITH ANDROID OPERATING SYSTEM

ABSTRACT

This work aims to study technologies needed to propose an architecture that enables Voip using these technologies in smart phones with the Android operating system. First will be studied the possibilities of existing VoIP platform for this, and after it is proposed a new approach with some differences. The main objective of this new approach is to be an intelligent and transparent to the user. It should allow one to use the same software on your phone you choose between conventional telephony and VoIP. The second should be an option only when the device has data network available. This work will be done in partnership with the completion of course work the student Uzejka Guilherme de Moraes (2011). William will plan the project on the client side, which will run on Android. Since this paper will focus on the server part, which will provide the necessary structures for the client to work.

Keywords: Smartphones, Android, SIP, Voip, Web Services, REST, 3G, Skype.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

VOIP	Voice Over Internet Protocol
WIFI	wireless standard for connecting electronic devices
3G	3rd generation mobile telecommunications
SMS	Short Message Service
XMPP	Extensible Messaging and Presence Protocol
XML	Extensible Markup Language
RTP	Real-time Transport Protocol
P2P	Peer-to-Peer
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
HTTP	Hypertext Transfer Protocol
SOAP	Simple Object Access Protocol
RPC	Remote procedure call
WSDL	Web Service Definition Language
REST	Representational State Transfer
GSM	Global System for Mobile
RFC	Request for Comments
SO	Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

O segmento de Voip (Voice Over Internet Protocol, ou Voz sobre IP em português) é um segmento consolidado no mercado. Com a popularização dos acessos à internet através de Banda Larga, cada vez mais esse meio de originar e receber chamadas de voz, que não utiliza a rede de telefonia convencional, será utilizado.

Outra tecnologia que vem despontando nos últimos anos, e já tida por vários especialistas como o futuro da computação, é os dispositivos móveis, em especial os SmartPhones. Cada vez mais celulares são equipados com inúmeras funcionalidades e extrapolam a barreira das simples ligações telefônicas. Aparelhos desse tipo já costumam vir com conectividade avançada, como conexões Wifi e/ou 3G.

Da união desses dois nichos surge a proposta desse trabalho. Levar o Voip para dentro dos SmartPhones e permitir aos seus usuários utilizarem o mesmo de uma forma transparente, sem necessidades de complexas configurações, ou contatos inicias e ajustes com a pessoa para a qual se deseja efetuar a ligação.

Para isso, será necessário um serviço que conheça os clientes conectados naquele momento e saiba exatamente se pode ou não completar uma ligação via Voip para o mesmo. Esse serviço deve ser capaz de registrar o status de um cliente, e de informar esse mesmo status para outros dispositivos que desejam contato com ele.

Este trabalho será complementado (e ao mesmo tempo será um complemento) do TCC do também aluno de graduação Guilherme de Moraes Uzejka. (2011). O trabalho do Guilherme será desenvolver o “Client” para SmartPhones que rodem o sistema operacional Android, desenvolvido pela Google. (ANDROID, 2011). Esse cliente será capaz de utilizar todos os serviços providos pela plataforma, e completar ligações através da mesma.

1.1 MOTIVAÇÃO

A motivação para este projeto surge no fato da tecnologia Voip estar sendo amplamente utilizada nos mais variados nichos, inclusive mais recentemente em SmartPhones. Neste tipo de plataforma, carecem ainda aplicações que objetivem a praticidade para o usuário final, que não necessitem de configurações e esforços adicionais.

Como segunda parte da motivação, surgem as afinidades do autor. Sua experiência profissional e conhecimento das tecnologias Voip, e seu atual interesse pela plataforma Android. Os interesses em comum do aluno Guilherme de Moraes Uzejka (2011), e o fato da arquitetura como um todo ser suficientemente grande para um Trabalho de Conclusão, fazem com que a união e complementaridade dos dois trabalhos seja ideal.

1.2 OBJETIVO

O objetivo do trabalho é primeiramente estudar o panorama atual do Voip nos Smartphones. Após o estudo, serão verificados quais tecnologias podem ser usadas para prover esta nova arquitetura. A mesma deve permitir que aparelhos com Android se comuniquem por voz usando dados de uma forma simples e transparente para o usuário. O objetivo deste trabalho é prover a arquitetura servidora, e já permitir um acoplamento simples para o cliente que será estudado e desenvolvido pelo aluno Guilherme de Moraes Uzejka (2011).

Como ápice dos dois trabalhos, é objetivado que a arquitetura seja funcional e permita que um protótipo rode em aparelhos reais, e a parte do servidor rode em uma máquina remota provendo toda a estrutura para que uma chamada de voz seja realizada, utilizando rede de dados, seja ela 3G ou Wifi.

1.3 ORGANIZAÇÃO DO TEXTO

O texto será organizado em seis capítulos, com os seguintes objetivos:

- **Capítulo 2:** apresentar o “estado da arte” das tecnologias Voip nos Smartphones. Serão analisados quais protocolos e quais softwares já existem e quais as principais diferenças entre eles. O foco do capítulo será no sistema Android.

- **Capítulo 3:** apresentar rapidamente um breve estudo sobre as formas de se prover Web Services, serviço este que será necessário no desenvolvimento do modelo.

- **Capítulo 4:** a partir do estudo realizado nos capítulos anteriores, será desenvolvido o modelo que do qual se espera obter as funcionalidades objetivadas. Serão feitos comentários sobre cada tecnologia utilizada.

- **Capítulo 5:** programar e demonstrar um protótipo baseado no modelo construído no capítulo 4. O mesmo deverá rodar em uma máquina real, conectada remotamente na Internet.

- **Capítulo 6:** conclusões sobre as tecnologias estudadas e sobre o modelo e protótipo atingido.

2 SOLUÇÕES EXISTENTES

Voz sobre IP é o roteamento de conversação humana usando a Internet ou qualquer outra rede de computadores baseada no Protocolo de Internet. Assim a transmissão de voz se torna mais um dos serviços suportados pela rede de dados. Também é conhecida como: VoIP (Voice over Internet Protocol), telefonia IP, telefonia Internet, telefonia em banda larga e voz sobre banda larga (WIKIPEDIA VOIP, 2010).

Com o Voip se difundido largamente, hoje no segmento corporativo, grande parte das empresas utilizam serviços deste gênero de uma forma ou de outra. Os principais atrativos deste tipo de solução são a excelente redução de custos e a flexibilidade das soluções que podem ser montadas.

Já no público residencial, o Voip também já traz grandes atrativos, principalmente pela redução de custos, ponto que sempre pesa para o usuário final. Soluções de software que rodam diretamente no PC dos usuários e permitem ligações a baixo custo, ou até mesmo gratuitas, atingem uma grande massa de pessoas. Outros tipo de serviços, como por exemplo, cartões telefônicos, terminais telefônicos ATA, e sites de venda de minutos, também se difundem largamente. Assim, levam o Voip até mesmo para pessoas que não tem acesso a Internet e não tem familiaridade com computadores.

Neste capítulo são apresentados os principais protocolos utilizados hoje em dia, para ligações de Voz sobre Ip, e serão comentados os principais softwares que exploram cada protocolo, focando logicamente nos dispositivos móveis. O Android foi usado como plataforma alvo para este breve pesquisa, portanto, todos os softwares aqui citados possuem versão para este SO de SmartPhones.

2.1 PROTOCOLO SKYPE

O Skype é notoriamente o software de maior sucesso de Voip. O que começou como um pequeno programa para a plataforma Windows, hoje é uma imensa comunidade de usuários que acessam este serviço das mais variadas plataformas, e utilizam o mesmo para se comunicar por voz.

O Skype também dá nome ao seu protocolo, visto que o mesmo desde o início possui uma implementação própria logo acima da camada de rede TCP. O protocolo é proprietário e é baseado na arquitetura peer-to-peer, um dos maiores trunfos do Skype. A rede Skype não se comunica com nenhuma outra rede VoIP sem a devida licença da empresa Skype (SKYPE, 2010).

A arquitetura básica do protocolo é baseada no conceito de um diretório que é descentralizado e distribuído entre os clientes ou nós. A rede contém três tipos de entidades: supernós, nodos normais, e o servidor de login. Cada cliente mantém um cache de host com o endereço IP e números de porta de supernós acessíveis. Qualquer cliente com boa largura de banda, sem firewall e capacidade de processamento pode se tornar um Supernó. Para clientes atrás de firewalls ou NAT simétrico, somente resta serem nodos normais. Isso acaba colocando um peso extra para aqueles que se conectam a Internet sem NAT, pois, o Skype pode utilizar os seus computadores e sua própria conexão com a rede para transmitir informações para outros utilizadores do serviço (WIKIPEDIA SKYPE, 2010).

As principais funções de um cliente do Skype são: login, pesquisa do usuário, chamadas de início e fim, meios de transferência e mensagens de presença. Já vemos claramente que o protocolo disponibiliza funções interessantes, como possibilidade de buscar por usuários e, uma base única de usuários, funcionalidades essas que veremos mais a frente fazem falta nos outros protocolos.

Recentemente no dia 10/05/11 foi anunciado a compra pela Microsoft da empresa Skype do grupo de investidores que detinha o controle da mesma pelo valor de US$ 8,5 bilhões. "O Skype é um serviço fenomenal que é amado por milhões de pessoas no mundo. Juntos, nós criaremos o futuro das comunicações em tempo real, fazendo com que as pessoas se conectem facilmente a sua família, amigos, clientes, colegas de trabalho de qualquer parte do mundo", disse Steve Ballmer, CEO da Microsoft, ao comentar a aquisição (BLOG TECNOLOGIA, 2011).

2.1.1 CLIENTE OFICIAL

A própria Skype Inc. No momento que desenvolveu o protocolo, lançou o primeiro cliente para a plataforma Windows. A API do Skype está disponível para que

outras versões sejam criadas, porém toda e qualquer implementação deve seguir rigorosas regras de licenciamento do protocolo.

O cliente oficial possui até o presente momento versões para as seguintes plataformas convencionais: Windows, Linux, Mac. Para as plataformas móveis: Android, IOS (Iphone, Ipad, IPod Touch), Symbiam (celulares da Nokia e Sony Ericsson), Maemo (celulares da Nokia). Está disponível também para algumas linhas de televisões da Samsung e Panasonic e ainda existem modelos de hardwares sendo lançados pela própria empresa, como o SkypePhone3 (somente em poucos países do mundo), e telefones Wifi.

A implementação para Android data como disponível desde janeiro de 2009, sendo que inclusive a versão para Android saiu antes que a para o IOS da Apple. Porém até mesmo hoje[1], ainda não há total estabilidade do produto. Inclusive o mesmo se mostra incompatível com o Samsung Galaxy S, que é um dos modelos mais atuais de SmartPhones disponíveis no mercado brasileiro. Uma das telas rodando no Android pode ser visto na Figura 1.

Funcionalidades prometidas pela versão para Android do cliente, de acordo com o site do produto (SKYPE FOR ANDROID, 2011) [2]:

- **Instant messaging (IM)** - Mensagens instantâneas;
- **Group IM** - Mensagem em grupo;
- **Skype-to-Skype calling** - Chamadas entre clientes Skype;
- **Calling phones and mobiles** - Chamadas para telefones fixos e móveis;
- **Receiving Skype calls and calls to Online Numbers** - Receber chamadas de Skype e de número comprados;
- **Participating in conference calls** - Participar de conferências;

1 Esta análise foi realizada no mês de Dezembro de 2010, portanto pode não se refletir nos dias de hoje.
2 Tradução para o português não literal feita pelo autor

- **Synchronizing with Android phone book** - Sincronismo da lista de contatos com a lista do Android;

- **Importing from native phonebook** - Importar os contatos da lista de contatos nativas do aparelho;

- **Call forwarding** - Encaminhamento de chamadas;

- **Showing favorite contacts** - Mostrar contatos favoritos;

- **Contact list filtering via groups** - Filtro de contatos por grupo.

Funcionalidades do protocolo ainda não suportadas na plataforma Android:

- **Skype video calling** - Chamada de vídeo;

- **Voicemail** - Caixa postal;

- **Skype SMS** - Mensagem Instantânea do Skype;

- **File transfer** - Transferência de arquivos.

Figura 1: Cliente Skype para Android

2.1.2 CLIENTES ALTERNATIVOS

Na plataforma Android, dois software em suas primeiras versões suportavam comunicação usando o protocolo Skype. O Fring, que será demonstrado mais adiante, suportava este protocolo até Julho de 2010. Porém, uma demanda gerada pelo Fring, em sua implementação de vídeo conferência para Iphone, acabou por iniciar um atrito entre a Skype e a Fring, o que culminou com a remoção total do suporte ao Skype no software.[3]

Outro caso semelhante é o Nimbuzz, que também será demonstrado mais adiante, que teve o suporte ao Skype encerrado em Outubro de 2010. Os motivos são os mesmos, ou seja, desentendimentos entre as empresas envolvidas[4].

3 Informação obtida em: http://www.adrenaline.com.br/telecom/noticias/5561/fring-desativa-o-servico-de-skype.html

4 Informação obtida em: http://brasil.blog.nimbuzz.com/2010/10/25/o-que-a-recisao-entre-o-skype-e-a-nimbuzz-significa-para-voce/

2.2 PROTOCOLO XMPP

O protocolo XMPP teve suas origens do protocolo jabber, criado no ano de 1998 por Jeremie Miller, como um software Open Source. Este protocolo, que tinha como objetivo inicial ser um projeto de servidor open source (jabberd) foi que deu a origem às especificações finais do XMPP, na RFC 3920 e RFC 3921 (WIKIPEDIA XMPP, 2010).

XMPP é a abreviação de Extensible Messaging and Presence Protocol e é um protocolo totalmente aberto de comunicações para middleware orientado a mensagens baseadas em XML. O protocolo originalmente servia para comunicações de mensagens instantâneas (IM), informação de presença e manutenção de lista de contatos no lado do servidor. Possui também suporte a transferência de arquivos. Sempre foi pensado de uma forma extensível e hoje também encontra aplicação em VOIP.

Diferentemente da maioria dos protocolos de mensagens instantâneas, como o implementado pelo skype, por exemplo, o XMPP usa um sistema aberto de desenvolvimento, o que permite a qualquer um implementar um serviço XMPP que possa interoperar com implementações de outras organizações. Muitas dessa implementações são distribuídas de forma gratuita e/ou livre.

O XMPP já foi usado por mais de 10 milhões de pessoas em todo mundo, nos mais variados softwares, incluindo o Google Talk, da empresa Google que é uma das grande apoiadoras dos projetos relacionados a este protocolo.

2.2.1 EXTENSÃO JINGLE

O Jingle é uma extensão para o protocolo XMPP que implementa peer-to-peer (P2P) para controle de sessão (sinalização) para multimídia, como por exemplo em voz sobre IP (VoIP) ou comunicações de videoconferência. Foi criado pelo Google e a XMPP Standards Foundation. As transmissões multimídia são encapsuladas usando o Real-time Transport Protocol (RTP). Se necessário, a comunicação por NAT é assistida usando Iteractive Connectivity Establishment (ICE). (WIKIPEDIA JINGLE, 2010).

A especificação final do Jingle foi finalizada no final de 2009, portanto, algumas implementações feitas antes desta data ainda precisam de alterações para total compatibilidade com o protocolo.

A biblioteca libjingle, usado pelo Google Talk para implementar Jingle, foi liberada para o público sob uma licença BSD. O fato da implementação de um cliente potencialmente forte como o Google Talk usar o Jingle, mostra que o protocolo, apesar de relativamente novo, tem grande chance de se popularizar. O fato de uma negociação Jingle resultar em uma sessão de Real-time Transport Protocol (RTP), faz essa sessão compatível com as de sinalizações usando protocolo SIP. Além disso, a semântica de sinalização Jingle foi projetado para ser compatível com SIP e do Session Description Protocol (SDP), tornando clara a implementação do protocolo em gateways já existentes para SIP.

2.2.1.1 CLIENTE NIMBUZZ

O Nimbuzz permite efetuar ligações, conversas por chat, envio de mensagens e compartilhamento de arquivos em tempo real, e sem nenhum custo. Você pode juntar todas os seus amigos do Skype, MSN, Yahoo, ICQ, AIM, Google Talk e outros. (NIMBUZZ, 2010).

O Nimbuzz possui um servidor próprio, onde é feito o cadastro criando uma conta, e após vincula as suas contas de outros softwares de mensagens à do Nimbuzz. Essa funcionalidade, suportada por muitos dos clientes ditos multi-protocolos é extremamente interessante, pois permite ao usuário ter uma única conta, e falar com usuários de vários protocolos.

O Nimbuzz possui clientes para as mais variadas plataformas. Softwares para Windows e MacOs estão disponíveis, e uma versão Web permite o acesso por qualquer computador com browser instalado. Versões mobile existem para as plataformas IOS, Symbian, e Android, atingindo mais de 1000 modelos de aparelhos.

A versão para Android, que pode ser vista na Figura 2, suporta todas as funcionalidades da plataforma, permitindo rodar no celular um agregador de protocolos de mensagens. As funcionalidades de Voip estão presentes e existe um

discar dentro do próprio programa que permite efetuar chamados telefônicas, através do NimbuzzOut (NIMBUZZ OUT, 2010). Este serviço permite a compra de créditos para efetuar chamadas de voz para qualquer lugar do mundo por tarifas baixas.

Figura 2: Cliente Nimbuzz para Android

2.2.1.2 CLIENTE TANGO

Tango é um software recente, lançado em 2010, que vem com uma idéia interessante. Seguindo a linha do serviço Face Time, que é um serviço da Apple para IOS que permite efetuar vídeo chamadas através de seus aparelhos, sem custo de ligação convencional, o Tango busca fazer isso, mas ampliando a gama de dispositivos, suportando então IOS e Android (TANGO, 2010).

Sua proposta é bastante próxima da idéia deste trabalho de conclusão, que é ser o mais transparente possível para o usuário final, à exceção, lógico, que o Tango suporta vídeo, o que foge do escopo do trabalho. Cadastros não são necessários, e após a simples instalação do software, já aparecem em sua tela as pessoas da sua lista de contatos que possuem o Tango em seus celulares. Ao simples toque no nome da pessoa, uma videochamada é iniciada com seu contato, e nenhuma tráfego na rede convencional é efetuado, todos dados são transmitidos através da internet.

Uma pesquisa na web mostrou que o protocolo utilizado nas entranhas do Tango é o XMPP com Jingle[5]. O cliente Tango é promissor e o fato de suportar vídeo chamada mostra que o protocolo XMPP tem muito potencial, e pode se estabelecer como um dos melhores no segmento em que atua.

2.3 PROTOCOLO SIP

Dentre todos os protocolos certamente o mais popular é o SIP. O Protocolo de Iniciação de Sessão (Session Initiation Protocol – SIP) está no nível de aplicação e é bastante similar ao HTTP, onde usa o modelo de requisição e resposta para estabelecer uma sessão. O protocolo é padrão da Internet Engineering Task Force (IETF) (RFC 2543, 1999). O SIP é protocolo unicamente de sinalização. Depois de estabelecida a sessão qualquer tipo de mídia pode ser utilizada, incluindo aí áudio e vídeo (WIKIPEDIA SIP, 2010).

SIP surgiu em meados da década de 1990 inicialmente para que fosse possível adicionar ou remover participantes dinamicamente numa sessão multicast. O SIP recebeu uma adoção rápida como padrão para comunicações integradas e aplicações que usam presença (Presença significa a aplicação estar consciente da sua localização e disponibilidade).

O SIP foi claramente inspirado nos protocolos já existentes e populares, como SMTP (e-mail) e o HTTP (páginas da web), que utilizam o formato de texto, com algumas definições sintáticas e semânticas. No SIP pode-se estabelecer, mudar e terminar chamadas entre dois ou mais utilizadores numa rede IP de uma maneira totalmente independente do conteúdo de dados da chamada. Principais características:

- Possui apenas seis métodos;
- Apresenta independência do protocolo de transporte;
- É baseado em texto.

5 Site onde foi encontrado: http://mobile.engadget.com/2010/09/30/tango-launches-3g-and-wifi-video-calling-on-ios-and-android-no/

Os principais componentes da arquitetura do SIP são:

Agente do Utilizador (UA):

É o responsável pela utilização do serviço, também chamado de terminal SIP, ou então de softphone (no caso de softwares fazendo o papel de terminal). O próprio agente funciona como um cliente no período de inicialização da sessão e também age como um servidor, quando responde a um pedido de sessão. Neste caso temos uma arquitetura cliente servidor. Porém, existe uma peculiaridade que é o fato de o mesmo dispositivo ser cliente ou servidor, dependendo da forma que for usado (originar ou receber chamadas). O agente utilizador por si só já seria capaz de receber chamadas e gerenciar a mesma. Os componentes adicionais servem somente para gerenciamento e funcionalidades adicionais.

Servidor Proxy SIP (SipProxy):

É um tipo de servidor intermediário. É usado para centralizar e encaminhar chamadas para outro nodo SIP (outro SipProxy, UA, etc.). O SipProxy pode ser utilizado para, por exemplo, manter uma base central de todos agentes utilizadores existentes naquela rede e encaminhar a chamada para o IP de um deles. Assim, quando o servidor receber o pedido de uma ligação, pode utilizar múltiplos métodos para tentar resolver o pedido de endereço de host, incluindo busca de DNS, busca em base de dados ou retransmitir o pedido para o “próximo” servidor proxy.

Servidor de registro:

Pode ser utilizado para centralizar informações dos usuários disponíveis na rede. Cada UA que se conecta avisa o servidor de registro, que armazena as informações do UA, e pode fornecer essas informações para o SipProxy. Frequentemente também servidores de registro são implementados no mesmo serviço que o SipProxy.

A combinação de um SipProxy com um servidor de registro dá ao SIP grande flexibilidade de arquitetura. Permite por exemplo que um mesmo utilizador mude de localização que está registrado (IP de sua conexão), e possa ser localizado. Outro resultado da arquitetura do SIP é a sua adequação natural como um ambiente de

colaboração devido às suas habilidades de apresentar múltiplos tipos de dados, aplicações, multimídia, etc. com uma ou mais pessoas.

2.3.1 Cliente Fring

O Fring é um software disponível para múltiplas plataformas e que suporta diversos dos protocolos de comunicação existentes hoje em dia. O diferencial do Fring em relação ao Nimbuzz é que ele é voltado exclusivamente para dispositivos móveis. Existem versões disponíveis para sistema operacional Symbian, para o IOS da Apple, e para o Android. (FRING, 2010).

Da mesma forma que o Nimbuzz, o Fring suportou Skype por algum tempo, mas houve alguns desentendimentos entre o Skype e a Fring, que fez com que o suporte a Skype fosse retirado. Hoje o fring conta com seu próprio serviço de chamadas de saída, o FringOut. O mesmo permite que chamadas sejam efetuadas diretamente do SmartPhone para qualquer número telefônico do mundo, a preços baixos. O Fring também suporta comunicação entre seus usuários de forma gratuita, tanto por áudio quanto por vídeo.

O Fring possui a possibilidade de adicionar qualquer provedor SIP para uso de chamadas Voip, caso o usuário não queira usar o serviço FringOut. Para isso, é necessário que informe nas suas configurações os dados do SipProxy desse provedor. A versão para Android, que pode ser vista na Figura 3, é uma das mais funcionais e suporta todas as funcionalidades da plataforma.

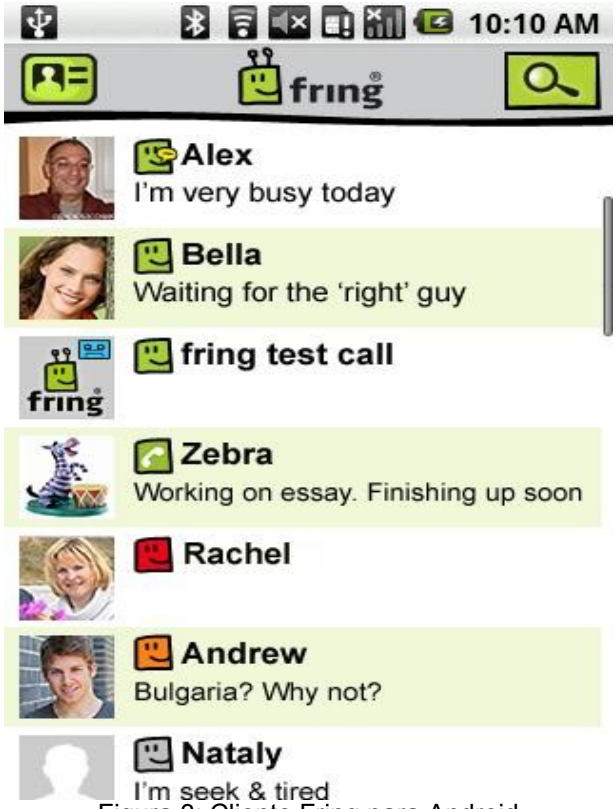


Figura 3: Cliente Fring para Android

2.3.2 CLIENTE SIPDROID

O SipDroid é bastante conhecido no mundo Android, pois foi uma das primeiras implementações do protocolo SIP para a plataforma. O que diferencia o SipDroid dos softwares apresentados até aqui é que ele foi concebido especificamente para o sistema Android e para o protocolo SIP. Assim existe uma maior integração com o Sistema Operacional, e uma maior cobertura das funcionalidades do protocolo SIP. Desde a versão 1.5 o SipDroid suporta também chamadas de Vídeo utilizando SIP (SIPDROID, 2010).

Uma das mais interessantes integrações que o mesmo possui com o sistema, é que se pode configurar para que todas as ligações feitas utilizando o discador padrão do sistema, saiam diretamente por Voip, de uma forma transparente. Para usar o SipDroid, o usuário deve registrar algum provedor SIP, ajustando em suas configurações as informações do mesmo. Na figura 4 pode ser visto uma imagem do software na sua tela inicial.

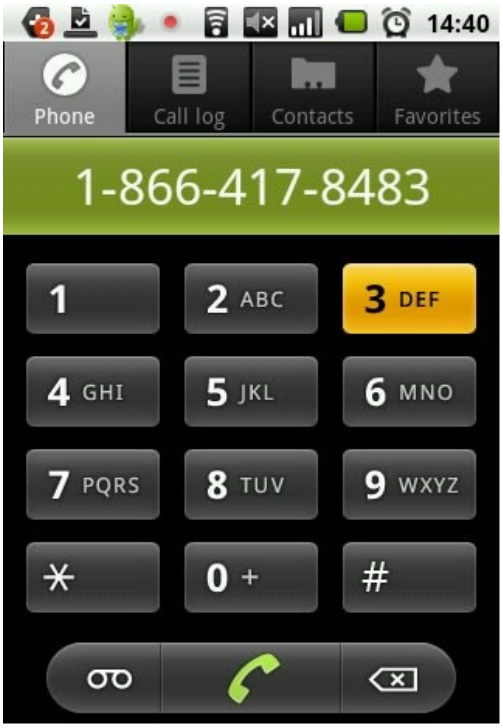


Figura 4: Cliente SipDroid

2.3.3 CLIENTE CSIPSIMPLE

Outro cliente do mesmo nível do SipDroid é o CSipSimple. Possui basicamente as mesma funcionalidades, se integrando perfeitamente ao discador nativo. Assim como no SipDroid, necessita de um provedor SIP para ser utilizado, que deve ser ajustado nas configurações. Possui algumas funcionalidades diferenciadas, como um assistente de configuração rápida, possibilidade de permitir chamadas ou não pelo tipo de conexão (3G, Edge, Wifi) e integração da chamadas enviadas e recebidas com os registros normais do Android. Na Figura 5 pode ser visto o CsipSimple em ação.

Figura 5: Cliente CSipSimple

3 ESTUDO DE FORMAS DE PROVER WEB-SERVICES

Para o desenvolvimento deste trabalho, será necessário alguma forma para que a aplicação cliente no Android possa interagir rapidamente com o servidor e publicar informações como seu status (online, offline) e informações (ip, número de telefone, etc.). A forma mais utilizada hoje em dia para fazer este tipo de comunicação é através de Web Services, que nada mais é que uma forma genérica de fazer comunicações entre sistemas que estejam em arquiteturas, linguagens e sistemas operacionais diferentes. Essa comunicação deve ser sempre através da Web. No interesse do projeto serão comparados as duas formas mais conhecidas de utilizar Web Services, SOAP e REST.

3.1 SOAP

O SOAP (Simple Object Access Protocol) é um protocolo bastante utilizado para troca de informações em plataformas distribuídas. Ele é baseado em XML para formatação das mensagens e utiliza outros protocolos da camada de aplicação, normalmente RPC e HTTP para negociação de transmissão das mensagens (WIKIPEDIA SOAP, 2010).

Uma mensagem SOAP normalmente consiste de três partes: um envelope, que define o conteúdo da mensagem e como ela será processada; um conjunto de regras que devem expressar instâncias dos tipos de dados da aplicação; e uma convenção de como serão as chamadas de procedimentos e as respostas.

Normalmente servidores SOAP são implementados usando servidores HTTP, e as mensagens são documentos XML que aderem à especificação da W3C.

SOAP pode ser usado tanto de forma anônima como com autenticação (nome/senha). Os pedidos SOAP podem ser feitos em três padrões: GET, POST e SOAP. Os padrões GET e POST são idênticos aos pedidos feitos por navegadores Internet. O SOAP é um padrão semelhante ao POST, mas os pedidos são feitos em XML e permitem recursos mais sofisticados como passar estruturas e arrays. As respostas das requisições também são documentos XML. O XML descreve perfeitamente os dados em tempo de execução e evita problemas causados por

mudanças nas chamadas das funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo muito robusto.

O SOAP define também um padrão chamado WSDL, que descreve perfeitamente os objetos e métodos disponíveis, através de páginas XML acessíveis através da Web. A idéia é a seguinte: quem publicar um serviço cria também estas páginas. Quem quiser chamar o serviço, pode usar estas páginas como “documentação” de chamada e também para validar se foi feita algum alteração na chamada da função.

3.2 REST

O REST foi descrito por Roy Fielding em sua tese de doutorado que contestando alguns princípios do SOAP propôs uma nova arquitetura para construção de Web Services.

REST significa Representational State Transfer, em português seria Transferência de Estado Representacional. Basicamente no REST cada URL única é uma representação de algum objeto. REST usa HTTP como ele foi concebido, com GET, POST, PUT e DELETE (estes últimos dois quase não são utilizados, mas estão na especificação desde o início). Uma chamada REST nada mais é que uma chamada para uma URL. O retorno desta URL poderia ser um XML, JSON, YAML, etc.

Podemos ver hoje muitas empresas de grande porte usando Web Services deste tipo, entre elas: Yahoo nos serviços web, incluindo o Flickr, a API do del.icio.us, PubSub, Bloglines, Technorati. eBay e Amazon provem seus métodos das duas formas: REST e SOAP. Amazon através de suas estatísticas informa que 90% de seu trafego é através de REST, por opção dos usuários.

3.3 COMPARATIVO

Vantagens do REST:

• Leve – não exige uma quantidade grande de XML que pode comprometer o desempenho;

• Os resultados podem ser lidos naturalmente;

• Fácil de construir – não são necessários grandes implementações, sendo a maioria das funcionalidades desenvolvidas com poucas linhas de código.

Vantagens do SOAP:

• Fácil de consumir;

• Rígido - verificação de tipo, aderência a especificação do sistema;

• As ferramentas de desenvolvimento possuem funções prontas para gerar códigos e templates WSDL por exemplo.

Em resumo, caso a necessidade seja um modelo independente de linguagem, plataforma e protocolo, SOAP é a melhor opção. Agora, se o serviço resume-se especificamente a um serviço web, onde a informação tende a ser transportada somente através do protocolo HTTP, com certeza o REST tenderá a ser implementado de uma forma muito mais simples.

4 MODELAGEM

Neste capítulo será descrita a modelagem do sistema como um todo. Pretende-se apresentar todos os pontos necessários para o funcionamento do mesmo. O sistema unirá partes de tecnologias já existentes e softwares prontos, com algumas novas partes que terão que ser desenvolvidas, por serem específicas desta solução. Este modelo está sendo pensado para que possa ser implementado de uma forma rápida, justamente com foco em ter neste mesmo trabalho uma demonstração funcional do sistema.

4.1 DEFINIÇÕES

Após os estudos realizados nos primeiros capítulos deste trabalho, e do conhecimento prévio do autor nas tecnologias envolvidas, algumas decisões precisam ser tomadas, para servir como base do modelo.

4.1.1 PROTOCOLO VOIP

Fundamentalmente é necessário definir o protocolo de comunicação VOIP que será utilizado. A escolha natural recai sobre o SIP, pela sua ampla utilização, por sua maturidade no mercado, pela quantidade de ferramentas disponíveis de forma livre para gerenciamento do protocolo. Em especial temos softwares que atuam como servidores Sip, que estão disponíveis de forma livre são maduros e estáveis. Alguns exemplos são Asterisk, OpenSips, OpenSer, Kamalio, etc.

Soluções já implementadas no Android de cliente para SIP, como SipDroid, e CSipClient também influenciaram na decisão, principalmente pela parte do trabalho do Guilherme de Moras Uzejka (2011), que possivelmente irá precisar de um software base para seu trabalho.

4.1.2 SERVIDOR SIP

Após definido que o protocolo utilizado será SIP, é necessário definir uma servidor para atuar centralizando os registros de todos os usuários, e saber se determinado usuário está online e para qual IP uma chamada deve ser encaminhada. Quem faz esse papel em arquiteturas SIP é o chamado Sip Proxy, que pode acumular também a função de servidor de registro. (funções descritas no capítulo 2.3).

Para este trabalho foram levantados duas possibilidades: utilizar como servidor Proxy o OpenSips, ou então usar o Asterisk para este papel. Existem outras implementações, de servidores Proxy SIP, mas por serem muito similares ao OpenSIPS, nem foram consideradas.

OpenSIPS: é uma implementação de código livre robusta e madura de um servidor SIP. Roda em Linux e sistemas Unix-Like. Porém ele é mais do que um simples Proxy, pois ele inclui funcionalidades de nível de aplicação. Pode ser usado como componente central de qualquer solução Voip que utilize o protocolo SIP para sinalização. Possui um motor muito flexível e personalizável de roteamento. Sendo totalmente modular, inclui módulos para conversas com vídeo, mensagens instantâneas, serviços de presença, entre outros (OPENSIPS, 2011).

Asterisk: é um software para Linux que transforma um computador em um servidor de comunicação Voip. Suporta vários protocolos, entre eles o SIP. Possui muitas opções de configuração, e atualmente é bastante utilizado em várias empresas, principalmente por seus poderes como central PCX, servidor de conferência e aplicações interativas de voz. O Asterisk também possui seu código livre, mas é suportado por uma empresa chamada Digium, que fornece soluções baseadas no mesmo (VOIP-INFO, 2011).

Optou-se por utilizar o Asterisk pela maior facilidade de configuração, pela possibilidade de colocar suas configurações em um banco de dados, e pelas possibilidades de expandir funcionalidades usando o AMI e o AGI, dois protocolos prontos para conexão de programas externos.

O AMI (Asterisk Manager Interface) permite um programa conectar em uma instancia do Asterisk e executar comandos ou ler eventos do PBX através de uma conexão TCP/IP .

Já o AGI (Asterisk Gateway Interface) foi criado para adicionar funcionalidades ao Asterisk. O recurso é uma fronteira de comunicação entre dois componentes de software. Com ele é possível uma integração operacional entre sistemas e o Asterisk. Abaixo, na Figura6, visualizamos um diagrama encontrado na internet sobre o funcionamento do AGI:[6]

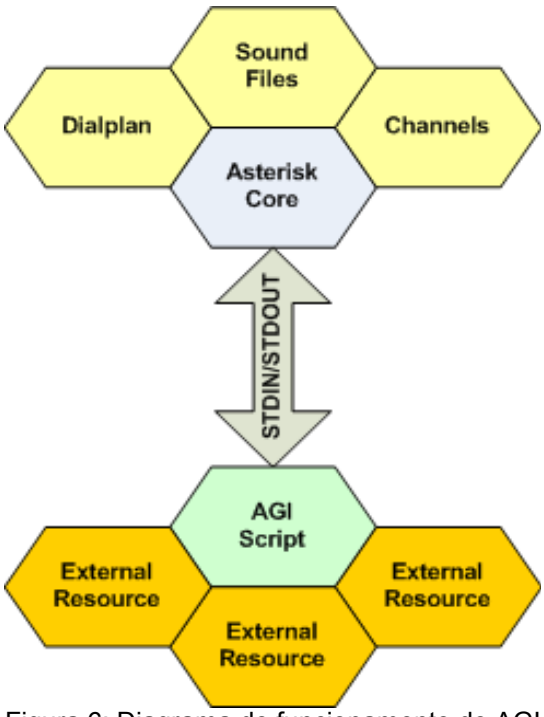


Figura 6: Diagrama de funcionamento do AGI

4.1.3 WEB SERVICES

Baseado nas idéias introdutórias, também era necessário a escolha de uma arquitetura de Web Services para ser utilizada. Foi optado pelo REST, em detrimento ao SOAP, pela sua simplicidade, e velocidade, para quem sabe possamos ter uma melhor performance. As requisições devem ser muito rápidas, utilizar pouco tráfego

6 Retirado do site: http://www.bestlinux.com.br/index.php/dicas/118/5435

de dados e de preferência desprender pouco gasto de energia, algo crucial em dispositivos móveis.

REST virou opção também pelo trabalho do Guilherme de Moraes Uzejka (2011), pois sendo baseado em HTTP há inúmeros exemplos de utilização desse protocolo diretamente de forma nativa no Android. Aliás REST é frequentemente primeira opção em plataformas móveis justamente por todos os fatores comentados da sua simplicidade.

4.1.4 LINGUAGEM DE PROGRAMAÇÃO

Para desenvolvimento do Web Services, se fez necessário o uso de alguma linguagem de programação. Nesse caso não existe nenhuma restrição, pois praticamente todas as linguagens para Web permitem a construção de Web Services. Foi optado então pela linguagem PHP, principalmente pela facilidade de instalação no Linux, e pela familiaridade do autor com a mesma.

4.2 COMPONENTES DO SISTEMA

Com as definições em mãos é preciso descrever todos componentes necessários para o sistema funcionar e demonstrar a forma como os mesmos irão ser interligar.

4.2.1 HARDWARE E SISTEMA OPERACIONAL

Cabem aqui alguns breves comentários sobre questões de infra-estrutura básica para o projeto ser viabilizado. O servidor do sistema será responsável por centralizar as ligações da plataforma, e toda sinalização SIP para estabelecer a chamada, assim como todo áudio das mesmas deve passar por essa máquina.

O hardware para este servidor irá depender muito mais de quantidade de usuários que o sistema terá. Questões de escalabilidade do sistema não serão aprofundadas neste trabalho, visto que foge totalmente do escopo. Cabe comentar

somente que intuitivamente o gargalo da plataforma será o próprio Asterisk, visto que o mesmo controla a chamada e faz a negociação via sinalização SIP. Portanto a escalabilidade do sistema será praticamente equivalente a outros sistemas que rodem sobre o Asterisk. Estudos encontrados na internet sobre escalabilidade de um hardware com esse software podem ser aplicados a esse trabalho. Um pequeno aumento no uso de recursos pode ser previsto, em virtude dos componentes adicionados, mas nada de muito impactante.

Sendo assim podemos dizer que o hardware necessário é um servidor padrão PC, com conectividade à internet. Um IP com numeração válida e que seja fixo é necessário pois esse ip será o ponto de conexão dos clientes no Android. Por facilidade é recomendado que o ip esteja configurado diretamente na máquina, sem firewalls ou NAT no meio do caminho, porém caso algum desses componentes de rede esteja presente é possível também configurar o sistema, porém alguns cuidados adicionais podem ser necessários.

Como sistema operacional será utilizado o GNU/Linux. Novamente não existe nenhuma distribuição específica, somente é recomendado uma versão razoavelmente atual (de pelo menos dois anos para cá) para que não possam surgir problemas de incompatibilidade na instalação do Asterisk, e demais componentes.

Assim está montado o cenário básico para o projeto. Um servidor Linux configurado e conectado a internet é o ponto de partida.

4.2.2 SERVIÇO ASTERISK

Componente central do sistema o Asterisk deverá gerenciar todas as ligações. O mesmo possui inúmeras opções e formas de trabalhar, mas neste caso serão utilizados somente os recursos necessários para funcionamento do sistema deste trabalho.

Basicamente precisamos configurar o núcleo do Asterisk para atender requisições do protocolo SIP em uma determinada porta e determinado IP da máquina. Neste modelo será convencionado o uso da porta padrão para sinalização, 5060, no protocolo de transporte UDP. Para tráfego de áudio será definido o uso de

somente um codec, o codec GSM, que é livre, e implementado por grande parte dos clientes SIP.

O protocolo SIP encapsula o protocolo SDP para transmissão das informações sobre o áudio, como por exemplo a porta de áudio, e o codec a ser utilizado. Para transmissão do áudio em si, o protocolo RTP é utilizado. As portas a serem utilizadas para transmissão do áudio são definidas na configuração do Asterisk. Foi mantido a configuração padrão do mesmo, portas de 10000 a 20000. Cada ligação irá usar umas dessas portas, como canal para transmissão do áudio codificado como GSM. O áudio também utiliza como transporte o protocolo UDP.[7]

Em seguida o Asterisk deve ser configurado para registrar os usuários, associando uma linha para cada um deles. Para este sistema o funcionamento será bem simples. Para cada Smartphone que se cadastrar no sistema, será criado um registro associado ao número telefônico do mesmo, com 11 dígitos: 0 + AC + Numero. (ex: 05199991234).

No Asterisk para cada linha registrada, deve ser associado um contexto para o qual será encaminhada a chamada que chegar desta linha. Esse contexto é normalmente utilizado para executar comandos do Asterisk, como por exemplo tocar um áudio, solicitar dígitos, ou encaminhar chamada para algum local específico. No escopo do sistema deste trabalho, um único contexto será suficiente, sendo assim de forma fixa todas as linhas registradas irão apontar para o mesmo. Esse contexto terá somente dois comandos: efetuar a chamada para o número discado pelo cliente, e após encerrar.

Tendo as linhas criadas e o contexto configurado, uma chamada pode ser realizada tendo como origem umas das linhas configuradas e destino alguma das outras.

7 Mais detalhes sobre o protocolo SIP não serão aprofundados neste trabalho, pois fogem do escopo do mesmo. Na apêndice B, existe um exemplo de sinalização completa de uma chamada, que pode ser consultado para uma rápida compreensão sobre os comentários acima.

4.2.2.1 CONFIGURAÇÕES EM BANCO DE DADOS

Definidas as configurações básicas do Asterisk, surge logicamente a demanda específica deste sistema, que é poder criar sem intervenção humana linhas no Asterisk e permitir o imediato uso das mesmas. Por padrão no Asterisk essas linhas são configuradas através de arquivos texto gravados no disco. A edição desses arquivos via software seria um tanto quanto inviável, devido á complexidade da sintaxe do mesmo.

Entretanto o Asterisk disponibiliza por padrão módulos que permitem acesso a Banco de Dados, o que é uma opção bem mais viável para este trabalho. O Asterisk permite nativamente duas formas básicas de ser configurado através de um Banco de Dados:

- **Asterisk RealTime**: mecanismo que permite colocar as seguintes configurações no banco: sippeers, sipusers, iaxpeers, iaxusers, voicemail, musiconhold, queues, queue_members, extensions. Cada uma destas possui então uma estrutura de tabela definida e o Asterisk irá consultar em tempo real, ou seja, no momento que o recurso for requisitado, estas tabelas para verificar a existência do determinado recurso. Neste modo de utilização, ao adicionar qualquer registro nestas tabelas, não é necessário reiniciar o serviço do Asterisk.

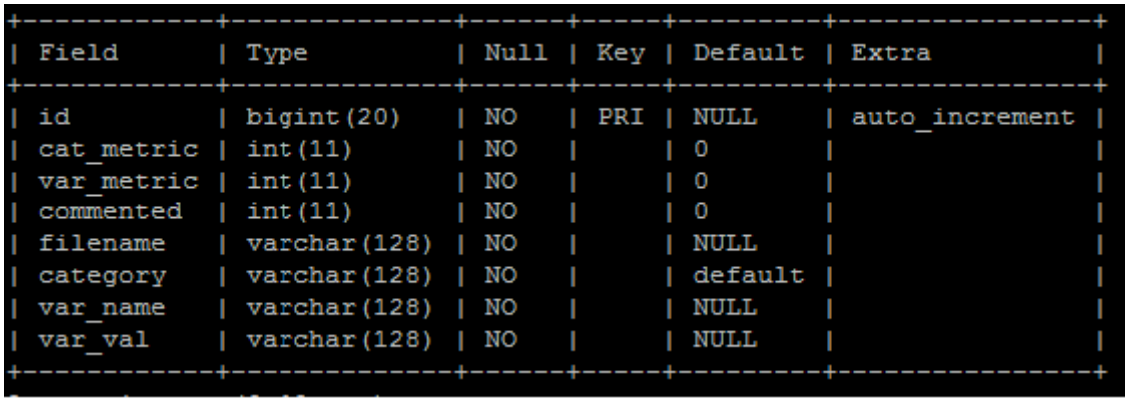
- **Asterisk RealTime Static**: mecanismo que permite colocar qualquer configuração do Asterisk em banco, utilizando uma tabela única, onde cada registro representa uma linha de configuração, em um determinado arquivo (uma coluna serve para representar o mesmo). Neste modo porém, sempre que uma configuração for alterada, adicionada ou removida, deve ser invocado o comando “reload” do Asterisk, que irá efetuar uma busca nos registros do banco e atualizar o status das configurações.

Para este trabalho basicamente será necessário criar dinamicamente linhas, ou sipusers usando o termo no software. Assim sendo, qualquer uma das duas formas de armazenamento seria possível, e à primeira vista a mais recomendada e prática seria a primeira, visto que não seria necessário o “reload” do serviço, após a adição de uma nova linha. Entretanto existe uma limitação no primeiro mecanismo,

no que diz respeito à verificação da condição de uma linha. Não é possível perguntar ao Asterisk com precisão se um determinado usuário está ou não online naquele momento.

Essa funcionalidade, de verificar a condição de uma determinada linha no sistema, é requisito fundamental do modelo do sistema, portanto recai-se sobre a segunda forma de armazenamento de configurações em banco de dados para o Asterisk, o “Asterisk RealTime Static”.

Basicamente para configuração desse modelo definido precisamos escolher e configurar um Banco de Dados, entre os suportados pelo Asterisk. Por simplicidade de instalação no SO Linux, foi definido o uso do Mysql. A tabela única que deve ser criada tem a seguinte estrutura:



Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	NULL	auto_increment
cat_metric	int(11)	NO		0	
var_metric	int(11)	NO		0	
commented	int(11)	NO		0	
filename	varchar(128)	NO		NULL	
category	varchar(128)	NO		default	
var_name	varchar(128)	NO		NULL	
var_val	varchar(128)	NO		NULL	

Figura 7: Estrutura tabela configurações do Asterisk no Mysql

Criada a tabela no Mysql, e com alguns pequenos ajustes na configuração do Asterisk, o mesmo já consegue verificar novas adições nesta tabela. Como adicional, para perfeito funcionamento desta configuração, todo programa que fizer uma alteração deve também efetuar o comando “reload” no Asterisk.

4.2.2.2 VERIFICANDO STATUS DE UMA LINHA DO SISTEMA

Conforme comentado anteriormente, tanto na forma padrão de armazenamento de configurações do Asterisk, arquivos texto, como na forma “Asterisk RealTime Static”, pode-se facilmente verificar o status de uma linha no

sistema. Isso é possível devido a uma das fases do protocolo SIP, a parte do Registro.

A parte do Registro nada mais é que uma sinalização de aviso, quando um cliente conecta em um Servidor SIP. Uma mensagem específica de “REGISTER” é enviada e é obtida uma resposta “OK”, que significa que o servidor entendeu que o determinado usuário está disponível para receber fazer e receber ligações. Após esta etapa, o próprio servidor SIP será responsável por verificar o status deste cliente, mandando sinalizações específicas para isso, como “OPTIONS” e/ou “INFO”, mantendo assim sempre um lista consistente de quais de seus usuários estão “conectados” naquele momento.

Resta então ter uma forma de consultar esse status, já controlado pelo servidor Asterisk. Como já foi visto anteriormente, existem dois mecanismos de acessar esse tipo de informação: AGI e AMI. Para este trabalho optou-se por usar o AMI, Asterisk Manager Interface.

Para simplificar a implementação de uso do AMI, existe uma ferramenta chamada MonAst[8] que é um painel para monitoramento de servidores Asterisk. Esse software possui um parte que implementa todas chamadas do AMI e disponibiliza um daemon que roda em uma porta TCP e possui comandos bem simples, como “GET STATUS” que retorna todas as linhas do sistema e seu status. Esse daemon será utilizado e a implementação poderá consultar o mesmo para validar o status de usuário no sistema.

4.2.3 REST WEB SERVICES

Como definido anteriormente para fazer as notificações entre o software cliente no SmartPhone Android e o servidor do sistema, foi optado pelo uso de Web Services do tipo REST. Basicamente será necessário o desenvolvimento de dois métodos que farão a iteração necessária para a plataforma funcionar conforme o esperado.

8 http://monast.sourceforge.net

4.2.3.1 MÉTODOS DA API

REGISTRAR:

O método “registrar” deve ser chamado no primeiro acesso de determinado telefone ao sistema. Será implementado como um método do tipo POST, que recebe como entrada o número que deve ser registrado.

Basicamente sua implementação consiste de uma conexão ao Banco de Dados Mysql e a execução de “Inserts” que efetuem a configuração da linha na tabela de configurações do Asterisk. Importante também é validar a existência anterior desta linha no Banco, para que não haja configurações duplicadas que possam impactar no funcionamento. Após a criação da linha, esse método também deverá efetuar um “reload” do Asterisk, para que de imediato a linha possa ser utilizada.

Este método não retorna nenhum tipo de conteúdo.

LIGAR:

O método “ligar” deve ser sempre chamado pelo cliente do SmartPhone quando existir uma intenção de completar uma ligação. Este método é implementado via GET, e recebe como parâmetro o número com o qual se deseja falar. O retorno do mesmo pode ser duas strings:

- “**on**” = significa que a linha do número para o qual se deseja efetuar uma chamada, está conectada via Voip no Asterisk.

- “**off**” = significa que a linha deste número não está conectada no momento, ou mesmo não existe na base, ou não é um número válido.

Assim sendo, esse método “ligar” permiti instantaneamente verificar o status de qualquer linha do sistema, com uma simples chamada ao Web Services REST.

4.2.3.2 FRAMEWORK SLIM

Conforme foi descrito anteriormente, para o desenvolvimento dos Web Services optou-se pela linguagem PHP. Para auxiliar no desenvolvimento foram pesquisado alguns Frameworks existentes nessa linguagem para a programação de Web Services REST.

Numa rápida pesquisa foi encontrado o Slim. O Slim através de classes do PHP permite o desenvolvimento muito rápido de Web Services. O exemplo da página inicial do sistema é o seguinte:

```
<?php
require 'Slim/Slim.php';
Slim::init();
Slim::get('/hello/:name', function ($name) {
    echo "Hello $name";
});
Slim::run();
?>
```

Este exemplo demonstra a simplicidade do Framework. Com 7 linhas de código, já temos um Web Service do tipo GET rodando e que permite a partir da chamada “/hello/NOME”, receber como retorno “Hello NOME”.

4.3 ARQUITETURA GERAL

O diagrama da Figura 8 mostra o funcionamento geral projeto do sistema. No cenário temos dois SmartPhones rodando Android e conectados a rede de dados 3G ou Wifi. A seta azul representa as requisições REST efetuados pela software cliente, e as setas vermelhas representam a ligação via protocolo SIP. No detalhe do canto esquerdo, em verde, temos as aplicações que rodam do lado do servidor, o Asterisk, o Banco de Dados e o Web Server controlando os Web Services REST.

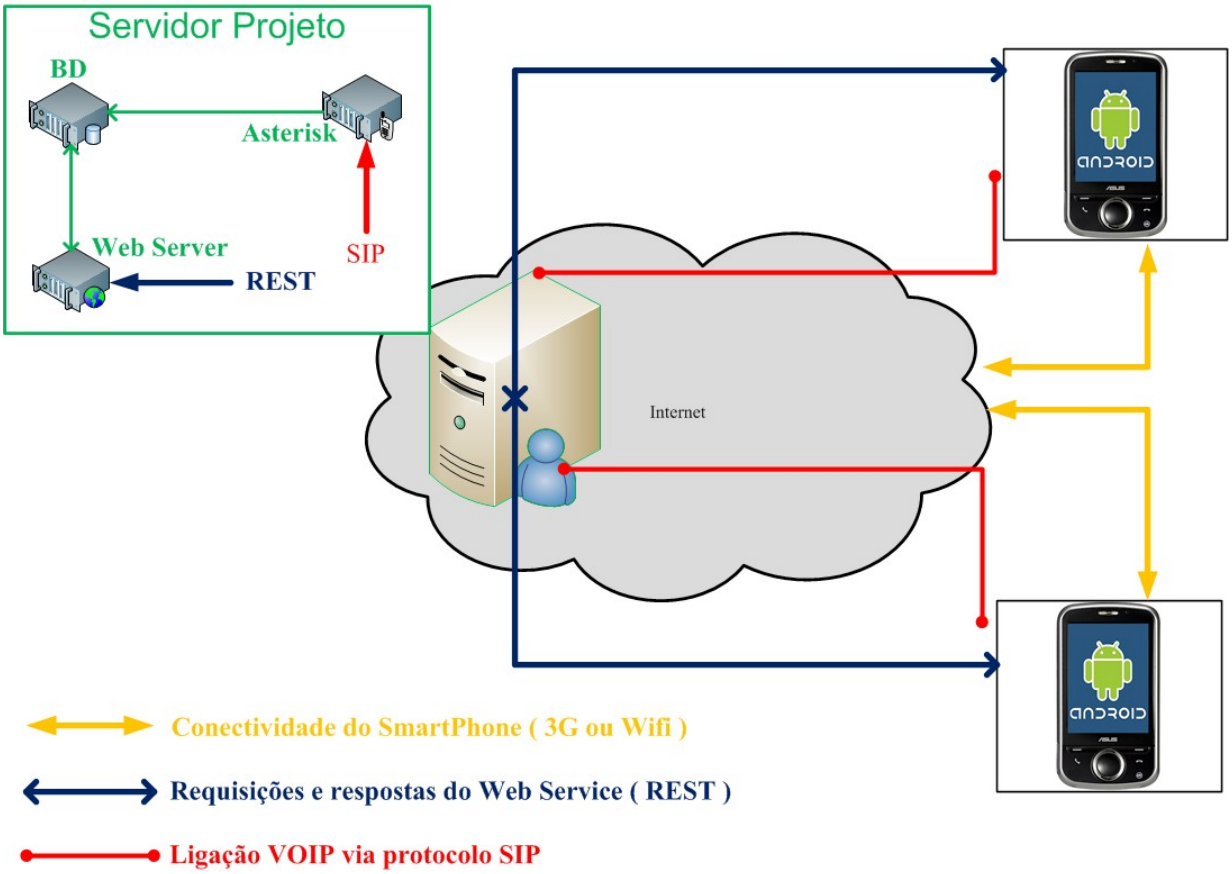


Figura 8: Diagrama da arquitetura

5 IMPLEMENTAÇÃO

Neste capítulo será descrito a implementação que foi realizada com base no modelo estudado no capítulo 4 deste trabalho. O objetivo principal foi conseguir um protótipo funcional, que permitisse completar uma chamada passando pela arquitetura montada. No final deste capítulo serão descritos os testes de funcionalidade realizados. Para obter o máximo de realidade nos testes, foi utilizado o software desenvolvido no trabalho do aluno Guilherme de Moras Uzejka (2011), que fez a parte do cliente Android.

5.1 INSTALAÇÃO SISTEMA OPERACIONAL

Conforme comentado no modelo, o requisito para instalação do sistema é uma máquina com sistema operacional Linux, conectada à Internet. Para os testes deste trabalho, foi alocada uma máquina no serviço Amazon EC2[9]. A configuração de hardware é a seguinte:

- Processador: Intel(R) Xeon(R) CPU E5430 @ 2.66GHz;
- Memória: 512 MB;
- Espaço em disco: 15 GB.

Como sistema operacional, o serviço EC2 disponibiliza várias imagens de distribuições Linux. Optou-se por instalar a versão Ubuntu Server 10.10. Por padrão as máquinas alocadas no EC2 utilizam um IP privado, o qual pode receber acesso externo via alguns redirecionamentos existentes. Porém, para os serviços de voz deste sistema, é necessário um ip público, que pode ser alocado na Amazon por demanda. Assim sendo, as configurações de rede ficaram as seguintes:

- IP configura na máquina: 10.209.199.8;
- IP externo alocado: 174.129.243.191;

9 Amazon EC2 significa Elastic Compute Cloud, que é o serviço da Amazon de servidores nas nuvens. Basicamente o serviço consiste de um painel de gerenciamento, onde é possível alocar servidores por demanda e com as configurações necessárias para determinada aplicação.

- Acesso aos serviços da máquina via Port Forwading[10];
- Portas redirecionados do ip válido para o ip interno: 22 (TCP), 80 (TCP), 5060 (UDP), 10000-20000 (UDP);
- Entrada DNS criada para facilitar o uso dos serviços. Voip.mbanton.net → 174.129.243.191.

5.2 INSTALAÇÃO E CONFIGURAÇÃO ASTERISK

O processo de instalação do Asterisk foi bastante simples, já que nas distribuições Ubuntu existem os repositórios de pacotes, com praticamente todas as ferramentas necessárias já preparadas para rodar. O comando utilizado para instalar o Asterisk e o componente para o mesmo que permite integração com o mysql foi:

```
[root:~] $ apt-get install asterisk-mysql
```

Após esse comando, todos os arquivos necessários para instalação serão baixados e instalados no sistema operacional. Para iniciar o serviço do Asterisk utilizou-se o seguinte comando:

```
[root:~] $ /etc/init.d/asterisk start
```

Após iniciado o Asterisk possui uma console própria, onde podem ser executado comandos para interagir com o mesmo. Alguns exemplos básicos podem ser vistos abaixo:

10 A técnica de Port Forwarding é conhecida na redes de computadores e consiste basicamente do redirecionamento de toda requisição que chega em um ip:porta para outro ip:porta, sendo esse na mesma máquina ou não.

```
[root:~] $ asterisk -r
domU-12-31-39-07-C4-FA*CLI> reload                    # Reinicia o serviço
domU-12-31-39-07-C4-FA*CLI> core set debug 9999       # Aumenta Debug
Core debug was 1 and is now 9999
domU-12-31-39-07-C4-FA*CLI> core set verbose 9999 # Aumenta verbose
Verbosity was 1 and is now 9999
domU-12-31-39-07-C4-FA*CLI> sip set debug on          # Ativa debug do SIP
SIP Debugging enabledA*CLI>
domU-12-31-39-07-C4-FA*CLI> help                      # Mostra os comandos
.......
domU-12-31-39-07-C4-FA*CLI> exit                      # Sai da console do Asterisk
[root:~] $
```

5.2.1 CONFIGURAÇÕES BÁSICAS DO ASTERISK

Todos os arquivos de configuração do Asterisk ficam no diretório /etc/asterisk. Algumas modificações foram feitas em relação às opções padrão que vem na instalação. Nas caixas abaixo, onde existe “...” indica que no arquivo existem mais opções que não foram alteradas e portanto foram omitidas do texto.

- Arquivo extensions.conf

```
[root:~] $ vim /etc/asterisk/extensions.conf
…
; O arquivo extensions.conf contém as configurações dos contextos que serão
; usados para processar um chamada. Conforme foi descrito no modelo, para que o
; sistema deste trabalho funcione, necessitamos de somente um contexto,
; que atenda a chamada, e encaminhe a mesma para o número de destino (outro
; usuário que está logado na plataforma). O nome escolhido para este contexto,
; que será necessário mais adiante, foi: “ANDROID_VOIP”
;
[ANDROID_VOIP]
exten => _X.,1,Dial(SIP/${EXTEN},30)
exten => _X.,n,HangUp()
```

- Arquivo logger.conf

```
[root:~] $ vim /etc/asterisk/logger.conf
…
; Este arquivo permite configurar as opções de log do Asterisk.
; O que foi alterado aqui foi a ativação do arquivo full, através da respectiva linha
; abaixo. Os arquivos de log do asterisk ficam na pasta /var/log/asterisk/ e o log
; foi ativado pois permite visualizar com detalhamento o funcionamento do asterisk
; o que foi importante durante o desenvolvimento do protótipo
[logfiles]
console => notice,warning,error
messages => notice,warning,error
full => notice,warning,error,debug,verbose
```

- Arquivo sip.conf:

```
[root:~] $ vim /etc/asterisk/sip.conf
…
; O arquivo sip.conf especifica todos os comportamentos do Asterisk para o
; protocolo SIP, incluindo as linhas que podem se autenticar no sistema.
; Porém no caso deste projeto foi utilizado o banco Mysql para armazenar as linhas.
;
; As configurações gerais usadas para o trabalho são as seguintes:
;
[general]
…
; Endereço de ip local, e porta para o serviço SIP ficar em “listen”
; No caso foi utilizado 0.0.0.0 como IP e 5060 como porta assim o asterisk
; automaticamente irá servir em todas interfaces da máquina
udpbindaddr=0.0.0.0:5060
…
; No caso do ambiente utilizado neste protótipo é necessário especificar
; o parâmetro localnet, que diz a rede à qual a máquina está fisicamente conectada
; e também o parâmetro externip, que diz qual ip que estão chegando as conexões
; do protocolo SIP. Esse parâmetros são necessários devido ao Port Forwarding
; usado na máquina de testes.
localnet=10.0.0.0/255.0.0.0
externip=174.129.243.191:5060
nat=yes
...
```

5.2.2 CONFIGURAÇÕES NO BANCO DE DADOS

Conforme foi descrito no modelo, para este trabalho foi necessário implementar alguma forma de armazenar as configurações. Foi optado pelos motivos

já citados pelo mecanismo Asterisk Realtime Static. Para o funcionamento desse mecanismo, é necessário primeiramente instalar e configurar o Banco de Dados. Para instalação do Mysql na máquina:

```
[root:~] $ apt-get install mysql-server
```

Após precisamos criar um banco de dados e um usuário para o Asterisk, além de criar a tabela ast_config, que já foi mostrada no modelo:

```
[root:~] $ mysql -u root -p
Enter password:
mysql> CREATE DATABASE asterisk;
Query OK, 1 row affected (0.03 sec)
mysql> GRANT ALL PRIVILEGES ON asterisk.* TO asterisk@localhost identified by
'a12345';
Query OK, 0 rows affected (0.04 sec)
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
mysql> USE asterisk;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysqll> CREATE TABLE ast_config (id bigint primary key not null
auto_increment,cat_metric int not null default 0,var_metric int not null default
0,commented int not null default 0,filename varchar(128) not null,category
varchar(128) not null default 'default',var_name varchar(128) not null,var_val
varchar(128) not null);
Query OK, 0 rows affected (0.17 sec)
mysql> exit;
Bye
```

Para configurar o Asterisk para buscar configurações do banco, devemos editar dois arquivos de configuração do mesmo:

- Arquivo extconfig.conf

```
[root:~] $ vim /etc/asterisk/extconfig.conf
…
; Cada linha neste arquivo específica a configuração de um arquivo de configuração
; do asterisk. A alteração realizada abaixo indica que o arquivo “sip.conf” utilizará
; a engine “mysql” , buscará pela configuração “asterisk.mysql” e usará a tabela
; “ast_config”
[settings]
sip.conf => mysql,asterisk.mysql,ast_config
```

- Arquivo res_mysql.conf

```
[root:~] $ vim /etc/asterisk/res_mysql.conf
…
; O arquivo res_mysql.conf serve para criar conexões com bancos de dados do
; tipo Mysql. Cada conexão é identificada por um nome, que é apresentado entre
; colchetes, e serve como identificador para ser usado no arquivo extconfig.conf
; No nosso caso criamos uma conexão chamada asterisk.mysql que acessa
; o banco criado anteriormente para o projeto.
[asterisk.mysql]
dbhost=127.0.0.1
dbname=asterisk
dbuser=asterisk
dbpass=a12345
dbport=3306
requirements=warn ; or createclose or createchar
```

Concluídos esses passos, o Asterisk utilizará somente as configurações da tabelas ast_config do Mysql para suas especificações que anteriormente estavam no arquivo sip.conf. Assim sendo primeiramente carregar as configurações já existentes no arquivo para a nova tabela, que por enquanto está vazia. Para fazer esse procedimento foi encontrado um script auxiliar em perl[11], que lê o arquivo sip.conf e insere os dados na tabela. O script se chama load_res_config.pl. Antes de executar o script foi necessário a instalação de alguns pacotes.

```
[root:~] $ apt-get install perl libclass-dbi-mysql-perl
[root:~] $ cd /etc/asterisk
[root:~] $ wget http://www.marlow.dk/tech/asterisk/load_res_config.pl
[root:~] $ vim load_res_config.pl
…
# Alterar a linha abaixo com os dados para acesso ao Mysql
  $dbh = DBI->connect("dbi:mysql:dbname=asterisk", "asterisk", "a12345") || die
$DBI::errstr;
…
[root:~] $ ./load_res_config.pl sip.conf
```

A partir desse momento todas as configurações do arquivo sip.conf foram carregadas no Mysql. Agora um reload no asterisk através de sua console, deixa o mesmo perfeitamente configurado para atuar como Servidor SIP do trabalho.

5.3 DESENVOLVIMENTO WEB SERVICES REST

Para desenvolvimento dos Web Services foi utilizada a linguagem PHP, pelos motivos já citados na modelagem. Primeiramente para instalação dos códigos desenvolvidos foi necessário instalar o WebServer apache e o php. Também é requerido instalar o Python e do Twisted-Python, que são dependências do projeto MonAst. Instalação via apt-get:

11 Pode ser obtido o fonte desse script através do seguinte link: http://www.marlow.dk/tech/asterisk/

```
[root:~] $ apt-get install apache2 php-apache php5 php5-mysql python twisted-python
```

Para fazer o desenvolvimento das funcionalidades de acesso ao Asterisk, conforme foi explicado no modelo, é necessário a utilização do MonAst. O procedimento de instalação do mesmo inclui o download do pacote e a instalação de configuração para que o serviço monast.py inicialize juntamente com o SO.

```
[root:~] $ cd /root/
[root:~] $wget http://downloads.sourceforge.net/project/monast/MonAst%20for%20Asterisk%201.6/2.0/monast-2.0.tar.gz
[root:~] $ tar -zxvf monast-2.0.tar.gz
[root:~] $ cd monast-2.0/
[root:~] $ ./install.sh
[root:~] $ cp contrib/init.d/rc.debian.monast /etc/init.d/monast
[root:~] $ vi /etc/monast.conf
…
# Configuração para o monast conseguir conectar no Asterisk
[server: ASTERISK_LOCAL ]
hostname=127.0.0.1
hostport=5038
username=monast_user
password=monast_secret

[root:~] $ vi /etc/asterisk/manager.d/monast.conf
; Liberação no Asterisk para o usuário do monast conectar
[monast_user]
secret=monast_secret
writetimeout=100
read=system,call,log,verbose,command,agent,user,config,originate
write=system,call,log,verbose,command,agent,user,config,originat

[root:~] $ /etc/init.d/monast start
```

Com o Monast instalado, restou somente efetuar a configuração do Web Server, o Apache, para permitir que uma pasta do sistema seja usada para desenvolvimento dos scripts em PHP. No Ubuntu o apache já vem pré-configurado, sendo necessário somente a adição de um site na devida pasta:

```
[root:~] $ vim /etc/apache2/sites-available/voip
<VirtualHost *:80>
  ServerName voip.mbanton.net
  ServerAdmin webmaster@localhost
  DocumentRoot /var/voip/
  …
</VirtualHost>
[root:~] $ cd /etc/apache2/sites-enabled
[root:~] $ ln -s ../sites-available/voip
[root:~] $ /etc/init.d/apache2 reload
```

A configuração neste ponto está concluída. A pasta “/var/voip” é a pasta de trabalho para o sistema a ser desenvolvido. Primeiramente colocamos nela o código do Framework escolhido para desenvolvimentos dos Web Services REST, o Slim.

```
[root:~] $ cd /tmp/
[root:~] $ wget https://github.com/codeguy/Slim/zipball/master -O slim.zip
[root:~] $ unzip slim.zip
[root:~] $ cd codeguy-Slim-99820f7
[root:~] $ mv Slim /var/voip/
```

Após devemos criar um arquivo .htaccess, conforme instruções da documentação do Slim para que as urls possam ser utilizadas no formato direto, como por exemplo http://voip.mbanton.net/ligar/, ao invés de chamar o próprio script php como seria o convencional http://voip.mbanton.net/index.php?funcao=ligar.

```
[root:~] $ vim /var/voip/.htaccess
# .htaccess é o arquivo padrão que o apache verifica para reconfigurar opções
# do servidor web para aquela pasta em específico.
# Neste caso essas configurações são indicadas pela documentação do Slim
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php [QSA,L]
```

Na pasta /var/voip/ foram desenvolvidos os scripts em PHP que provêm as APIS. O código completo delas pode ser conferido na Apêndice A, mas a parte principal da implementação é a seguinte:

```
//GET ligar
Slim::get('/ligar/:numero', function ($numero) {
    $conAsterisk = new Asterisk(MONAST_HOST,MONAST_PORT);
    $status = $conAsterisk->ligar($numero);
    echo $status;
});

//POST registrar
Slim::post('/registrar/', function () {
     $conBanco = new BancoDados(BD_HOST, BD_USER, BD_PASS, BD_BASE,
BD_TABELA);
     $numero = Slim::request()->post('numero');
     $conBanco->registrar($numero);
});
```

Basicamente a implementação dos métodos segue o que foi definido no modelo. O método “ligar” utiliza uma instância da classe desenvolvida Asterisk para consultar o Servidor sobre o status de um determinado usuário no sistema. Já o

método “registrar” utiliza uma instância da classe BancoDados para inserir um novo usuário no sistema.

Toda implementação de código deste projeto foi desenvolvida em PHP, e pode ser consultada no Apêndice A. A implementação ficou divida basicamente em quatro arquivos:

- ✔ **index.php** => Implementação usando Slim das chamadas das APIS;
- ✔ **Config.php** => Constantes e configurações para o sistema;
- ✔ **Asterisk.php** => Classe de comunicação com o Asterisk, através do Monast;
- ✔ **BancoDados.php** => Classe de manipulação do Banco de Dados Mysql.

5.4 TESTES FUNCIONAIS

Para testar o protótipo implementado não foi utilizado nenhuma metodologia formal de testes, mas somente testes funcionais. Durante o desenvolvimento primeiramente foi necessário testar somente o funcionamento dos Web Services REST em separado, para depois validar o teste do sistema como um todo, efetuando ligações via plataforma.

5.4.1 TESTES WEB SERVICES

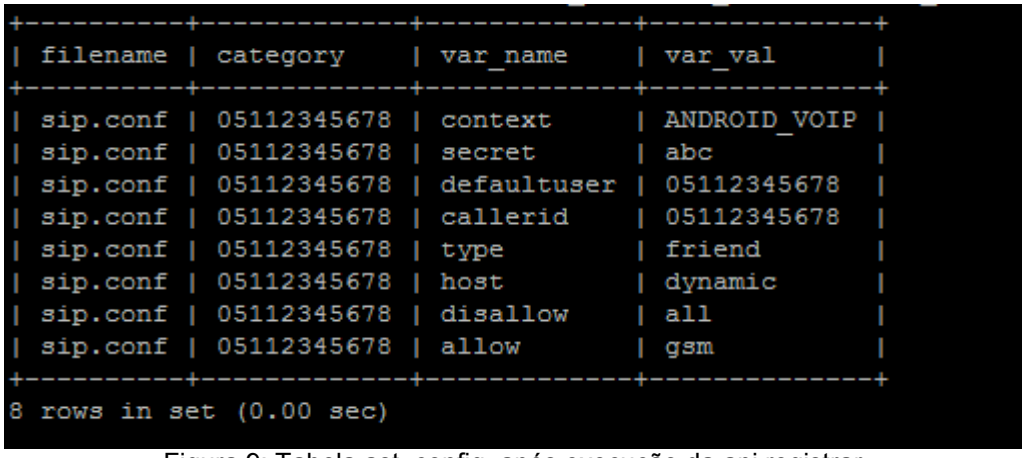
Para efetuar testes de funcionamento nos Web Services, a forma mais simples é efetuar requisições http diretamente ao servidor. No Linux existe um utilitário chamado curl, que permite fazer requisições http. Além disso foi utilizado o utilitário ngrep, para monitorar a chegada da requisição http ao servidor e a resposta de sucesso.

5.4.1.1 TESTANDO API REGISTRAR

Para testar a api de registro, após todo cenário configurado, e os arquivos php desenvolvidos rodando corretamente, devemos chamar a url http://voip.mbanton.net/registrar/ passando uma variável chamada número via método POST. O comando usado para testar foi:

```
[root:~] $ curl -d 'numero=05112345678' http://voip.mbanton.net/registrar/
# No momento da requisição acima, podemos visualizar a requisição
[root:~] $ ngrep “” tcp port 80
T 174.129.243.191:53758 -> 10.209.199.8:80 [AP]
POST /registrar/ HTTP/1.1.
User-Agent: curl/7.21.0 (i686-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.18.
Host: voip.mbanton.net.
Accept: */*.
Content-Length: 18.
Content-Type: application/x-www-form-urlencoded.
.
numero=05112345678
###
T 10.209.199.8:80 -> 174.129.243.191:53758 [AP]
HTTP/1.1 200 OK.
Date: Tue, 21 Jun 2011 02:46:01 GMT.
Server: Apache/2.2.16 (Ubuntu).
X-Powered-By: PHP/5.3.3-1ubuntu9.4.
Set-Cookie: PHPSESSID=34597749d71672dc3b1d91a7b0d3f6e3; path=/.
Expires: Thu, 19 Nov 1981 08:52:00 GMT.
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0.
Pragma: no-cache.
Content-Length: 0.
Set-Cookie: 34597749d71672dc3b1d91a7b0d3f6e3=flash%7Ca%3A0%3A%7B%7D; path=/.
Vary: Accept-Encoding.
Content-Type: text/html.
.
```

Após a requisição concluída, devemos conectar no Mysql e verificar se na tabela ast_config, foram inseridos os registros relativos ao número registrado:



filename	category	var_name	var_val
sip.conf	05112345678	context	ANDROID_VOIP
sip.conf	05112345678	secret	abc
sip.conf	05112345678	defaultuser	05112345678
sip.conf	05112345678	callerid	05112345678
sip.conf	05112345678	type	friend
sip.conf	05112345678	host	dynamic
sip.conf	05112345678	disallow	all
sip.conf	05112345678	allow	gsm

8 rows in set (0.00 sec)

Figura 9: Tabela ast_config, após execução da api registrar

Os campos representados na tabela ast_config mostrada na Figura 9, nada mais são que as configurações para esta linha no sistema. Basicamente as configurações são sempre as mesmas, a única informação que se alteram é o número telefônica usado para se registrar.

5.4.1.2 TESTANDO A API LIGAR

Para testar a api de verificação do status, foi necessário primeiramente criar uma linha no sistema e utilizar algum cliente SIP para se registrar no Asterisk. Após então a API deveria ter o seguinte comportamento:

- ✔ Em caso do cliente SIP estar ativo, a chamada deveria retornar “**on**”;
- ✔ Em caso do cliente SIP estar inativo, a chamada deveria retornar “**off**”.

Foi novamente usado o comando “curl” para realizar os testes porém neste caso a variável passada como parâmetro é via método GET. O comando é como segue:

```
# Teste quando o cliente SIP estava ativo
[root:~] $ curl http://voip.mbanton.net/ligar/05112345678/
on
[root:~] $ ngrep “” tcp port 80
##
T 174.129.243.191:56254 -> 10.209.199.8:80 [AP]
GET /ligar/05112345678 HTTP/1.1.
User-Agent: curl/7.21.0 (i686-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.18.
Host: voip.mbanton.net.
Accept: */*.
.

###
T 10.209.199.8:80 -> 174.129.243.191:56254 [AP]
HTTP/1.1 200 OK.
Date: Tue, 21 Jun 2011 02:59:02 GMT.
Server: Apache/2.2.16 (Ubuntu).
X-Powered-By: PHP/5.3.3-1ubuntu9.4.
Set-Cookie: PHPSESSID=84cf693efdb7001287ce39780a2b7ba8; path=/.
Expires: Thu, 19 Nov 1981 08:52:00 GMT.
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0.
Pragma: no-cache.
Content-Length: 3.
Set-Cookie:  84cf693efdb7001287ce39780a2b7ba8=flash%7Ca%3A0%3A%7B%7D; path=/.
Vary: Accept-Encoding.
Content-Type: text/html.
.
###
T 10.209.199.8:80 -> 174.129.243.191:56254 [AP]
on
#
```

O exemplo demonstrado mostra uma execução para o cliente SIP ativo. Para o cliente SIP inativo a requisição é exatamente a mesma, porém como foi comprovado no teste, a API retornou off.

5.4.2 TESTES DE LIGAÇÕES

Como teste final de integração foi feito uma simulação real de uso do sistema. Foram utilizados para o teste dois aparelhos com sistema Android, um Galaxy S da Samsung, e um Motorola Atrix. Em cada um deles foi instalada a aplicação final desenvolvida pelo aluno Guilherme.

Logo após o primeiro acesso foi validada a criação de duas linhas no banco de dados do Asterisk, uma para cada celular utilizado. Cada linha criada continha o respectivo número telefônico real do CHIP gsm inserido em cada aparelho. Após esta etapa, já era possível monitorar o status dos aparelhos. Chamadas manuais a api ligar, retornavam então “on” para ambos os números telefônicos.

Verificado o funcionamento das apis, foi feito uma ligação usando o discador implementado pelo Guilherme de Moras Uzejka no Android, de um aparelho para o número do outro. Para validar que a chamada estaria mesmo trafegando via VOIP, foram acompanhados no console do asterisk os logs da chamada. Após foi repetido o mesmo procedimento invertendo o aparelho que estaria originando a chamada. Tudo funcionou conforme o esperado. A chamada completou via Voip e pode-se conversar normalmente, com boa qualidade.

Após foi validado o comportamento no caso de um dos telefones não estar conectado no sistema. Foram desativados as redes de dados de um dos aparelhos, e a partir do outro foi efetuada uma tentativa de discar para ele. O resultado foi que o cliente Android consultou a api ligar, mas obteve “off” como resposta. Assim o software encaminhou a chamada para o discador convencional do Android e a ligação tocou da forma convencional no aparelho que estava sem rede dados.

5.5 PROTÓTIPO ALCANÇADO

Após os testes funcionais, pode-se verificar que o objetivo estabelecido de montar um protótipo funcional foi alcançado, pois o mesmo obtido permite completar chamadas e mostrar o funcionamento real da plataforma. Logicamente algumas questões da implementação do servidor ficaram simplificadas e podem causar problemas em caso de testes mais exaustivos e cenários de produção com muitos usuários usando o sistema ao mesmo tempo. Os scripts em PHP desenvolvidos são bem simples, e não dispõem de tratamentos contra erros e exceções. Logs de sistema também não foram incluídos. Um sistema de log completo é fundamental para sistemas que rodam como serviços.

Outras questões também não foram testadas, como desempenho e escalabilidade. O registro de vários usuários simultâneos, e o uso de várias chamadas ao mesmo tempo, são exemplos de situações que poderiam causar problemas inesperados. O tratamento das apis para o caso de um usuário tentar ligar para outro que esteja em uma ligação Voip também não foi testado e outras situações deste gênero poderiam acontecer. Para simplicidade do protótipo essas questões foram deixadas em segundo plano.

O protótipo serviu para demonstrar que o modelo proposto no capítulo quatro foi viável e que pode ser implementado em sua versão simples sem grandes complexidades. A expansão desse protótipo para um sistema real de produção exigiria mais algumas horas de programação, e principalmente de testes, além de buscar soluções para os problema citados acima.

6 CONCLUSÃO

Os softwares Voip existentes atualmente para plataforma Android possuem alguns impeditivos para o usuário final. Por exemplo, normalmente necessitam que para o seu uso seja criado um usuário e uma senha, o que pode acabar afastando o serviço usuários mais leigos. Outra questão é o fato de cada um possuir o seu sistema de contas independente dos outros. Isso acaba por prejudicar a adesão em massa. Nessa nova proposta de arquitetura, nenhum cadastro seria necessário e o sistema seria capaz de descobrir quais conhecidos do usuário estão com o software instalado e pronto para conversar via Voip.

Do lado do servidor, o estudo das tecnologias que estão disponíveis, muitas delas de forma livre, mostra que sem grandes esforços de desenvolvimento, pode-se montar um arquitetura robusta e funcional de Voip. A opção utilizada como software central, o Asterisk, já é maduro o suficiente tanto que já ganhou a confiança do mundo corporativo. O mesmo possui muitas possibilidades de expansão e integração, o que permite montar soluções Voip flexíveis. Funcionalidades adicionais poderiam ser colocadas facilmente, como contabilização das chamadas por usuário, Uras para atendimentos de voz, funcionalidades de caixa postal, etc.

O modelo feito para o servidor é genérico e apesar do foco deste trabalho ser a implementação de um cliente para plataforma Android, facilmente outras plataformas móveis poderiam ter softwares compatíveis. Protocolos comuns nesse tipo de dispositivo foram utilizados pensando nisso, como SIP para ligações e Web Services do tipo REST, que são os mais recomendados para dispositivos móveis.

Outra possibilidade vislumbrada é transformar o projeto em um produto comercial, que teria o foco de redução de custos tantos para pessoa física, quanto para empresas. O sistema poderia por exemplo ter várias opções de rotas Voip[12], para completar chamadas para outros números, inclusive internacionais por exemplo, que não estejam no sistema. Assim o servidor poderia, no momento de efetuar a chamada, responder para o software do cliente o quanto custará o minuto da mesma. O software cliente, ou mesmo o próprio usuário através de opções na tela, poderia então optar se faz a chamada por aquele custo, se usa a rede convencional.

12 rotas Voip são fornecidas por empresas que atuam nesse segmento, que normalmente fornecem um ip para onde a ligação deve ser enviada por SIP, e um tabela de custos que cobrará para cada destino e ou tipo de ligação (Fixo/Celular)

Comercialmente, poderia ser cobrada também uma taxa simbólica por minuto para tráfego entre clientes da plataforma. Para o usuário seria vantajoso pois estaria pagando um menor custo pela chamada, e ao mesmo tempo para empresa que fornecesse o produto o rendimento seria muito grande, a partir do momento que o volume de chamadas crescesse.

Já na parte do cliente, mais no caso do trabalho do aluno Guilherme de Moras Uzejka, o Android se mostrou uma plataforma extremamente flexível e com implementações às vezes de código aberto, que permitem ter um ponto de partida muito bom, tanto para protótipos de sistemas, como para para produtos finais. Devido a suas características de liberdade, permite ao desenvolvedor ter acesso a códigos exemplo, ferramentas de auxílio, e documentação vasta, o que seguramente acelera este tipo de projeto.

A parceria deste projeto, com a divisão em dois trabalhos, mostrou-se válida, pois permitiu implementar a solução completa e abordar tecnicamente pontos específicos de cada lado da plataforma. O trabalho em conjunto e a constante comunicação entre os dois autores foi fundamental para o sucesso do projeto. As tecnologias e plataformas escolhidas proporcionaram um aprendizado grande para os dois alunos, e também permitiram aplicar muitos dos conhecimentos adquiridos na universidade.

Por fim, espera-se que este trabalho sirva de motivador para outras soluções envolvendo Android, ou mesmo outras plataformas móveis, e telefonia Voip. Existe um grande mercado a ser explorado nessa união de tendências, e o potencial corporativo deste tipo de solução é bastante grande. Já para o mundo acadêmico essas tecnologias devem ser olhadas mais profundamente, pois elas estão se estabilizando no mercado, e o mesmo está aberto a soluções inovadoras que tragam principalmente integração, comunicação e facilidades em geral para a vida das pessoas.

REFERÊNCIAS

ANDROID. Disponível em: <http://www.android.com> Acesso em: Abr. 2011.

BLOG TECNOLOGIA Uol. Disponível em: <http://tecnologia.uol.com.br/ultimas-noticias/redacao/2011/05/10/microsoft-compra-skype-de-grupo-de-investidores-por-us-85-bilhoes.jhtm>. Acesso em: Jun. 2011.

CERAMI Ethan. Web Services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly, 2002.

FRING Client. Disponível em: <http://www.fring.com/>. Acesso em: Nov. 2010.

GONÇALVES Flavio E. Building Telephony Systems with OpenSIPS 1.6,Publisher: Packt Publishing, 2010.

KAMAILIO Server. Disponível em: <http://www.kamailio.org/>. Acesso em: Mai. 2011.

MOBILE Endgadget Blog. Disponível em: <http://mobile.engadget.com/2010/09/30/tango-launches-3g-and-wifi-video-calling-on-ios-and-android-no/>. Acesso em: Dez. 2010.

NIMBUZZ OUT. Disponível em: <http://www.nimbuzzout.com>. Acesso em: Nov. 2010.

NIMBUZZ Web Site. Disponível em: <http://www.nimbuzz.com/>. Acesso em: Nov. 2010.

OPENSIPS Server. Disponível em: <http://www.opensips.org>. Acesso em: Mai. 2011.

RICHARDSON Leonard & Sam Ruby. RESTful Serviços Web, O'Reilly, 2007.

SIP-ROUTER Project. Disponível em: <http://sip-router.org/>. Acesso em: Mai. 2011.

SIPDROID Client. Disponível em: <http://sipdroid.org/>. Acesso em: Dez. 2010.

SKYPE FOR ANDROID. Disponível em: <http://www.skype.com/intl/pt-br/get-skype/on-your-mobile/download/skype-for-android/> Acesso em: Mai. 2011.

SKYPE Web Site. Disponível em: <http://www.skype.com>. Acesso em: Nov. 2010.

TANGO Web Site. Disponível em: <http://tango.me/index.php>. Acesso em: Dez. 2010.

UZEJKA, Guilherme de Moraes, Modelando um cliente VoIP inteligente para a plataforma Android. UFRGS, 2011.

VOIP-INFO Reference Guide. Disponível em: <http://www.voip-info.org/ >. Acesso em: Mai. 2011.

WIKIPEDIA JINGLE Protocol. Disponível em: <http://en.wikipedia.org/wiki/Jingle_(protocol)>. Acesso em: Nov. 2010.

WIKIPEDIA REST. Disponível em <http://pt.wikipedia.org/wiki/REST>. Acesso em: Mai. 2011.

WIKIPEDIA SIP. Disponível em: <http://pt.wikipedia.org/wiki/SIP>. Acesso em: Dez. 2010.

WIKIPEDIA SIPDROID. Disponível em: <http://en.wikipedia.org/wiki/Sipdroid>. Acesso em: Dez. 2010.

WIKIPEDIA SKYPE Protocolo. Disponível em: <http://en.wikipedia.org/wiki/Skype_protocol>. Acesso em: Nov. 2010.

WIKIPEDIA SOAP Protocol. Disponível em: <http://pt.wikipedia.org/wiki/SOAP>. Acesso em: Dez.2010.

WIKIPEDIA VOIP. Disponível em: <http://pt.wikipedia.org/wiki/Voz_sobre_IP>. Acesso em: Nov. 2010.

WIKIPEDIA WEB SERVICES. Disponível em: <http://pt.wikipedia.org/wiki/Web_services>. Acesso em: Jun. 2011.

WIKIPEDIA XMPP Protocol. Disponível em: <http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol>. Acesso em: Nov. 2010.

APÊNDICE A – IMPLEMENTAÇÃO WEB SERVICES REST

Arquivo index.php

```
<?php
/* Implementacao de WebServices para arquitetura Voip
* Trabalho de conclusao de curso - Ciencia da Computacao - UFRGS
* Autor: Marcelo Bublitz Anton
*
* Arquivo index.php
*
* Contem a criacao dos Web Services REST, usado Framework SLIM
* Comentarios em Ingles sao do arquivo base do Framework, e foram mantidos
*
*/

require 'Config.php';
require 'Asterisk.php';
require 'BancoDados.php';
require 'Slim/Slim.php';

/**
 * Step 2: Initialize the Slim application
 *
 * Here we initialize the Slim application with its default settings.
 * However, we could also pass a key-value array of settings.
 * Refer to the online documentation for available settings.
 */
Slim::init(array('templates.path' => './'));

/**
 * Step 3: Define the Slim application routes
 *
 * Here we define several Slim application routes that respond
 * to appropriate HTTP request methods. In this example, the second
 * argument for `Slim::get`, `Slim::post`, `Slim::put`, and `Slim::delete`
 * is an anonymous function. If you are using PHP < 5.3, the
 * second argument should be any variable that returns `true` for
 * `is_callable()`. An example GET route for PHP < 5.3 is:
 *
 * Slim::get('/hello/:name', 'myFunction');
 * function myFunction($name) { echo "Hello, $name"; }
 *
 * The routes below work with PHP >= 5.3.
 */

//GET ligar
Slim::get('/ligar/:numero', function ($numero) {

        $conAsterisk = new Asterisk(MONAST_HOST,MONAST_PORT);
        $status = $conAsterisk->ligar($numero);

        echo $status;

});

//POST registrar
Slim::post('/registrar/', function () {

        $conBanco = new BancoDados(BD_HOST, BD_USER, BD_PASS, BD_BASE, BD_TABELA);
```

```
	$numero = Slim::request()->post('numero');

	$conBanco->registrar($numero);

});

// Pagina customizada de erro
Slim::notFound('custom_not_found_callback');
function custom_not_found_callback() {
		Slim::render('Sobre.php');
}

/**
 * Step 4: Run the Slim application
 *
 * This method should be called last. This is responsible for executing
 * the Slim application using the settings and routes defined above.
 */
Slim::run();

?>
```

Arquivo config.php

```
<?php
/* Implementacao de WebServices para arquitetura Voip
* Trabalho de conclusao de curso - Ciencia da Computacao - UFRGS
* Autor: Marcelo Bublitz Anton
*
* Arquivo Config.php
*
* Contem as definicoes gerais do comportamento dos Web Services
*
*/

// Constantes de configuracao do sistema
define('BD_HOST','localhost');
define('BD_USER','asterisk');
define('BD_PASS','a12345');
define('BD_BASE','asterisk');
define('BD_TABELA','ast_config');
define('MONAST_HOST', 'localhost');
define('MONAST_PORT', '5039');

// Parametros fixos dos peers do sistema
// Serao utilizados para insercao nas tabelas do asterisk
define('AST_TYPE','friend');
define('AST_SECRET','abc');
define('AST_CONTEXT','ANDROID_VOIP');
define('AST_HOST','dynamic');
define('AST_DISALLOW','all');
define('AST_ALLOW','gsm');
define('AST_ARQ_SIP','sip.conf');

// Comando para reload do asterisk, usado apos uma insercao de um usuario no sistema
define('AST_RELOAD_COMMAND','/usr/bin/sudo /usr/sbin/asterisk -r -x reload');
?>
```

Arquivo Asterisk.php

```
<?php
/* Implementacao de WebServices para arquitetura Voip
* Trabalho de conclusao de curso - Ciencia da Computacao - UFRGS
* Autor: Marcelo Bublitz Anton
*
* Arquivo Asterisk.php
*
* Classe responsavel pela comunicacao com o daemon monast.py
* que responde com o status de cada usuario no sistema
*
*/

class Asterisk {

	// Variaveis de instancia
	private $monAstHost;
	private $monAstPort;

	// Metodo construtor
	public function __construct($host, $port) {
		$this->monAstHost = $host;
		$this->monAstPort = $port;
	}

	// Metodo socketGetStatus
	// Conecta no daemon monast.py e envia o comando GET STATUS
	// O retorno deste comando no socket eh retornado como uma string unica para o retorno da funcao
	private function socketGetStatus(){

		$isStatus = false;
		$buffer   = "";

		$sock = @socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
		if ( $sock === false ) {
			session_write_close();
			echo "ERRO FATAL SOCKET CREATE";
			exit;
		}
		$conn = @socket_connect($sock, $this->monAstHost, $this->monAstPort);
		if ( $sock === false ) {
			session_write_close();
			echo "ERRO FATAL SOCKET_CONNECT";
			exit;
		}

		$id = rand();
		socket_write($sock, "SESSION: ". $id . "\r\n");
		while ($message = socket_read($sock, 1024 * 16)) {
				$buffer .= $message;

				if ($buffer == "NEW SESSION\r\n" || $buffer == "OK\r\n") {
					$buffer = "";
					socket_write($sock, "GET STATUS\r\n");
				}
				if ($buffer == "ERROR: Authentication Required\r\n") {
					session_write_close();
					socket_write($sock, "BYE\r\n");
					echo "ERRO FATAL SOCKET READ";
					exit;
				}
```

```
                                        if (strpos($buffer, "BEGIN STATUS\r\n") !== false)
                                                $isStatus = true;

                                        if (strpos($buffer, "END STATUS\r\n") !== false) {
                                                $buffer = trim(str_replace("BEGIN STATUS\r\n", "",
str_replace("END STATUS\r\n", "", $buffer)));
                                                $isStatus = false;
                                                socket_write($sock, "BYE\r\n");
                                        }
                                }
                                socket_close($sock);

                                return $buffer;
                        }


                        // Metodo ligar
                        // Metodo publico da classe
                        // Recebe um numero ao qual se deseja verificar a possibilidade ligar
                        // Retorna 'on' caso seja possivel, ou 'off' caso nao seja, ou em caso de numero nao encontrado
                        public function ligar($numero){

                                // Pega as informacoes do monast.py
                                $conteudoSocket = $this->socketGetStatus();

                                // Para cada conteudo entre chaves
                                foreach ( preg_split('%[\{\}]%i',$conteudoSocket) as $linha ) {

                                        $status = "";
                                        $peer = "";
                                        // Para cada valor separado por virgulas
                                        foreach ( preg_split('%,%',$linha) as $elemento ) {

                                                $indexa = preg_split('%:%',$elemento);
                                                // Nao eh elemento, pulamos
                                                if ( count($indexa) != 2 )
                                                        break;

                                                list($chave,$valor) = $indexa;
                                                $chave = trim(trim($chave),'\"');
                                                $valor = trim(trim($valor),'\"');;

                                                if ( $chave == "Status" )
                                                        $status = $valor;

                                                if ( $chave == "Peer" ) {
                                                        $indexa2 = preg_split('%/%',$valor);

                                                        // Nao eh peer do tipo SIP. Paramos com o laco mais externo,
podemos passar ao proximo resultado
                                                        if ( ( count($indexa2) != 2 ) || ( stristr($indexa2[0], "SIP") ===
false ) )
                                                                break;

                                                        $peer = $indexa2[1];

                                                }

                                        }

                                        // Se vazio, entao nao temos um peer valido
                                        if ( (empty($status)) || (empty($peer)) )
                                                continue;
```

```
                        // Se encontramos o peer do numero desejado, retornamos on se status Registered
                        if ( $peer == $numero ) {
                                if ( $status == "Registered" )
                                        return "on";
                                else
                                        return "off";
                        }

                }

                // Numero nao encontrado, retornamos off como padrao
                return "off";

        }

}

?>
```

Arquivo BancoDados.php

```
<?php
/* Implementacao de WebServices para arquitetura Voip
* Trabalho de conclusao de curso - Ciencia da Computacao - UFRGS
* Autor: Marcelo Bublitz Anton
*
* Arquivo BancoDados.php
*
* Classe responsavel pela iteracao com o Banco de Dados Mysql
*
*/

class BancoDados {

        // Variaveis de instancia
        private $bdHost;
        private $bdUser;
        private $bdPass;
        private $bdBase;
        private $bdTabela;

        // Metodo construtor
        public function __construct($host, $user, $pass, $base, $tabela) {
                $this->bdHost = $host;
                $this->bdUser = $user;
                $this->bdPass = $pass;
                $this->bdBase = $base;
                $this->bdTabela = $tabela;
        }

        // Metodo registrar
        // Metodo publico da classe
        public function registrar($numero){

                $conexao = @mysql_connect($this->bdHost, $this->bdUser, $this->bdPass) or die ("ERRO FATAL MYSQL_CONNECT");
                @mysql_select_db($this->bdBase,$conexao) or die ("ERRO FATAL MYSQL_SELECT_DB");

                $query = mysql_query("SELECT COUNT(1) FROM " . $this->bdTabela . " WHERE CATEGORY = '" . $numero . "'");
                $resultado = mysql_fetch_row($query);
```

```
		// Usuario nao existe na base, devemos cria-lo
		if ( $resultado[0] == 0 ) {
			$query = mysql_query("SELECT MAX(cat_metric) FROM " . $this->bdTabela);
			$resultado = mysql_fetch_row($query);
			$metric = $resultado[0] + 1;

			// Parametros basicos do peer. Definidos estaticamente neste ponto
			$dados['type'] = AST_TYPE;
			$dados['callerid'] = $numero;
			$dados['defaultuser'] = $numero;
			$dados['secret'] = AST_SECRET;
			$dados['context'] = AST_CONTEXT;
			$dados['host'] = AST_HOST;
			$dados['disallow'] = AST_DISALLOW;
			$dados['allow'] = AST_ALLOW;

			$this->insereNoBanco($numero,$metric,$dados,$conexao);

			$this->reloadAsterisk();
		}

		@mysql_close($conexao) or die ("ERRO FATAL MYSQL_CLOSE");

	}

	// Funcao que efetiamente faz o insert no banco
	private function insereNoBanco($numero,$metric,$dados,$conexao){
			$var_metric = 0;
			foreach($dados as $indice => $valor) {
				mysql_query("INSERT INTO " . $this->bdTabela . "
(cat_metric,var_metric,filename, category, var_name, var_val )
					VALUES ($metric, $var_metric, '" . AST_ARQ_SIP . "', '$numero',
'$indice', '$valor')",$conexao);
				$var_metric++;
			}
	}

	// Funcao que faz reload do asterisk
	private function reloadAsterisk(){

		exec(AST_RELOAD_COMMAND,$saida,$retorno);

		if ( $retorno != 0 ) {
			echo "ERRO FATAL EXECUTANDO COMANDO: \"" .
AST_RELOAD_COMMAND . "\"" . "\n";
			echo 'Retorno do comando: ' . $retorno . "\n";
			echo 'Linhas: ' . "\n";
			foreach($saida as $s) {
				echo $s . "\n";
			}
		}

	}

}
?>
```

APÊNDICE B – EXEMPLO DE SINALIZAÇÃO SIP DE UMA CHAMADA

Exemplo de sinalização SIP de REGISTRO de um terminal no Asterisk:

#
U 201.21.97.113:46179 -> 10.209.199.8:5060
REGISTER sip:voip.mbanton.net SIP/2.0.
Via: SIP/2.0/UDP 201.21.97.113:46179;rport;branch=z9hG4bKPjkjriNcJhMz26BPLSrZfPCyz6mCrB94f3.
Route: <sip:voip.mbanton.net;transport=udp;lr>.
Max-Forwards: 70.
From: <sip:05199991234@voip.mbanton.net>;tag=v7Ehz6QZmOeFVwIJAdtktG3l9qmLos.L.
To: <sip:05199991234@voip.mbanton.net>.
Call-ID: 5ORKNcbS.J1odAgjJjLHp3B5CmRh-1py.
CSeq: 16988 REGISTER.
User-Agent: CSipSimple r841 / GT-I9000-10.
Contact: <sip:05199991234@201.21.97.113:46179;transport=UDP;ob>.
Contact: <sip:05199991234@192.168.0.102:46179;ob>;expires=0.
Expires: 900.
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS.
Authorization: Digest username="05199991234", realm="asterisk", nonce="7d5275ae", uri="sip:voip.mbanton.net", response="cd5a3483f7512bac043c4916f2966fea", algorithm=MD5.
Content-Length: 0.
.

#
U 10.209.199.8:5060 -> 201.21.97.113:46179
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 201.21.97.113:46179;branch=z9hG4bKPjkjriNcJhMz26BPLSrZfPCyz6mCrB94f3;received=201.21.97.113;rport=46179.
From: <sip:05199991234@voip.mbanton.net>;tag=v7Ehz6QZmOeFVwIJAdtktG3l9qmLos.L.
To: <sip:05199991234@voip.mbanton.net>;tag=as4a860ecd.
Call-ID: 5ORKNcbS.J1odAgjJjLHp3B5CmRh-1py.
CSeq: 16988 REGISTER.
Server: Asterisk PBX 1.6.2.7-1ubuntu1.1.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO.
Supported: replaces, timer.
Expires: 900.
Contact: <sip:05199991234@201.21.97.113:46179;transport=UDP;ob>;expires=900.
Date: Tue, 21 Jun 2011 03:19:31 GMT.
Content-Length: 0.
.

Exemplo de uma chamada do SipClient Android, até o Asterisk. Por simplicidade, somente um lado da requisição foi demonstrado:

#
U 201.21.97.113:40130 -> 10.209.199.8:5060
INVITE sip:05199991234@voip.mbanton.net SIP/2.0.
Via: SIP/2.0/UDP 201.21.97.113:40130;rport;branch=z9hG4bKPjYJusFG6U.ErIzwCcKGlqQKdQNxk.ubCn.
Max-Forwards: 70.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>.
Contact: <sip:05199991234@201.21.97.113:40130;ob>.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19115 INVITE.
Route: <sip:voip.mbanton.net;transport=udp;lr>.
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS.
Supported: replaces, 100rel, timer, norefersub.
Session-Expires: 1800.

Min-SE: 90.
User-Agent: CSipSimple r841 / GT-I9000-10.
Content-Type: application/sdp.
Content-Length: 253.
.
v=0.
o=- 3517615712 3517615712 IN IP4 201.21.97.113.
s=pjmedia.
c=IN IP4 201.21.97.113.
t=0 0.
a=X-nat:0.
m=audio 4000 RTP/AVP 3 101.
a=rtcp:4001 IN IP4 201.21.97.113.
a=rtpmap:3 GSM/8000.
a=sendrecv.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.

#
U 10.209.199.8:5060 -> 201.21.97.113:40130
SIP/2.0 401 Unauthorized.
Via: SIP/2.0/UDP
201.21.97.113:40130;branch=z9hG4bKPjYJusFG6U.ErIzwCcKGlqQKdQNxk.ubCn;received=201.21.97.113;rpo
rt=40130.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>;tag=as52c21a6d.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19115 INVITE.
Server: Asterisk PBX 1.6.2.7-1ubuntu1.1.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO.
Supported: replaces, timer.
WWW-Authenticate: Digest algorithm=MD5, realm="asterisk", nonce="271bd31b".
Content-Length: 0.
.

#
U 201.21.97.113:40130 -> 10.209.199.8:5060
ACK sip:05199991234@voip.mbanton.net SIP/2.0.
Via: SIP/2.0/UDP 201.21.97.113:40130;rport;branch=z9hG4bKPjYJusFG6U.ErIzwCcKGlqQKdQNxk.ubCn.
Max-Forwards: 70.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>;tag=as52c21a6d.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19115 ACK.
Route: <sip:voip.mbanton.net;transport=udp;lr>.
Content-Length: 0.
.

#
U 201.21.97.113:40130 -> 10.209.199.8:5060
INVITE sip:05199991234@voip.mbanton.net SIP/2.0.
Via: SIP/2.0/UDP 201.21.97.113:40130;rport;branch=z9hG4bKPjd7Cotxkcc-NpZ2C3mC7TmLWRF5K3LeB..
Max-Forwards: 70.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>.
Contact: <sip:05199991234@201.21.97.113:40130;ob>.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19116 INVITE.
Route: <sip:voip.mbanton.net;transport=udp;lr>.
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER, MESSAGE,
OPTIONS.
Supported: replaces, 100rel, timer, norefersub.
Session-Expires: 1800.

Min-SE: 90.
User-Agent: CSipSimple r841 / GT-I9000-10.
Authorization: Digest username="05199991234", realm="asterisk", nonce="271bd31b",
uri="sip:05199991234@voip.mbanton.net", response="5d89e21e8e10b2efed8b44fe4c5f4888", algorithm=MD5.
Content-Type: application/sdp.
Content-Length: 253.
.
v=0.
o=- 3517615712 3517615712 IN IP4 201.21.97.113.
s=pjmedia.
c=IN IP4 201.21.97.113.
t=0 0.
a=X-nat:0.
m=audio 4000 RTP/AVP 3 101.
a=rtcp:4001 IN IP4 201.21.97.113.
a=rtpmap:3 GSM/8000.
a=sendrecv.
a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-15.

#
U 10.209.199.8:5060 -> 201.21.97.113:40130
SIP/2.0 100 Trying.
Via: SIP/2.0/UDP 201.21.97.113:40130;branch=z9hG4bKPjd7Cotxkcc-
NpZ2C3mC7TmLWRF5K3LeB.;received=201.21.97.113;rport=40130.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19116 INVITE.
Server: Asterisk PBX 1.6.2.7-1ubuntu1.1.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO.
Supported: replaces, timer.
Require: timer.
Session-Expires: 1800;refresher=uas.
Contact: <sip:05199991234@174.129.243.191>.
Content-Length: 0.
.

#
U 10.209.199.8:5060 -> 201.21.97.113:40130
SIP/2.0 200 OK.
Via: SIP/2.0/UDP 201.21.97.113:40130;branch=z9hG4bKPjd7Cotxkcc-
NpZ2C3mC7TmLWRF5K3LeB.;received=201.21.97.113;rport=40130.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>;tag=as188d7b05.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19116 INVITE.
Server: Asterisk PBX 1.6.2.7-1ubuntu1.1.
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO.
Supported: replaces, timer.
Require: timer.
Session-Expires: 1800;refresher=uas.
Contact: <sip:05199991234@174.129.243.191>.
Content-Type: application/sdp.
Content-Length: 277.
.
v=0.
o=root 202151605 202151605 IN IP4 174.129.243.191.
s=Asterisk PBX 1.6.2.7-1ubuntu1.1.
c=IN IP4 174.129.243.191.
t=0 0.
m=audio 19922 RTP/AVP 3 101.
a=rtpmap:3 GSM/8000.

a=rtpmap:101 telephone-event/8000.
a=fmtp:101 0-16.
a=silenceSupp:off - - - -.
a=ptime:20.
a=sendrecv.

#
U 201.21.97.113:40130 -> 10.209.199.8:5060
ACK sip:05199991234@174.129.243.191 SIP/2.0.
Via: SIP/2.0/UDP 201.21.97.113:40130;rport;branch=z9hG4bKPjBaovO4K-4IhtqeNa8bZYxTM1u2XJbP8z.
Max-Forwards: 70.
From: <sip:05199991234@voip.mbanton.net>;tag=0QAmBIc2j4qEp2ri0mXQ69o2rehEcKmf.
To: <sip:05199991234@voip.mbanton.net>;tag=as188d7b05.
Call-ID: oPrlrBPyezNzoBbnbcwbukmayaREKxT7.
CSeq: 19116 ACK.
Content-Length: 0.
.