UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

IMPLANTAÇÃO E CUSTOMIZAÇÃO DE VOZ SOBRE IP COM SOFTWARES LIVRES

FRANCIS GUSLINSKI

BLUMENAU
2008

2008/2-10

FRANCIS GUSLINSKI

IMPLANTAÇÃO E CUSTOMIZAÇÃO DE VOZ SOBRE IP COM SOFTWARES LIVRES

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Francisco Adell Péricas, Orientador

BLUMENAU
2008

2008/2-10

IMPLANTAÇÃO E CUSTOMIZAÇÃO DE VOZ SOBRE IP

COM SOFTWARES LIVRES

Por

FRANCIS GUSLINSKI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Mestre – Orientador, FURB
Membro:	Prof. Sérgio Stringari, Mestre – FURB
Membro:	Prof. Paulo Fernando da Silva, Mestre – FURB

Blumenau, 10 de fevereiro de 2009.

Dedico este trabalho a meus pais, que sempre me apoiaram em todos os momentos e especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelas oportunidades na vida e a saúde que possuo.

À minha família, que sempre me apoiou nas minhas decisões e nos meus problemas.

Aos meus amigos, pela distração durante o stress causado pelo tcc.

Ao meu orientador, Francisco Adell Péricas, por ter acreditado na conclusão deste trabalho.

Um livro é um mundo que fala, um surdo que responde, um cego que guia, um morto que vive.

Padre Antônio Vieira

RESUMO

Este trabalho apresenta a implantação de um sistema de voz sobre IP utilizando software livre, resultando na configuração de uma central telefônica VoIP e na criação de dois aplicativos destinados a adição de usuários e ao monitoramento ativo de chamadas. A ferramenta para adição de usuários é acessada via interface web facilitando seu uso e a adição de novos ramais SIP. O monitoramento ativo permite que se acompanhe em tempo real as ligações feitas e recebidas pelo usuário monitorado. Desta forma inclui-se a solução de algumas necessidades dentro da telefonia tradicional: a central VoIP economizando nas chamadas, facilitando a criação de novos ramais e o recurso monitorando, a baixo custo que é muito apreciado por diretores de empresas.

Palavras-chave: Asterisk. VoIP. SIP.

ABSTRACT

This paper presents a implementation of system voice over IP using free software, resulting in configuring a central VoIP and the creation of two applications for the addition of users and tracking of active calls. The tool for addition of users is accessed via web interface facilitating its use and addition of new SIP extensions. The asset tracking allows real-time monitor the calls made and received by the user monitored. On this form is included the solution of some needs within the traditional telephony: the central VoIP economy calls, facilitating the creation of new branches and use the low-cost monitoring solution that is much appreciated by directors of companies.

Key-words: Asterisk. VoIP. SIP.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

BCC – Curso de Ciências da Computação – Bacharelado

CDR – *Call Detail Record*

CODEC – COdificador/DECodificador

DSC – Departamento de Sistemas e Computação

FXO – *Foreign eXchange Office*

FXS – *Foreign eXchange Station*

GPL – *General Public License*

GSM – *Global System for Mobile comunications*

HTTP – *HyperText Transfer Protocol*

IAX – *Inter Asterisk eXchange*

IP – *Internet Protocol*

MOS – *Mean Opinion Score*

NAT – *Network Address Translation*

OSI – *Open Systems Interconnection*

PABX – *Private Automatic Branch eXchange*

PCM – *Pulse Code Module*

PSTN – *Public Switched Telephone Network*

QoS – *Quality of Service*

RTCP – *Real Time Control Protocol*

RTP – *Real Time Protocol*

SIP – *Session Initiation Protocol*

SMTP – *Simple Mail Transfer Protocol*

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

UA – *User Agents*

UAC – *User Agents Client*

UAS – *User Agents Server*

UML – *Unified Modeling Language*

VoIP – *Voice over Internet Protocol*

SUMÁRIO

1 INTRODUÇÃO

Prejudicado por uma telefonista, Strowger inventou uma central eletromecânica, que ficou conhecida como central de Strowger, passo a passo ou de dois movimentos e que se chama hoje de *Private Automatic Branch eXchange* (PABX) (ALENCAR, 2002, p. 2). Segundo Gonzalez (2007, p. 24), a invenção do dispositivo de discagem, que possibilitou automatizar as ligações ficou conhecida como central telefônica que permite efetuar ligações entre telefones internos sem intervenção manual, ou ainda telefonar e receber telefonemas da rede externa.

Logo depois surgiu a telefonia digital, que empregava a transmissão de comunicação de voz em sistemas digitais. Com o advento dos computadores utilizando sistemas digitais e a Internet foi possível juntar os três: PABX, computador e Internet. Formou-se a tecnologia *Voice over Internet Protocol* (VoIP), em 1995 em Israel, quando um grupo interessado no assunto conseguiu desenvolver um sistema que permitisse utilizar os recursos multimídia de um computador doméstico para iniciar conversas de voz através da Internet (ALVES, 2005).

A implantação de softwares VoIP em empresas vem crescendo muito, devido à redução de custos que provê. A necessidade (e, em alguns casos, o custo) é verdadeiramente o principal objetivo das invenções (MEGGELEN; SMITH; MADSON, 2005, p. vii).

Este trabalho trata da implantação e customização de softwares livres VoIP. O Asterisk será o principal software utilizado. A arquitetura bem planejada do Asterisk dá flexibilidade para se criar módulos customizados que expandem o sistema de telefonia, ou até mesmo que servem como substitutos para os módulos padrões (DEMPSTER; GOMILLION, 2005, p. 6).

Neste trabalho foi desenvolvida uma aplicação de plano de discagem para o Asterisk, um módulo para a *dialplan*, com a função de monitorar chamadas ativas de entrada e de saída de um determinado ramal, discando para o ramal monitor assim que detectar as chamadas. Para facilitar a manutenção de usuários, foi necessário neste trabalho desenvolver uma interface *web* específica.

O Asterisk é configurado para utilizar vários recursos demonstrando sua complexidade e também o funcionamento correto da aplicação de plano de discagem a ser criada. Recursos como *Voice Mail*, recepção de fax, sala de conferência e unidade de resposta audível serão demonstrados.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é implantar e customizar um sistema de VoIP com o Asterisk, criando um módulo para a *dialplan* que permita que um ramal seja monitorado em ligações de entrada e de saída e uma interface *web* para adição dos usuários.

Os objetivos principais do trabalho são:

a) monitorar ligações em tempo real de um determinado ramal;
b) permitir adição de novos usuários SIP via navegador *web.*

Os objetivos secundários alcançados através de customização do Asterisk:

a) configurar serviço de *Voice Mail* para canais *Session Initiation Protocol* (SIP);
b) interligar via protocolo *Inter Asterisk eXchange* (IAX) com outro Asterisk;
c) receber fax e enviá-los via correio eletrônico para um determinado endereço;
d) interligar com uma central telefônica via interface de hardware;
e) permitir captura de chamadas;
f) criar salas de conferência;
g) configurar unidade de resposta audível;
h) fazer bilhetagem das ligações em banco de dados;
i) configurar lógicas de contexto das ligações;

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos.

O capítulo 1 apresenta estrutura geral do trabalho, resumo, introdução, os objetivos específicos desse trabalho e a localização dos assuntos abordados no trabalho.

No capítulo 2 está disposta toda a fundamentação teórica, explicando o VoIP, o Asterisk, sua instalação, configuração e sua *Application Programming Interface* (API), os principais protocolos de comunicação como o SIP e o IAX, a importância do COdificador/DECodificador (CODEC) e os trabalhos correlatos.

O capítulo 3 descreve como foi configurado o Asterisk, como foi desenvolvido o módulo para *dialplan* e a interface *web* para manutenção de usuários. Também a descrição dos requisitos e especificações através dos diagramas de casos de uso, diagramas de classe e

diagrama de atividades, assim como todas as funcionalidades utilizadas.

No capítulo 4 são apresentadas as conclusões sobre o trabalho e as sugestões para extensões e trabalhos futuros.

2 VOIP

Voz sobre IP é uma tecnologia que permite a digitalização da voz para transmissão numa rede de computadores. Na situação tradicional, voz e dados trafegam por redes independentes, ambas otimizadas para atender as características de cada serviço. A tecnologia VoIP permite o tráfego de voz e dados na mesma rede abrindo as possibilidades para aplicações que integram voz em dados em um mesmo equipamento do usuário (FERREIRA; BRANDÃO, 2007, p. 2).

Inicialmente o VoIP foi desenvolvido para efetuar ligações telefônicas, ou seja, comunicação de voz entre dois microcomputadores conectados a Internet. Para isto utiliza-se o sistema de multimídia, saída de som e microfone, com um programa que gerencia toda a comunicação e os processos envolvidos, efetuando e recebendo chamadas (ALVES, 2005).

A transmissão de voz baseada em pacotes tem problemas pela forma que a fala acontece pois ela é totalmente incompatível com a forma pela qual o *Internet Protocol* (IP) transporta dados. Falar e escutar consiste em um fluxo de áudio que não pode ser interrompido: se os pacotes de dados que transportam a voz são perdidos ou chegam com um atraso maior que trezentos milissegundos, têm-se degradação na qualidade da conversa. (MEGGELEN; SMITH; MADSON, 2005, p. 110).

Conforme Gonzalez (2007, p. 30), a largura de banda é essencial para uma conversação VoIP. A largura de banda para a transmissão de voz depende de vários fatores e pode ser calculada facilmente de acordo com informações de amostragem e quantização da voz dependendo do algoritmo de compressão e descompressão. Para garantir que a conversação VoIP ocorra em tempo real, deve-se implantar regras de *Quality of Service* (QoS), que priorizam o tráfego.

O VoIP é uma tecnologia que permite a integração de duas ou mais centrais telefônicas através da Internet, sem a intermediação das operadoras de telefonia de longa distância. Uma solução interessante para interligar as redes de telefonia da matriz com as filiais ou parceiros reduzindo custos (GONZALEZ, 2007, p. 28).

Segundo Alves (2005), à popularização do serviço de banda larga, que atualmente é uma realidade para a maioria das pessoas e empresas, contribuiu para a convergência entre as redes de serviços sendo realizada através dos protocolos *Transmission Control Protocol/Internet Protocol* (TCP/IP), possibilitando agregar valores em uma ferramenta que já esta disponível, a Internet, popularizando assim o uso do VoIP.

2.1 ASTERISK

O Asterisk é um software que emula uma central privada de distribuição de chamadas, ou seja, uma central telefônica que também é conhecida como PABX (DEMPSTER; GOMILLION, 2005, p. 5). O nome “Asterisk” foi escolhido porque tanto era uma tecla de telefone comum, como também um símbolo curinga no Linux (por exemplo, `rm –rf *`) (MEGGELEN; SMITH; MADSON, 2005, p. vii).

Criado inicialmente por Mark Spencer, o Asterisk foi um projeto para ajudar no agendamento de suporte telefônico para a empresa que Mark possuía. Com o crescimento da economia Mark resolveu apostar no Asterisk como modelo de negócios e logo fez contato com Jim Dixon do projeto Zapata. No projeto Zapata foi desenvolvido um periférico com componentes eletrônicos básicos dispensando o sistema digital de processamento, que seria emulado via software pelo microprocessador, criando assim uma interface que comunica com a rede pública de telefonia. Juntos o Asterisk e o projeto Zapata revolucionaram a telefonia (SYNGRESS, 2007, p. 6).

Afirma Gonçalves (2005, p. 1) que o Asterisk é muito mais que um PABX padrão, pois como é um software livre e possui licença *General Public License* (GPL), seu código pode ser alterado para qualquer propósito necessário assim como para a criação de novos recursos desejados.

2.1.1 Instalação

Segundo Gonçalves (2005, p. 30), para instalar o Asterisk no Linux são necessários alguns pacotes de softwares instalados no sistema. Para utilização de periférico derivado do projeto Zapata e o recurso de conferência, será necessário instalar o pacote zaptel. As interfaces digitais E1 e T1 requerem o pacote libpri. Os pacotes kernel-headers, Makefile, autoconf, Bison, openssl, libc6, libasound, libnewt, libncurses5, gcc e zlib são pacotes de desenvolvimento do Linux necessários para compilar o Asterisk e devem ser instalados conforme a distribuição Linux utilizada. Por fim é necessário baixar o código fonte do Asterisk e os *addons* para comunicação com o MySQL. Os pacotes zaptel, libpri e o código fonte do Asterisk são encontrados no site da Digium (DIGIUM, 2008b). Para o funcionamento do fax deve instalar a biblioteca spandsp (SOFT-SWITCH, 2008).

Ainda segundo Gonçalves (2005, p. 41), a instalação dos pacotes deve seguir a seguir a ordem:

a) instalar os pacotes de desenvolvimento na distribuição;
b) compilar e instalar o pacote libpri;
c) compilar e instalar o pacote zaptel;
d) compilar e instalar o Asterisk;
e) compilar e instalar os *addons*.

2.1.2 Configuração

A configuração do Asterisk é feita toda através de arquivos textos que estão por padrão em `/etc/asterisk` com exceção do `/etc/zaptel.conf`, o qual guarda a configuração de placas analógicas e digitais.

Os arquivos de configurações e suas respectivas funcionalidades são (DEMPSTER; GOMILLION, 2005, p. 40):

a) `cdr_manager.conf`: este arquivo configura o *Call Detail Record* (CDR) para gerenciamento de chamadas;
b) `extconfig.conf`: com este arquivo pode-se optar por carregar as filas pelo mecanismo de banco de dados;
c) `extensions.conf`: este arquivo configura o comportamento do Asterisk, as rotas de discagem;
d) `features.conf`: este arquivo contém opções para estacionamento de chamadas, captura de ramais, captura de ligações de algum membro de dado grupo de captura;
e) `iax.conf`: este arquivo configura as conversas VoIP utilizando o protocolo IAX;
f) `indications.conf`: este arquivo configura alguns comportamentos do sistema de telefonia, como as cadências dos toques do telefone e tons, permitindo que disponibilize os sons que os usuários estão acostumados a ouvir, independente de seu país de origem;
g) `meetme.conf`: este arquivo de configuração define as salas de conferência. Pode-se também definir senhas para as conferências;
h) `rtp.conf`: este arquivo de configuração define as portas que serão utilizadas pelo *Real Time Protocol* (RTP);

i) `sip.conf`: este arquivo de configuração define os usuários do SIP e suas opções. Pode-se também definir opções globais para o SIP, como em que endereço de rede aguardar por conexões, qual porta utilizar e quais tempos de expiração serão utilizados;

j) `voicemail.conf`: este arquivo de configuração cria usuários de correio de voz e algumas opções globais para o Comedian Mail, o sistema de correio de voz do Asterisk;

k) `zapata.conf`: este arquivo configura os parâmetros da interface de telefonia Zapata. Será utilizado para configurar as placas da Digium. As placas da Digium são as que permitem a comunicação com a *Public Switched Telephone Network* (PSTN);

l) zaptel.conf: este arquivo de configuração define as portas *Foreign eXchange Office* (FXO) e *Foreign eXchange Station* (FXS) que fazem comunicação com os canais analógicos.

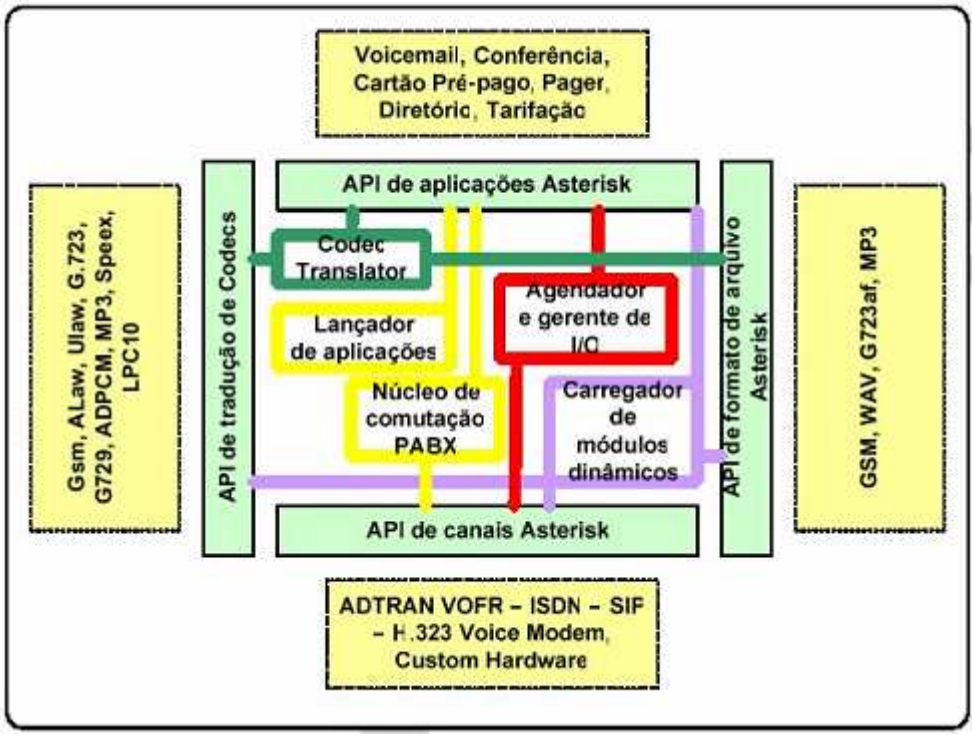
Em geral os arquivos de configuração do Asterisk têm as seções globais representadas entre colchetes ([global]), os parâmetros de configuração com sinal de igual (param=valor) e o símbolo sustenido (#) indica comentário, ou seja, o Asterisk ignora essa linha.

Para a configuração de canais analógicos deve-se definir a diferença entre os canais FXO e FXS. A diferença entre ambos é simplesmente qual fornece e qual recebe o tom de discagem, a porta FXO recebe e a porta FXS fornece o tom de discagem, sendo nomeadas assim pelo tipo físico de conexão que recebem. As portas são definidas pelas sinalizações que utilizam em oposição aos tipos físicos de portas que elas são. Uma porta física FXO deve receber sinalização FXS na configuração, isto porque a interface física recebendo o tom de discagem deve agir como um cliente. Na mesma linha de raciocínio uma porta física FXS deve receber sinalização FXO na configuração porque deve agir fornecendo tom de discagem, que é a sinalização FXO (MEGGELEN; SMITH; MADSON, 2005, p. 48).

Os canais analógicos FXS necessitam da definição de como os equipamentos requerem o tom de discagem. *Loop start, Ground start* e *Kewlstart* são os protocolos que podem ser definidos para a sinalização que requer o tom de discagem. O *Loop start* sinaliza fechando um curto-circuito, o *Ground start* aterrando um dos fios e o *Kewlstart* é o mesmo que o *Loop start* com uma inteligência maior conseguindo detectar desconexões do outro terminal (MEGGELEN; SMITH; MADSON, 2005, p. 50).

2.1.3 Application Programming Interface

O código fonte do Asterisk foi todo escrito em linguagem de programação C, para dar uma grande agilidade no sistema. Na Figura 1 mostra como o Asterisk foi dividido em módulos para permitir melhor gerenciamento e para criar aplicações sem a necessidade de alterar o núcleo do sistema. A documentação da API no sistema é gerada por uma ferramenta chamada DoxyGen, disponibilizada na *web* (DIGIUM, 2008d).



Fonte: Gonçalves (2005, p. 7).

Figura 1 – Arquitetura do Asterisk

Para programar no Asterisk existem algumas regras a serem seguidas, isso se quiser disponibilizar o código para a comunidade *OpenSource*. Estas regras são encontradas no Asterisk *Code Guidelines* as quais descrevem detalhes de uso da linguagem de programção C como a forma que deve ser utilizado o fechamento de blocos descrita no Quadro 1 e no Quadro 2, nomenclatura de funções e variáveis. Seguem algumas regras básicas (DIGIUM, 2008c):

a) não usar comentários estilo C++ (//);
b) todo código, nome de arquivos, comentários e funções devem estar em inglês;
c) funções e variáveis que não vão ser usadas fora do módulo devem ser declaradas estáticas;

d) funções que serão de uso global devem iniciar com `ast_` e ter o devido cabeçalho em `include/asterisk`;
e) não declare variáveis no meio dos blocos de programação;
f) quando ler inteiros com a função `scanf()` utilizar `%d`;
g) para manipulação de strings usar as funções da API em `include/asterisk/strings.h` como `ast_copy_string` que garante o caractere nulo no final da variável;
h) listas encadeadas definidas na API em `include/asterisk/linkedlists.h`;
i) alocações de memória utilizar a função `ast_calloc`.

```
if (foo) {
        bar();
} else {
        blah();
}
```

Fonte: Digium (2008c).

Quadro 1 – Bloco IF

```
switch (foo) {
case BAR:
        blah();
        break;
case OTHER:
        other();
        break;
}
```

Fonte: Digium (2008c).

Quadro 2 – Bloco CASE

Os cabeçalhos das funções da API do Asterisk se encontram no diretório `include/asterisk`. Segue abaixo uma lista de alguns dos cabeçalhos mais utilizados nas aplicações de plano de discagem (DIGIUM, 2008c):

a) `app.h`: funções comuns a todas aplicações, recepção de argumentos e parâmetros;
b) `audiohook.h`: funções relacionados ao áudio;
c) `channel.h`: funções de utilização dos canais de comunicação;
d) `dial.h`: funções de discagem;
e) `features.h`: funções de estacionamento e captura de chamadas;
f) `file.h`: funções de manipulação de arquivos;
g) `linkedlists.h`: macros para utilização na alocação de memória;
h) `logger.h`: funções que habilitam processo de *logging;*
i) `module.h`: funções relacionadas a carga e descarga de módulos;
j) `monitor.h`: funções para monitorar os canais;

k) `options.h`: opções do programa principal;

l) `paths.h`: caminhos definidos no arquivo `asterisk.conf`;

m) `strings.h`: funções de manipulação de caracteres;

n) `utils.h`: funções diversas como alocação de memória e criação de threads.

Uma nova aplicação do plano de discagem deve ser criada no diretório `apps/directory`. A novo módulo da *dialplan* deve conter um conjunto básico de funções, estruturas e variáveis em seu interior como definido em `apps/app_skel.c`. Inicialmente no código de um módulo da *dialplan* tem-se toda a parte de licença e autor, seguido da inclusão dos cabeçalhos básicos, a chamada da macro `ASTERISK_FILE_VERSION(__FILE__, "$Revision: 40722 $")` que identifica o nome do arquivo e a versão do mesmo e por fim as variáveis obrigatórias `*app`, `*synopsis` e `*descript` que trazem informação básica da *dialplan.* Abaixo o Quadro 3 representa o conteúdo inicial que uma *dialplan* deve conter segundo DIGIUM(2008d).

```
/*
 * Informações de licença e autor
 *
 */

/*** MODULEINFO
      <defaultenabled>no</defaultenabled>
 ***/

#include "asterisk.h"

ASTERISK_FILE_VERSION(__FILE__, "$Revision: 40722 $")

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#include "asterisk/file.h"
#include "asterisk/logger.h"
#include "asterisk/channel.h"
#include "asterisk/pbx.h"
#include "asterisk/module.h"
#include "asterisk/lock.h"
#include "asterisk/app.h"

static char *app = "Skel";
static char *synopsis =
"Skeleton application.";
static char *descrip = "This application is a template to build
other applications from.\n"
 " It shows you the basic structure to create your own Asterisk
applications.\n";
```

Fonte: Digium (2008d).

Quadro 3 – Código inicial de um módulo da *dialplan*

No Quadro 4 é exposto como o módulo de uma *dialplan* deve implementar a checagem e recepção de parâmetros. É indicado utilizar enumerações para que o compilador possa otimizar melhor o código.

```
enum {
        OPTION_A = (1 << 0),
        OPTION_B = (1 << 1),
        OPTION_C = (1 << 2),
} option_flags;

enum {
        OPTION_ARG_B = 0,
        OPTION_ARG_C = 1,
        /* This *must* be the last value in this enum! */
        OPTION_ARG_ARRAY_SIZE = 2,
} option_args;

AST_APP_OPTIONS(app_opts,{
        AST_APP_OPTION('a', OPTION_A),
        AST_APP_OPTION_ARG('b', OPTION_B, OPTION_ARG_B),
        AST_APP_OPTION_ARG('c', OPTION_C, OPTION_ARG_C),
});
```

Fonte: Digium (2008d).

Quadro 4 – Código de recepção de argumentos de um módulo da *dialplan*

Abaixo o Quadro 5 demonstra como devem ser as funções de carga e descarga do módulo da *dialplan* e a macro que registra a informação desta.

```
static int unload_module(void)
{
        int res;
        res = ast_unregister_application(app);
        return res;
}

static int load_module(void)
{
        return ast_register_application(app, app_exec, synopsis,
descrip);
}

AST_MODULE_INFO_STANDARD(ASTERISK_GPL_KEY, "Skeleton (sample)
Application");
```

Fonte: Digium (2008d).

Quadro 5 – Código de carga e descarga do módulo da *dialplan*

A execução do código do módulo da *dialplan* é iniciada na função `app_exec()` que recebe como parâmetro inicial um ponteiro para o canal que está sendo utilizado e um ponteiro para uma estrutura de dados que contém os parâmetros passados para o módulo. Nela é também feito a checagem dos parâmetros recebidos e programadas todas as funcionalidades que devem executar. No Quadro 6 é exibido o exemplo.

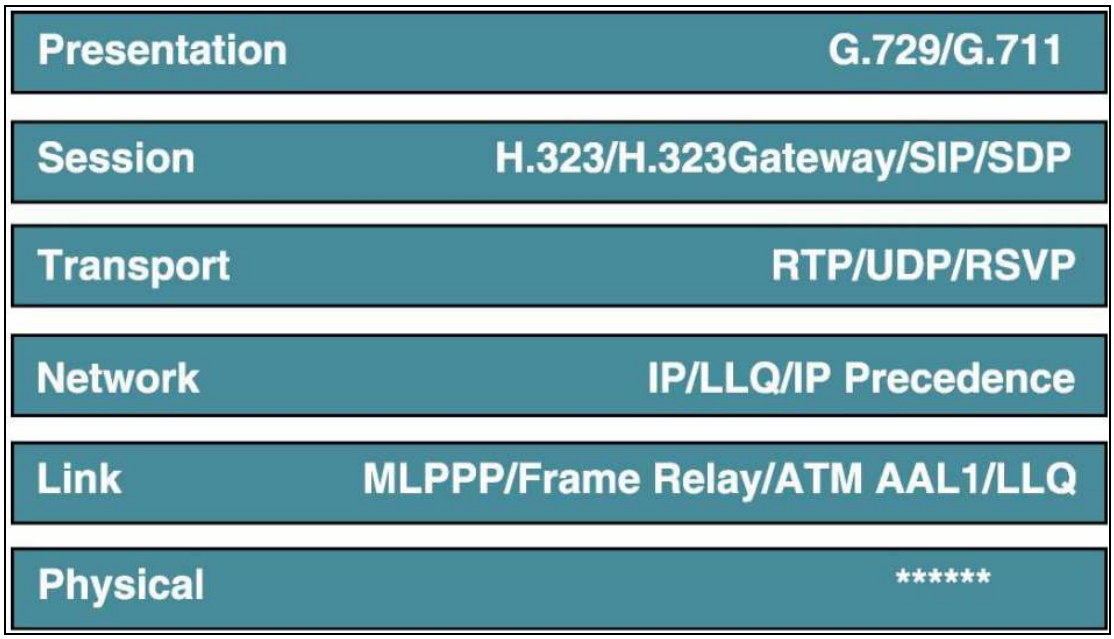
```
static int app_exec(struct ast_channel *chan, void *data)
{
        int res = 0;
        struct ast_flags flags;
        struct ast_module_user *u;
        char *parse, *opts[OPTION_ARG_ARRAY_SIZE];
        AST_DECLARE_APP_ARGS(args,
                AST_APP_ARG(dummy);
                AST_APP_ARG(options);
        );

        if (ast_strlen_zero(data)) {
                ast_log(LOG_WARNING, "%s requires an argument
(dummy|[options])\n", app);
                return -1;
        }

        u = ast_module_user_add(chan);

        /* Do our thing here */

        /* We need to make a copy of the input string if we are
going to modify it! */
        parse = ast_strdupa(data);

        AST_STANDARD_APP_ARGS(args, parse);

        if (args.argc == 2)
                ast_app_parse_options(app_opts, &flags, opts,
args.options);

        if (!ast_strlen_zero(args.dummy))
                ast_log(LOG_NOTICE, "Dummy value is : %s\n",
args.dummy);

        if (ast_test_flag(&flags, OPTION_A))
                ast_log(LOG_NOTICE, "Option A is set\n");

        if (ast_test_flag(&flags, OPTION_B))
                ast_log(LOG_NOTICE, "Option B is set with : %s\n",
opts[OPTION_ARG_B] ? opts[OPTION_ARG_B] : "<unspecified>");

        if (ast_test_flag(&flags, OPTION_C))
                ast_log(LOG_NOTICE, "Option C is set with : %s\n",
opts[OPTION_ARG_C] ? opts[OPTION_ARG_C] : "<unspecified>");

        ast_module_user_remove(u);

        return res;
}
```

Fonte: Digium (2008d).

Quadro 6 – Código principal da *dialplan*

2.2 PROTOCOLOS VOIP

Conforme Madeira (2007, p. 24), os protocolos VoIP são responsáveis diretamente pelo estabelecimento, finalização, negociação de mídia e supervisão da qualidade da chamada. SIP, IAX, H323, MGCP, Megaco e RTP são protocolos comuns nessa categoria. Na Figura 2 abaixo é colocado como se apresentam os protocolos citados acima no modelo *Open Systems Interconnection* (OSI).



Fonte: Gonçalves (2005, p. 106).

Figura 2 – Protocolos VoIP no modelo OSI

O protocolo RTP é o protocolo de transporte mais utilizado para a transferência de mídia e o protocolo *Real Time Control Protocol* (RTCP) para controle dessa transferência. A comunicação via RTP é feita através do *User Datagram Protocol* (UDP) por causa do seu *payload* menor.

Atualmente é comum a utilização dos protocolos H.323 e SIP em programas e equipamentos VoIP, porém o que está conquistando mercado é o IAX por sua facilidade em trabalhar com firewalls e *Network Address Translation (* NAT).

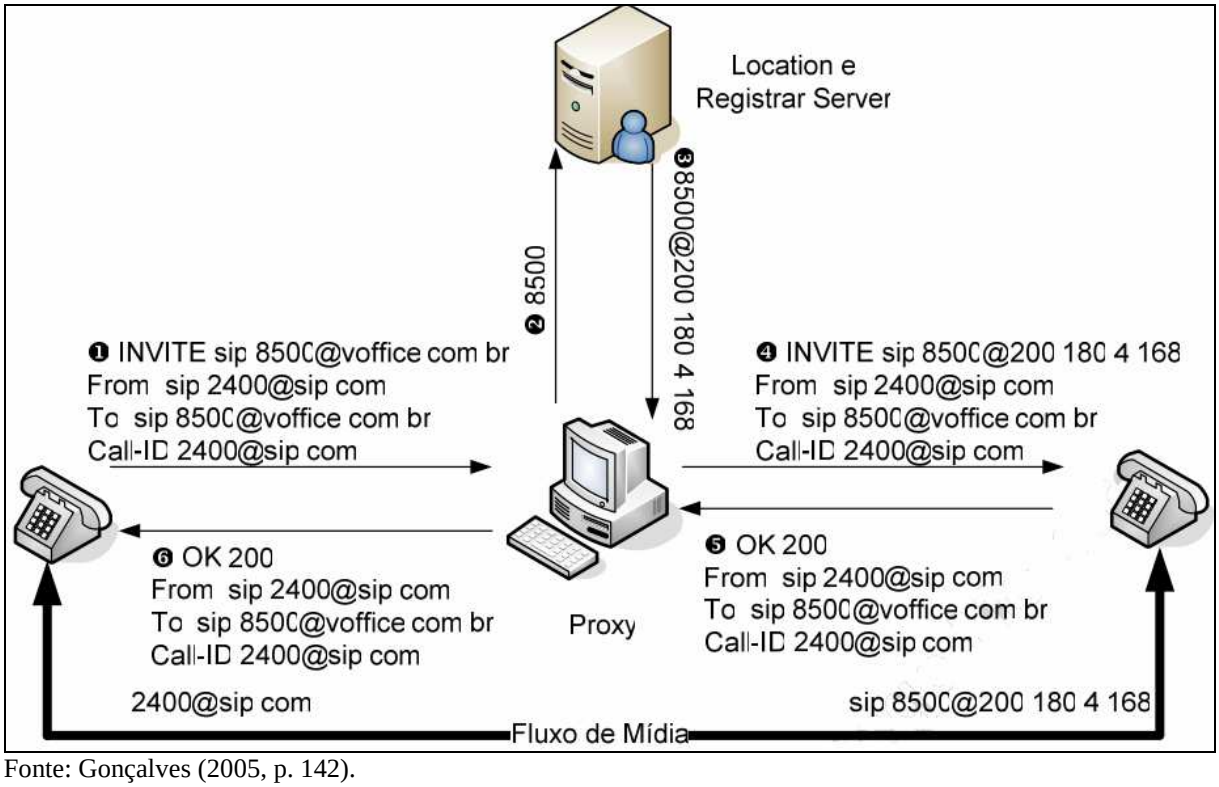
2.2.1 SIP

O SIP é uma camada de aplicação que controla como o protocolo estabelece, modifica e termina sessões multimídia ou uma chamada. Essas sessões de multimídia incluem

conferência de vídeo, telefonia via internet e aplicações similares (RFC 2543, 1999, p. 7).

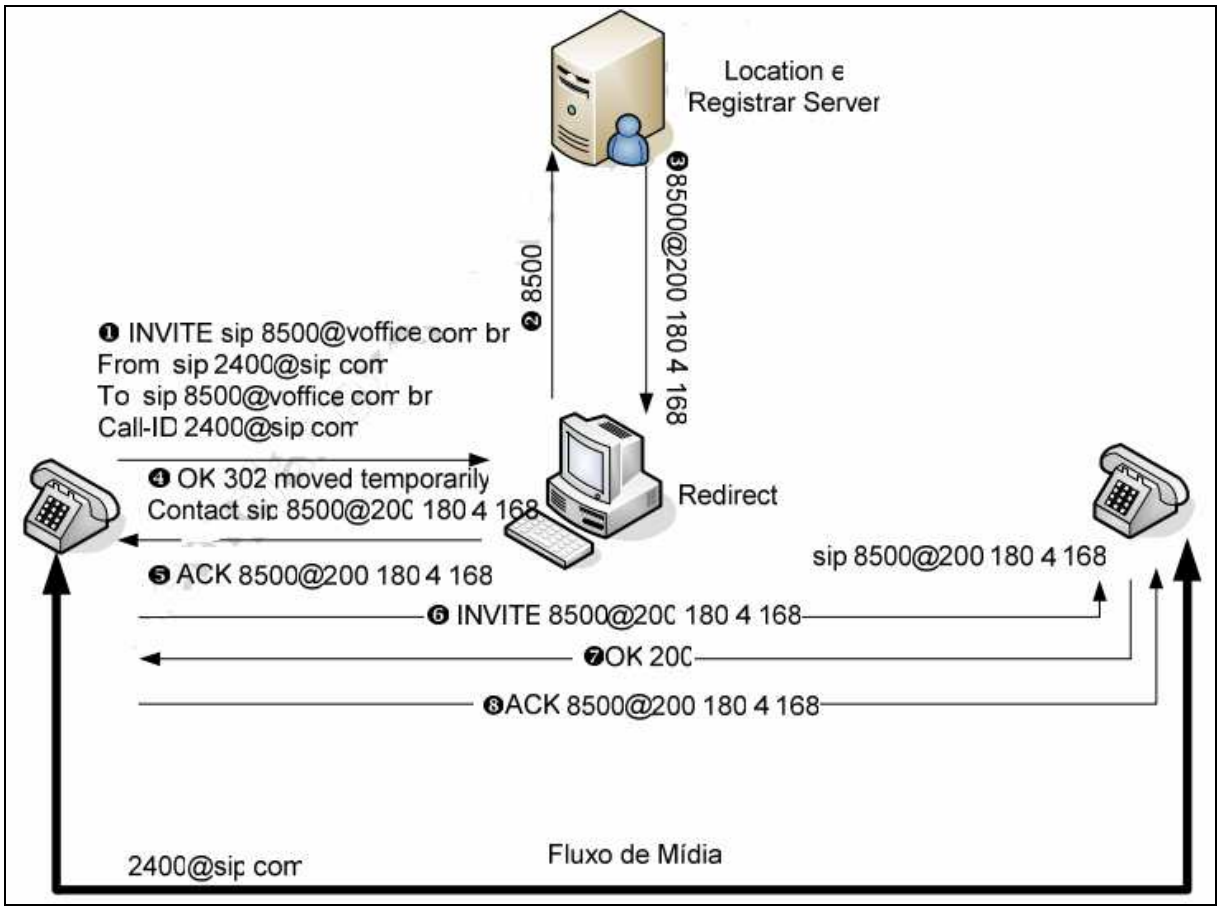
Existem cinco elementos centrais na arquitetura SIP. Algumas das definições de um ou mais elementos, se consolidam em um ou mais (MADEIRA, 2007, p. 26):

a) *User Agents* (UA): é um cliente que inicia uma conexão SIP, formado pelo *User Agent Client* (UAC) e de um *User Agent Server* (UAS). O UAC é quem gera as requisições e o UAS é quem as responde;
b) *proxy server*: recebe as conexões dos *User Agents* e transfere para o próximo servidor SIP se o UAC não está em sua administração. Possui as funcionalidades de autorização, autenticação, controle de acesso à rede e roteamento de chamadas (Figura 3);
c) *register server*: mantém as informações sobre os UAs e autentica-os. Normalmente está localizado no mesmo servidor que o *proxy server*;
d) *redirect server*: recebe os pedidos de conexão do emissor e retorna ao mesmo dados do destino ao invés de completar a chamada (Figura 4);
e) *location server*: é usado pelo *redirect server* ou pelo *proxy server* para identificar as possíveis localizações dos destinos chamados. Normalmente está localizado junto com o *register server*.



Fonte: Gonçalves (2005, p. 142).

Figura 3 – Proxy Server



Fonte: Gonçalves (2005, p. 142).

Figura 4 – Redirect Server

Segundo Hersent, Gurle e Petit (2002, p. 119), um cliente SIP chama outro ponto final enviando uma mensagem de pedido chamado `invite`. O `invite` indica o terminal de destino com quem deve estabelecer a conexão de mídia, assim como a codificação a ser usada para a mídia. O destino pode negar a codificação pedida e enviar outra codificação ao cliente de origem. Se o mesmo aceitar é estabelecida uma comunicação através do RTP.

Conforme Meggelen, Smith e Madson (2005, p. 111), o SIP ganhou popularidade por ser um protocolo simples com sintaxe similar a outros protocolos como o *HyperText Transfer Protocol* (HTTP) e o *Simple Mail Transfer Protocol* (SMTP). O maior problema do SIP é transportar as transações através de NAT e firewalls. Como ele tem as informações de endereçamento encapsuladas em seus conjuntos de dados, os firewalls e NATs não conseguem obter essa informação visto que trabalham em uma camada de rede inferior. Como o SIP utiliza portas diferentes da inicial conectada para a transferência do conteúdo multimídia, os firewalls e NATs acabam bloqueando essa conexão.

2.2.2 IAX

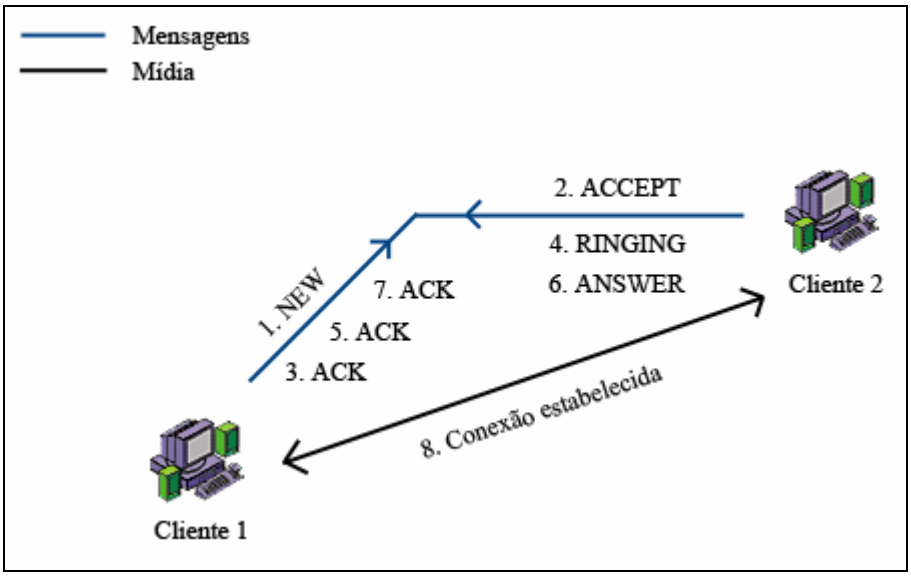
O Inter-Asterisk eXchange (IAX) é um protocolo de controle e transmissão de mídia através de redes IP. Foi desenvolvido por Mark Spencer e Frank Miller como uma alternativa aos protocolos já existentes, como o SIP e o H.323. Assim como o SIP, o IAX pode ser utilizado para qualquer tipo de sessão, seja ela de voz, vídeo, ambos ou outras (DÓRIA; JÚNIOR; OLIVEIRA, 2006).

Segundo Meggelen, Smith e Madson (2005, p. 111) o IAX foi desenvolvido para funcionar com dispositivos que façam uso de NAT utilizando uma única porta de número 4569 no protocolo UDP para sinalizar e transmitir a mídia. O IAX está na versão dois sendo muitas vezes chamado de IAX2, este otimizado para transporte de voz e interligação entre servidores Asterisk, suportando vários fluxos de áudio em um único datagrama de dados reduzindo assim uso de banda e processamento.

Conforme Dória, Júnior e Oliveira (2006), o IAX utiliza pacotes binários e três tipos de frames:

a) FULL: usado para dados de mídia ou sinalização que requerem confirmação;
b) MINI: usado para dados de mídia ou sinalização que não requerem confirmação;
c) META: usado pra transmissão de vídeos e agrupamento de streams.

Segue abaixo a Figura 5 ilustrando o funcionamento de uma conexão utilizando IAX.



Fonte: Dória, Júnior e Oliveira (2006).

Figura 5 – Conexão IAX

2.3 CODEC

Um Codificador/DECodificador (CODEC) é um modelo matemático para codificar digitalmente as informações analógicas como o áudio e o vídeo. O termo CODEC também pode ser relacionado à COmpressão/DEsCompressão pelo fato de comprimir a informação digitalizada. A finalidade dos algoritmos codificadores é representar os sinais de áudio e ou vídeo com a quantidade mínima de bits atingindo um balanço entre eficiência e qualidade (OZVOIP, 2008).

Conforme Hersent, Gurle e Petit (2002, p. 119), a qualidade de voz de um CODEC é medida através de notas *Mean Opinion Score* (MOS), essa nota é atribuída particularmente a cada CODEC por um grupo de ouvites que escutam várias amostras gravadas, a nota pode variar de 1 a 5.

Meggelen, Smith e Madson (2005, p. 114) definiram uma tabela de referência para os CODECs mais comuns ao Asterisk, segue abaixo a Tabela 1.

Tabela 1 – Referência na taxa de bits de dados dos CODECs

CODEC	**Taxa de Bits em kbps**
G.711	64
G.7216	16, 24 ou 32
G.723.1	5,3 ou 6,3
G.729A	8
GSM	13
iLBC	13,3 ou 15,2
Speex	Entre 2,15 e 22,4

O G.711 é o CODEC básico que implementa o método mais comum de codificar o áudio, o formato *Pulse Code Module* (PCM), do qual os outros CODECs derivam. Também é conhecido popularmente como alaw e µlaw. O formato µlaw usado na América do Norte utiliza um método de compactação diferente do alaw que é utilizado no resto do mundo.

G.726 é um CODEC parecido com o G.711, porém ele utiliza metade da largura de banda por trabalhar mandando somente informação da diferença do pacote anterior enviado. Era antes chamado de G.721, que atualmente é considerado obsoleto.

Sendo o G.723.1 um CODEC desenvolvido para utilizar baixo consumo de banda é também um CODEC proprietário, sendo necessário licença para seu uso. Anteriormente era conhecido somente como G.723, mas sua evolução forçou essa derivação.

O CODEC G.729A é o mais recomendado pela qualidade de som e largura de banda, porém para atingir tamanha taxa de compressão ele requer uma alta quantidade de processamento. O G729A é um CODEC proprietário, sendo necessário licença para seu uso.

O Asterisk considera o CODEC *Global System for Mobile comunications* (GSM) a melhor escolha de CODEC devido a seu baixo consumo de processamento se comparado a sua utilização de banda, porém é considerado com qualidade de som pouco inferior ao G.729A.

O iLBC e o Speex são CODECs que trabalham com taxa de banda variável fazendo alto consumo de processamento, ainda sendo o iLBC com uma licença particular e o Speex com licença BSD.

2.4 TRABALHOS CORRELATOS

O Asterisk possui muitas *dialplans* implementadas e as que assemelham-se aos objetivos deste trabalho são: `app_dial`, `app_monitor` e `app_chanspy`. Existe muitas interfaces *web* comerciais e não comerciais para configuração das extensões, ramais e manutenção de usuários, como o AsteriskNOW (DIGIUM, 2008a) e o VoiceOne (VOICEONE, 2008).

A `app_dial` é uma das principais *dialplans* do Asterisk sendo ela responsável pela discagem para ramais. A `app_monitor` permite que todas as chamadas naquele canal sejam gravadas em arquivo. Já a `app_chanspy` recebe uma chamada em um determinado ramal feita pelo monitor. Recebida a chamada, começa monitorar o canal programado se este estiver em ligação, utilizando para capturar a voz dos outros canais a API do audiohook, que será usado na criação da aplicação módulo da *dialplan* desse trabalho.

O AsteriskNOW é um projeto da própria Digium com o objetivo de por o sistema para funcionar em algumas horas. Para quem já tem algum conhecimento do projeto pode ser bem útil ao configurar alguns itens. Permite cadastrar usuários SIPs, criar rotas de ligações e configurar *Voice Mail,* dentre outros recursos.

Existem interfaces *web* bem mais elaboradas. A VoiceOne por exemplo facilita muito a configuração, porém todo esse esforço para tornar mais fácil a configuração tem um custo que a VoiceOne tenta suprir cobrando suporte (VOICEONE, 2008). Com o VoiceOne já vem inclusive um cliente SIP em Java para *web.*

3 DESENVOLVIMENTO

A implantação e customização do Asterisk nesse trabalho permite que um computador comum trabalhe como uma central telefônica digital, incluindo a funcionalidade de monitorar as ligações do usuário desejado.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A implantação e customização do Asterisk deverá:

a) realizar chamadas para usuários SIP (Requisito Funcional - RF);
b) receber chamadas da rede PSTN para usuários SIP (RF);
c) receber chamadas SIP para usuários SIP (RF);
d) comunicar com o servidor da filial via IAX (RF);
e) ter uma caixa *Voice Mail* para usuários SIP (RF);
f) receber fax (RF);
g) enviar por correio eletrônico fax recebidos (RF);
h) permitir captura de chamadas (RF);
i) ter uma unidade de resposta audível (RF);
j) fazer bilhetagem das chamadas em banco de dados (RF);
k) disponibilizar uma interface *web* para adição de usuários (RF);
l) implementar uma aplicação *dialplan* que monitore ligação SIP e ligue para um monitor (RF);
m) ter sala de conferência para usuários SIP (RF);
n) utilizar o banco de dados MySQL para a bilhetagem (Requisito Não Funcional - RNF);
o) utilizar sistema operacional Slackware Linux para o Asterisk (RNF);
p) utilizar linguagem C e a API do Asterisk para implementação de um módulo para a *dialplan* (RNF);
q) utilizar PHP para implementação da interface *web* (RNF).

3.2 ESPECIFICAÇÃO

Para especificar o sistema de manutenção de usuários e a nova *dialplan*, foi utilizada a orientação a objetos representada através da *Unified Modeling Language* (UML). A ferramenta utilizada para gerar os diagramas foi o Enterprise Architect.

3.2.1 Modelo conceitual da base de dados

Os dados de cadastro dos usuários SIP e bilhetagem das ligações efetuadas ficam guardados no MySQL, na tabela sip_buddies os usuários SIP e bilhetagem na tabela cdr (Figura 6).

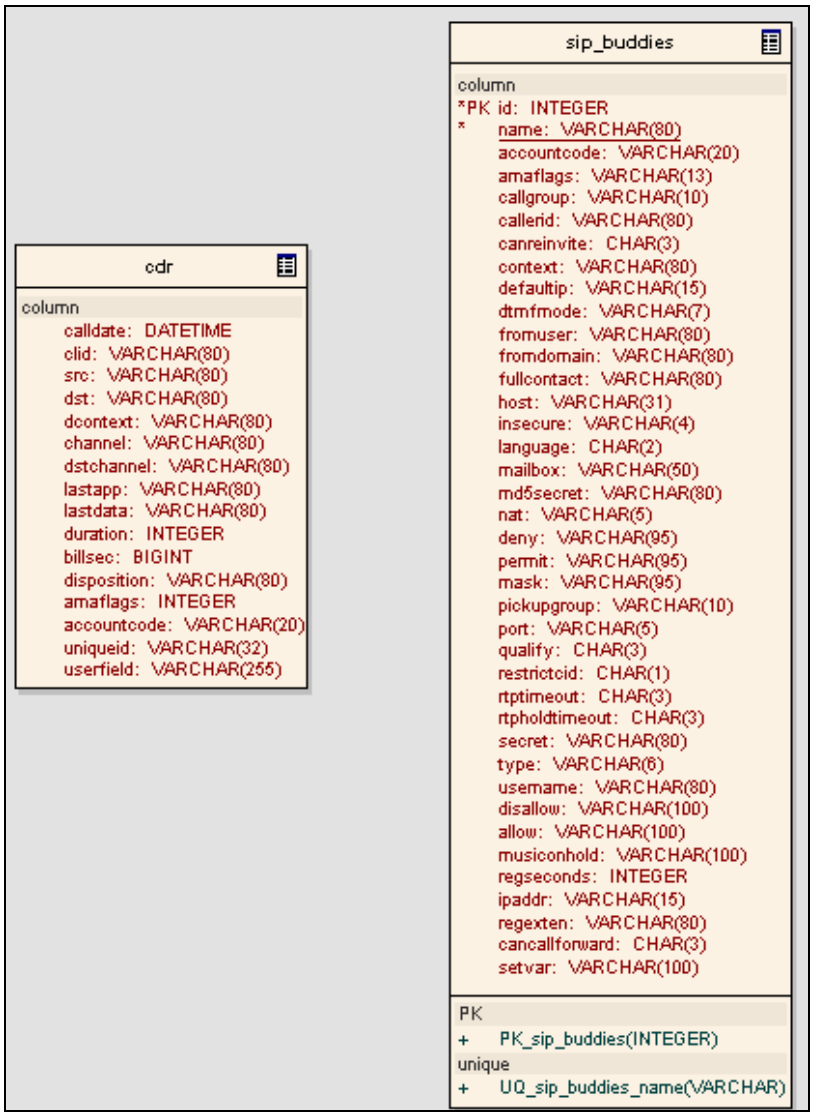


Figura 6 – Modelos de dados Asterisk

3.2.2 Diagrama de Atividades

O diagrama de atividades foi utilizado para modelar o módulo da *dialplan* criada, por se tratar de um digrama que mostra o fluxo do processamento de uma atividade para outra, para representar o processamento procedural da linguagem C.

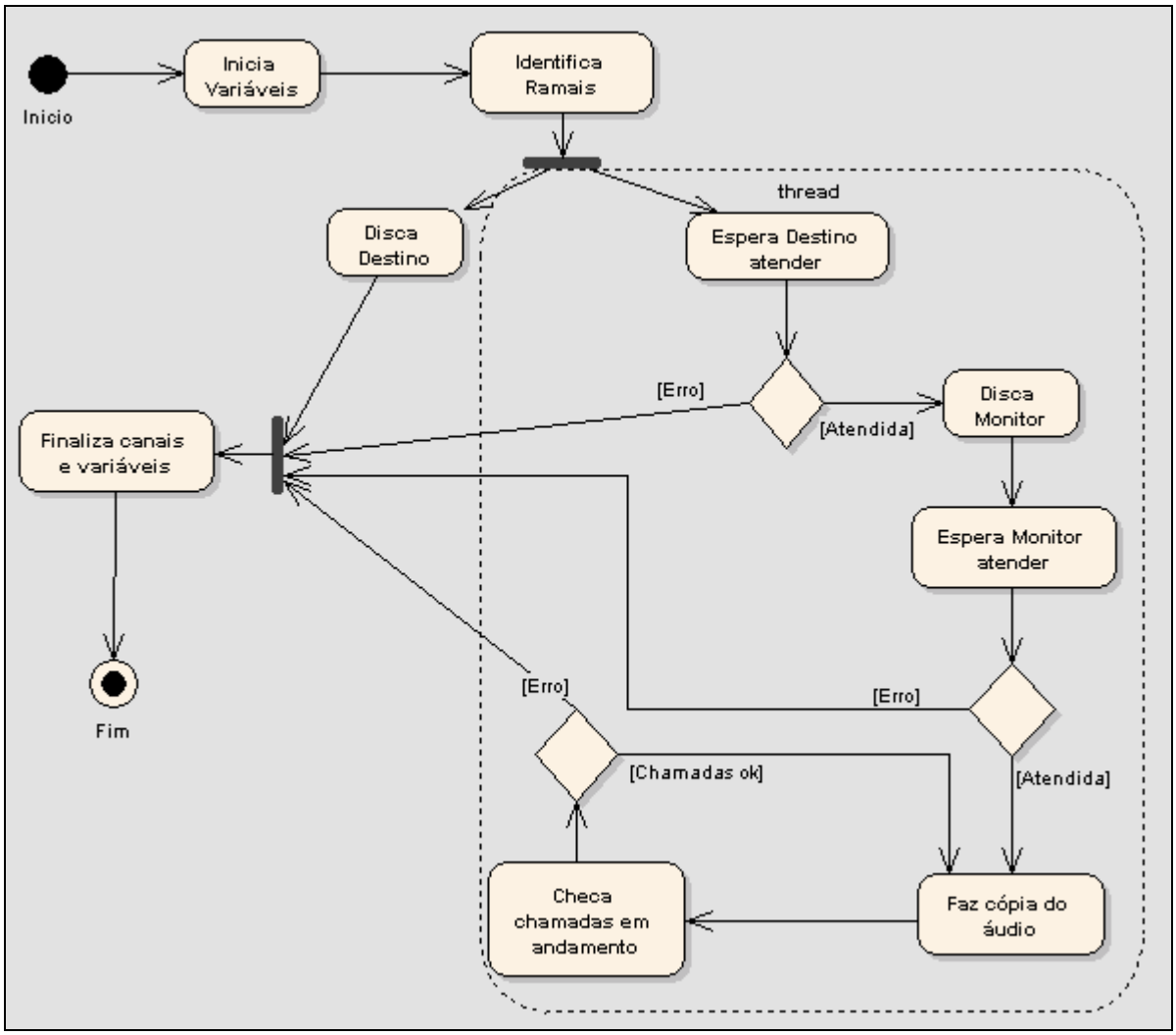


Figura 7 – Diagrama de atividades do módulo SpyDial

Na figura 7 demonstra-se o fluxo das atividades que o módulo SpyDial na *dialplan* vai executar quando inicializada:

a) inicia varáveis: inicialização da aplicação módulo e checagem de parâmetros;
b) identifica ramais: retira dos parâmetros os números do ramal e o monitor;
c) disca destino: inicia o processo de discagem para o destino original da chamada;
d) espera destino atender: espera o destino atender a chamada, caso o destino não atender ou houver algum erro na chamada, termina a *thread*;
e) disca monitor: disca para o monitor do ramal;

f) espera monitor atender: espera o monitor atender a chamada, caso o monitor não atender ou houver algum erro na chamada, termina a *thread*;
g) faz cópia do audio: executa o processo de cópia do áudio da chamada estabelecida entre o chamador e o destino para o monitor ouvir a conversa;
h) checa chamadas em andamento: executa a checagem se nenhuma das partes envolvidas na ligação desligou, caso sim termina vai para o fim do processo;
i) finaliza canais e variáveis: desliga todos o canais envolvidos na ligação e libera todos os recursos que foram utilizados na processo.

3.2.3 Diagrama de Casos de Uso

O diagrama de casos de uso representa as ações que o administrador do sistema deseja executar na interface *web*(Figura 8).

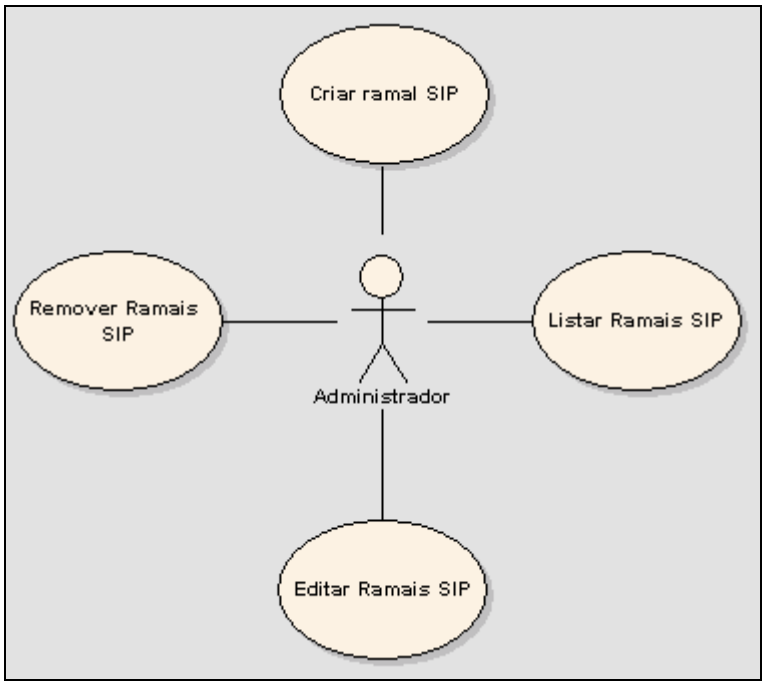


Figura 8 – Diagrama de casos de uso do Administrador

3.2.4 Diagrama de classes

O diagrama de classes representa a estrutura e relações das classes que modelam os objetos utilizados para construir a interface *web* que dará manutenção aos usuários SIP.

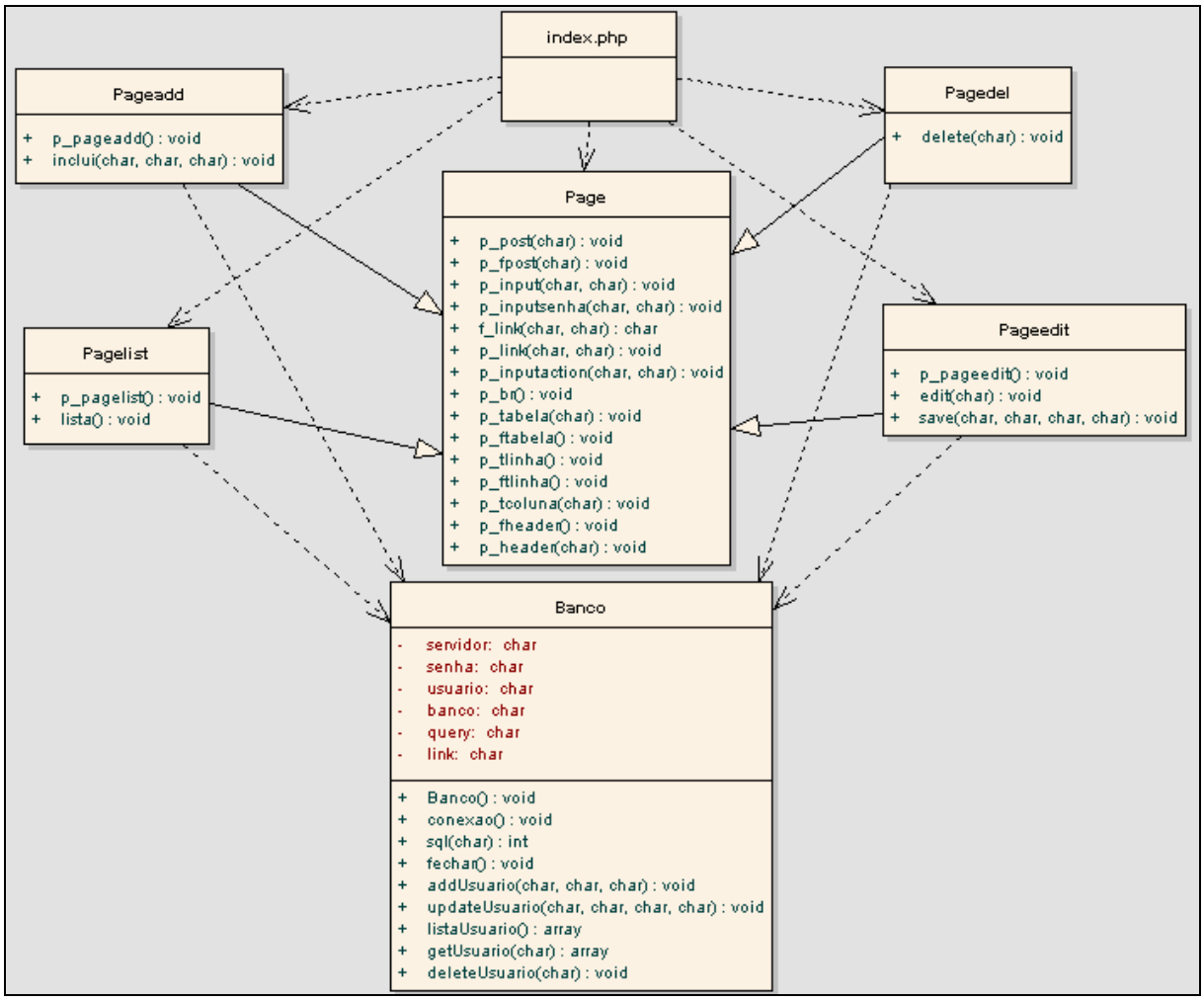


Figura 9 – Diagrama de classes da interface *web*

Segue abaixo a descrição das classes representadas na figura 9:

a) classe `Banco`: classe responsável por toda a comunicação com o banco de dados MySQL;

b) classe `Page`: classe responsável pelos comandos da linguagem HTML;

c) classe `Pageadd`: classe responsável pela adição do usuário;

d) classe `Pagelist`: classe responsável pela listagem dos usuários cadastrados;

e) classe `Pageedit`: classe responsável pela edição dos usuários cadastrados;

f) classe `Pagedel`: classe responsável pela remoção dos usuários cadastrados;

g) `index.php`: não é uma classe e é responsável pela inicialização da página de cadastros.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para a criação da interface *web* para manutenção de usuários SIP, foi utilizada a linguagem de programação PHP. Para guardar a configuração dos usuários SIP foi usado o banco de dados MySQL.

A programação da nova *dialplan* foi desenvolvida na linguagem C utilizando-se um editor de texto comum, o VIM.

3.3.2 Implementação do sistema

Para resolver os requisitos do problema tratado começou-se com a instalação do Linux Slackware. Tendo instalado o Linux deve-se atentar aos requisitos de pacotes descritos no seção 2.1.1.

3.3.2.1 Interface web

A criação da interface *web* necessitou do funcionamento do servidor *web* Apache e PHP com suporte ao banco de dados MySQL.

O arquivo `Page.php` tem as funções de uso geral que imprimem códigos HTML utilizando-se da função `echo` do PHP (Quadro 7).

```
<?
class Page {
        /*
         * Imprime formulario html
         */
        public function p_post($pagina) {
                echo "<form action = \"$pagina\" method =
\"post\">";
                echo "<br>";
        }
        public function p_fpost($botao) {
                echo "<input type = \"submit\" value =
\"$botao\"/>";
                echo "</form>";
        }
.....
?>
```

Quadro 7 – Trecho de código do `Page.php`

A página inicial é tratada pelo `index.php` instanciando todos os objetos necessários para a aplicação e também tratando os métodos `GET` e `POST` do HTML (Quadro 8).

```
<?
include("Page.php");
include("Pageadd.php");
include("Pagelist.php");
include("Pageedit.php");
include("Pagedel.php");

$menu = new Page();
$menu->p_header("Menu");
$menu->p_link("index.php?action=add","Adicionar");
$menu->p_br();
$menu->p_link("index.php?action=list","Listar");
$menu->p_br();

if(isset($_GET["action"]))
{
        switch($_GET["action"]) {
        case "add":
                $oPage_add = new Pageadd();
                $oPage_add->p_pageadd();
                break;
        case "list":
                $oPage_list = new Pagelist();
                $oPage_list->lista();
                $oPage_list->p_pagelist();
                break;
        case "edit":
                $oPage_edit = new Pageedit();
                $oPage_edit->edit($_GET["name"]);
                $oPage_edit->p_pageedit();
                break;
        case "del":
                $oPage_del = new Pagedel();
                $oPage_del->delete($_GET['name']);
                break;
        }
}
```

```
elseif (isset($_POST["action"])) {
        switch($_POST["action"]) {
        case "add":
                $oPage_add = new Pageadd();
                $oPage_add-
>inclui($_POST["ramal"],$_POST["nome"],$_POST["senha"]);
                break;
        case "edit":
                $oPage_edit = new Pageedit();
                $oPage_edit-
>save($_POST["ramal"],$_POST['novoRamal'],$_POST["nome"],$_POST["senha"]);
                break;

        }
}
$menu->p_fheader();
?>
```

Quadro 8 – Código do `index.php`

Todo o código responsável pela comunicação com o banco de dados está no arquivo `Banco.php`.

3.3.2.2 Instalação do Asterisk

Segue no Quadro 9 os passos de instalação da `libpri`.

```
root@ast1:~# tar -zxf libpri-1.4-current.tar.gz
root@ast1:~# cd libpri-1.4.7
root@ast1:~/libpri-1.4.7# make
root@ast1:~/libpri-1.4.7# make install
```

Quadro 9 – Instalação da `libpri`

A instalação dos *drivers* Zapata é descrita no Quadro 11.

```
root@ast1:~# tar -zxf zaptel-1.4-current.tar.gz
root@ast1:~# cd zaptel-1.4.12.1
root@ast1:~/zaptel-1.4.12.1# ./configure
root@ast1:~/zaptel-1.4.12.1# make
root@ast1:~/zaptel-1.4.12.1# make install
```

Quadro 10 – Instalação dos *drivers* Zapata

Todos esses processos de instalação não devem gerar nenhum erro, caso tenha gerado erros revise a sua instalação do Linux. Na instalação do Asterisk também instalou-se os exemplos de configuração e a documentação (Quadro 11).

```
root@ast1:~# tar -zxf asterisk-1.4-current.tar.gz
root@ast1:~# cd asterisk-1.4.22
root@ast1:~/asterisk-1.4.22# ./configure
root@ast1:~/asterisk-1.4.22# make
root@ast1:~/asterisk-1.4.22# make install
root@ast1:~/asterisk-1.4.22# make samples
root@ast1:~/asterisk-1.4.22# make progdocs
```

Quadro 11 – Instalação do Asterisk

Instalação dos `addons` para funcionamento com banco de dados MySQL (Quadro 12).

```
root@ast1:~# tar -zxf asterisk-addons-1.4-current.tar.gz
root@ast1:~# cd asterisk-addons-1.4.7
root@ast1:~/asterisk-addons-1.4.7# ./configure
root@ast1:~/asterisk-addons-1.4.7# make
root@ast1:~/asterisk-addons-1.4.7# make install
```

Quadro 12 – Instalação dos `addons`

3.3.2.3 Configuração de autenticação via MySQL e comunicação entre ramais SIP

Para funcionar a autenticação via MySQL dos usuários SIP, deve ser alterado o arquivo `extconfig.conf` (Quadro 13).

```
[settings]
sipusers => mysql,asterisk,sip_buddies
sippeers => mysql,asterisk,sip_buddies
```

Quadro 13 – Autenticação via MySQL

O campo `mysql` indica em que base de dados deve autenticar, seguido do nome da base de dados a utilizar e o nome da tabela.

O arquivo `sip.conf` deve setar o contexto das ligações SIP por padrão para `incoming`, que vai ser utilizado nas extensões (Quadro 14).

```
[general]
context=incoming
```

Quadro 14 – Configuração do SIP

Todo usuário adicionado na interface *web*, o seu ramal deve ser criado como uma extensão no arquivo `extensions.conf`, para o Asterisk saber para onde ligar para o usuário (Quadro 15). Neste mesmo arquivo configuramos qualquer ramal.

```
[general]

[incoming]
exten=>10,1,Dial(SIP/10) ; Ramal 10 disca para usuario SIP 10
exten=>11,1,Dial(SIP/11) ; Ramal 11 disca para usuario SIP 11
exten=>12,1,Dial(SIP/12) ; Ramal 12 disca para usuario SIP 12
```

Quadro 15 – Ramal dos usuários SIP

3.3.2.4 Interligar com central telefônica ou receber chamadas da rede PSTN

Para interligar com uma central telefônica ou receber as chamadas da rede PSTN, devemos ter um hardware compatível com o projeto Zapata. Utilizou-se no trabalho uma

placa genérica do tipo FXO. O Linux necessita que carregue os módulos do núcleo do sistema para reconhecer essa placa (Quadro 16). Esses módulos devem ser carregados na inicialização do sistema.

```
root@ast1:~# modprobe zaptel
root@ast1:~# modprobe wcfxo
```

Quadro 16 – Carregar módulos no Linux

O arquivo `/etc/zaptel.conf` recebe a configuração de como reconhecer essa placa e utilizar os padrões de comunicação brasileiro (Quadro 17), enquanto o `/etc/asterisk/zapata.conf` define as configurações que o Asterisk deve interpretar para o hardware (Quadro 18). O arquivo `extensions.conf` deve ser alterado para ter o contexto PSTN, atender a ligação e mandá-la para o ramal SIP/10 (Quadro 19).

```
fxsks=1
loadzone=br
defaultzone=br
```

Quadro 17 – Configuração do `/etc/zaptel.conf`

```
[channels]
signalling=fxs_ks ; sinalizacao para o hardware fxo
callerid=asreceived ; uso do callerID, identificacao do numero
echocancel=yes ; habilitar cancelamento de eco
echocancelwhenbridged=yes ; cancelamento de eco para canais TDM
echotraining=400 ; ajuste do cancelamento de eco
context=PSTN ; contexto de entrada da ligacao
group=1 ; grupo qual faz parte para captura de chamadas
channel=>1 ; numero do canal/interface
```

Quadro 18 – Configuração do `/etc/asterisk/zapata.conf`

```
[PSTN]
exten=>s,1,Answer()
exten=>s,2,Dial(SIP/10)
exten=>s,3,Hangup()
```

Quadro 19 – Alteração no `/etc/asterisk/extensions.conf`

3.3.2.5 Comunicação com outro servidor Asterisk via IAX

Para habilitar entroncamento entre servidores Asterisk via IAX, deve ser configurado no arquivo `iax.conf` de cada servidor com o usuário para autenticar essa comunicação e o parâmetro de `trunk` (Quadro 20).

```
[general]
register=>iax1:senha@192.168.1.2

[iax2] ; nome do usuario
type=user ; tipo da conexao
auth=plaintext ; tipo de autenticacao
context=incoming ; contexto onde procurar as ligacoes
secret=senha ; senha
host=dynamic ; espera o telefone se registrar
callerid='iax2' ; identificador de chamada
trunk=yes ; modo de entroncamento
notransfer=yes ; nao transferir ligacao
```

Quadro 20 – Configuração do `/etc/asterisk/iax.conf`

3.3.2.6 Voice Mail

A configuração do *Voice Mail* é feita no `voicemail.conf` e no `extensions.conf`. No `voicemail.conf` se define o ramal, senha e nome do usuário. No arquivo `extensions.conf` deve se definir a aplicação *Voice Mail* para o determinado ramal (Quadro 21).

```
/etc/asterisk/voicemail.conf
200=>1234,200,francis@francis.eti.br

/etc/asterisk/extensions.conf
exten=>11,1,Dial(SIP/11,20) ; chama durante 20 segundos
exten=>11,2,Voicemail(u11) ; inicia Voice Mail caso ramal nao atende
exten=>11,3,Hangup()
exten=>11,102,Voicemail(b11) ; inicia Voice Mail para ramal ocupado
exten=>11,103,Hangup()

; Ouvir Correrio de voz
exten=>90,1,VoiceMailMain()
```

Quadro 21 – Definição do *Voice Mail*

3.3.2.7 Captura de chamadas

Para a captura de chamadas funcionar é necessário definir grupos para os ramais SIP. Cada ramal tem o parâmetro *callgroup* que indica de qual grupo ele faz parte e *pickupgroup* que indica qual grupo ele pode fazer a captura de chamadas. A captura é acionada através da combinação *8 definida no arquivo `features.conf` pelo parâmetro *pickupexten*.

3.3.2.8 Unidade de resposta audível

Para configurar a unidade de resposta audível devem-se criar os arquivos de som, ou seja, o que o Asterisk vai tocar para o menu e cada entrada em respectivo. Estes arquivos de som devem estar no formato GSM para melhor compatibilidade com o Asterisk (Quadro 22).

```
/etc/asterisk/extensions.conf

exten=>1000,1,Wait(2) ; guarda 2 segundos
exten=>1000,2,Record(menu:gsm) ; grava arquivo menu com o que e
falado ao microfone do telefone e # termina gravacao
exten=>1000,3,Wait(2)
exten=>1000,4,Hangup()

; URA
[ura]
exten=>s,1,Answer()
exten=>s,2,Background(menu) ; Toca menu e espera por algum digito
exten=>s,3,Wait(2)
exten=>s,4,Goto(s,2) ; Volta para extensao s e posicao 2

exten=>1,1,playback(menu1) ; Toca menu1 ao digitar 1
exten=>1,2,Hangup()
exten=>2,1,Playback(menu2) ; Toca menu2 ao digitar 2
exten=>2,2,Hangup()
```

Quadro 22 – Configurando URA

3.3.2.9 Conferência de ramais

A conferência de ramais deve ser configurado um ramal que entra na sala de conferência e o número daquela sala de conferência para ser utilizado na aplicação MeetMe (Quadro 23).

```
/etc/asterisk/meetme.conf
[rooms]
conf=> 1999

/etc/asterisk/extensions.conf
exten=>9999,1,MeetMe(1999|p) ; inicia conferencia de numero 1999 e o
p permite que # saia da conferencia
exten=>9999,2,Hangup
```

Quadro 23 – Configurando conferência

3.3.2.10 Bilhetagem em MySQL

Para habilitar a bilhetagem no banco de dados MySQL basta alterar os parâmetros em `/etc/asterisk/cdr_mysql.conf` colocando o usuário e senha do banco de dados e criando a tabela conforme seção 3.2.1.

3.3.2.11 Receber e enviar fax via correio eletrônico

Deve ser configurado em `extensions.conf` a recepção de fax conforme Quadro 24.

```
exten =>
fax,1,SetVar(FAXFILE=/var/spool/asterisk/fax/${CALLERIDNUM}.tif)
exten =>
fax,n,SetVar(FAXFILENOEXT=/var/spool/asterisk/fax/${CALLERIDNUM})
exten => fax,n,rxfax(${FAXFILE})
exten => fax,n,System('/usr/bin/fax2mail ${CALLERIDNUM}
"${CALLERIDNAME}" FaxNum RecipName email@address.com ${FAXFILENOEXT}
p')
```

Quadro 24 – Configuração fax

3.3.2.12 Módulo da *dialplan* SpyDial

O novo módulo da *dialplan* foi implementado seguindo o padrão da `app_skell.c` que é o modelo para este tipo de aplicação. Foram criadas duas estruturas que serão responsáveis por guardar os canais dos envolvidos na ligação e os números que devem ser discados para efetuar a ligação (Quadro 25).

```
struct spydial {
        struct ast_audiohook audiohook;
        struct ast_channel *caller;
        struct ast_channel *callee;
        struct ast_channel *monitor;
        struct dial_number *dial;
        char unique_id[20];
        ast_mutex_t lock;
        int threadon;
};
/* Guarda numeros do monitor e do numero chamado */
struct dial_number {
        char monitor_number[MAX_NTECH];
        char monitor_tech[MAX_NTECH];
        char callee_number[MAX_NTECH];
        char callee_tech[MAX_NTECH];
};
```

Quadro 25 – Estruturas da SpyDial

A estrutura `spydial` contém outros três tipos de dados que são utilizados pelo Asterisk: `ast_audiohook` guarda os parâmetros necessários para se conseguir copiar um frame de voz de outro canal já estabelecido; `ast_channel` é o canal criado pelo Asterisk que trata cada dispositivo e ou usuário conectado a ele; `ast_mutext_t` é responsável pela mútua exclusão para aplicações que fazem uso de *threads*.

O corpo principal da aplicação é a função `app_spydial` que recebe o canal que iniciou a comunicação como parâmetro e os argumentos passados para a aplicação. É iniciada as estruturas principais como ponteiros para poder manipulá-las por toda a aplicação e declarada a *thread* (Quadro 26).

```
static int app_spydial(struct ast_channel *chan, void *data)
{
        struct ast_module_user *u;
        struct dial_number *dial;
        struct ast_channel *peer;
        struct spydial *spymonitor;

        /* Responsavel pela thread */
        pthread_attr_t attr;
        pthread_t thread;
...
}
```

Quadro 26 – Início da aplicação SpyDial

A função `get_dialnumber` recebe os argumentos e separa eles corretamente retornando um ponteiro para uma estrutura do tipo `dial_number`. Tendo os números corretos, é alocada a estrutura `spydial` e colocado seus valores iniciais. Caso nenhum erro aconteça é iniciado a `dospy_thread` e a `do_dial` que disca para o destino (Quadro 27).

```
dial = get_dialnumber(args.peers);

/* Aloca spymonitor */
spymonitor = ast_calloc(1, sizeof(*spymonitor));
spymonitor->dial = dial;
ast_channel_lock(chan);
spymonitor->caller = chan;
pbx_builtin_setvar_helper(chan, "DIALSTATUS", "0");
ast_channel_unlock(chan);
```

Quadro 27 – Trecho de código app_spydial.c

Na função `do_dial` é importante destacar as chamadas `ast_request` que requisita um canal, `ast_call` que faz a ligação e `ast_bridge_call` que interliga os dois canais.

A *thread* iniciada espera pela chamada original se completar e então disca para o ramal monitor, caso o monitor atenda, é iniciado o processo de cópia dos frames de áudio para o ramal do monitor. O processo de cópia é feito pelas funções da API audiohook do Asterisk, esse processo de cópia é iniciado lendo o frame de áudio através da função

`ast_audiohook_read_frame` e depois escrito no canal ao qual se deseja ouvir aquele áudio copiado. Todo o processo de cópia de frames de áudio está em um laço infinito verificando se as chamadas estão estabelecidas corretamente e continuando a freqüente cópia e escrita dos frames de áudio (Quadro 28).

```
ast_audiohook_init(&spymonitor->audiohook, AST_AUDIOHOOK_TYPE_SPY,
"SpyDial");
res = ast_audiohook_attach(spymonitor->caller, &spymonitor-
>audiohook);

...

/* le o frame */
f = ast_audiohook_read_frame(&spymonitorpeer->audiohook, samples,
AST_AUDIOHOOK_DIRECTION_BOTH, AST_FORMAT_SLINEAR);

...

ast_write(spymonitor->monitor, f);
```

Quadro 28 – Trecho de código app_spydial.c

3.3.3 Operacionalidade da implementação

Para criar os usuários SIP o administrador do sistema deve acessar via HTTP de qualquer navegador o endereço do servidor, será exibido às opções de adicionar e listar os usuários (Figura 10).

Figura 10 – Menu interface *web*

Ao adicionar ou remover um usuário na interface *web* (Figura 11), também deve ser adicionado ou removido seu ramal no arquivo `extensions.conf` que também é onde configura-se a nova aplicação módulo Spydial na *dialplan* para monitorar as chamadas recebidas de um usuário.

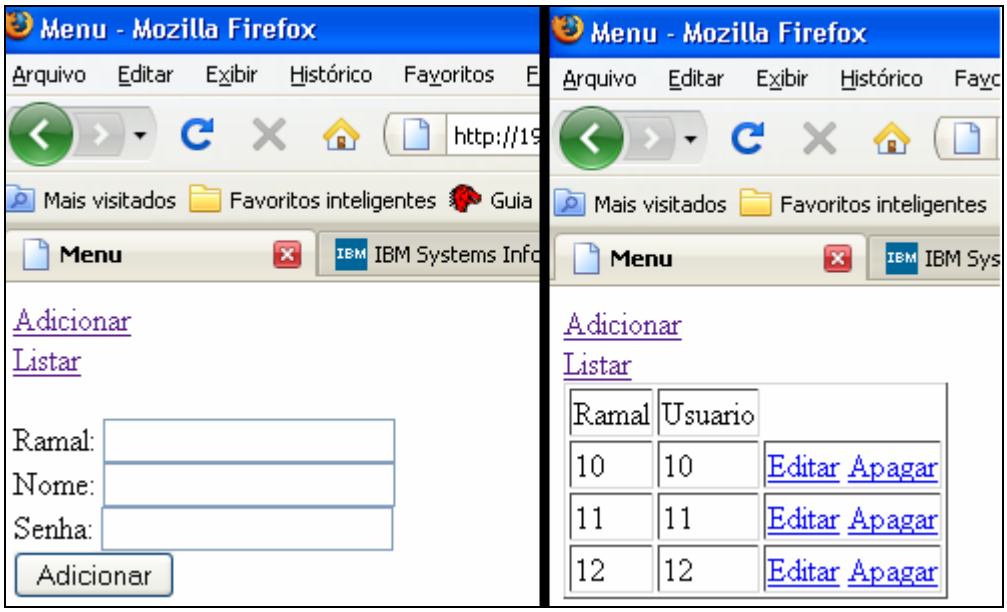


Figura 11 – Manutenção usuários na interface *web*

A utilização da nova aplicação SpyDial, deve ser configurada na *dialplan* a chamada `SpyDial` ao invés da `Dial`, seguindo os parâmetros na ordem: primeiro o ramal que vai monitorar separado por &; segundo o ramal que vai realmente receber a ligação (Quadro 29). Um exemplo de ligação sendo utilizada a SpyDial, no *software* da direita originando a ligação para o ramal 12 representado no *software* da esquerda, assim que o ramal 12 atender é disparada a ligação para o ramal 11 (Figura12).

```
[general]

[incoming]
...
exten=>12,1,SpyDial(SIP/11&SIP/12) ; Ramal 11 vai monitorar as
ligacoes recebidas do usuario SIP 12
...
```

Quadro 29 – *Dialplan* com SpyDial

Figura 12 – *Softwares* de telefone

3.4 RESULTADOS E DISCUSSÃO

O Asterisk se comporta de forma muito eficaz nos recursos configurados neste trabalho, principalmente quando se trata de uma ligação VoIP pura. A flexibilidade do software em trabalhar com diversas tecnologias e diversos CODECs de voz faz do Asterisk uma boa escolha como Central Telefônica VoIP.

Verificou-se que o SIP tem algumas dificuldades de trabalhar com NAT, tornando difícil algumas vezes utilizar os clientes SIP de outros lugares onde possam existir *firewalls* protegendo a rede. Em compensação o protocolo IAX resolve este tipo de problema com NAT e é o protocolo nativo do Asterisk, permitindo utilizar mais recursos que o SIP no Asterisk.

A aplicação de manutenção de usuários facilita troca de senhas e criação de novos ramais SIP e ainda o fato de a aplicação ser uma página *web*, a torna multi-plataforma evitando problemas de compatibilidade com sistema operacional. Uma questão importante não tratada na aplicação *web* é a questão de segurança: não foi imposto nenhuma restrição ao uso, sendo que qualquer usuário na rede que acessar o endereço no servidor poderá alterar os usuários.

Em trabalhos correlatos foi citado o VoiceOne que faz toda a configuração via página *web* de todos os arquivos editados manualmente no trabalho desenvolvido, inclusive a adição de usuários SIP. Foi citado também a `app_dial` e a `app_chanspy`, sendo a `app_dial` responsável direta por efetuar chamadas no Asterisk, referenciada na *dialplan* como Dial(technologia/número). A `app_dial` e seu código fonte foram estudados por completo para saber exatamente como Asterisk se comporta em uma chamada e com base nesses estudos foi feito a aplicação `app_spydial` para discar para outros ramais. A `app_chanspy`, faz espionagem de outros canais através do audiohook, sendo esta estudada para saber como implementar as chamadas do audiohook corretamente.

No trabalho a aplicação `app_spydial` foi limitada a apenas conseguir discar e estabelecer comunicação de forma simples com os ramais envolvidos e monitorar o canal desejado sem qualquer outro tipo de recurso avançado.

4 CONCLUSÕES

A proposta de implantar e customizar um sistema de voz sobre IP com softwares livres, configurando alguns recursos de uma central telefônica digital, com a criação de uma interface *web* para manutenção de usuários SIP e uma aplicação que monitore as chamadas foi completamente atendida.

O Asterisk funcionou perfeitamente com os recursos configurados no trabalho, sendo fácil de configurar e testar. Os serviços como o *Voice Mail*, a interligação com outro Asterisk, recepção de fax, interligar com central telefônica, captura de chamadas, salas de conferência, a bilhetagem em banco de dados e unidade de resposta audível foram configurados com êxito como descrito no capítulo 3 do trabalho. Apenas a configuração da lógica de *dialplans* que exige um planejamento melhor quando temos grande quantidade de ramais e diversos setores com configurações diferentes, fazendo muito uso de macros e chamadas `goto`.

Programar uma aplicação de módulo da *dialplan* no Asterisk não é uma tarefa trivial devido ao código fonte não ser bem documentado e não ter uma documentação de seu funcionamento, exigindo muita leitura de código fonte em C para entender sua estrutura interna. Outro grande problema enfrentado no Asterisk é a concorrência de processos, exigindo cuidado programador ao alocar e liberar algum recurso. Este trabalho mostrou um pouco da arquitetura do Asterisk e os passos para se criar um módulo da *dialplan*.

Desta forma conclui-se que arquitetura modular do Asterisk permite criar-se diversas aplicações, sem necessitar de conhecimento em algumas áreas, como a aplicação SpyDial criada nesse trabalho que se utiliza do processo de discagem comunicando com qualquer outra tecnologia suportada pelo Asterisk, comunicando-se via SIP sem a necessidade de programar o protocolo SIP.

Por fim, a tecnologia VoIP está vindo para ficar e substituir gradativamente a telefonia convencional, apenas ainda tendo empecilhos em alguns equipamentos de VoIP terem os preços mais altos que os correspondentes da telefonia tradicional.

4.1 EXTENSÕES

Este trabalho não abordou nenhuma questão de segurança no Asterisk, no VoIP e na

interface *web* também, melhorar esse questão é muito importante.

Na interface web sugere-se adicionar um novo recurso para que esta crie a extensão automaticamente, para não editar manualmente o arquivo `extensions.conf`.

REFERÊNCIAS BIBLIOGRÁFICAS

ALENCAR, Marcelo S. **Telefonia digital.** 4. ed. São Paulo: Érica, 2002.

ALVES, João B. M. **VoIP:** a primeira revolução do século 21. Florianópolis, 2005. Disponível em: <http://www.inf.ufsc.br/~adriana/fase_01/tgs/trab_03/Trabalho3.htm>. Acesso em: 06 ago. 2008.

DEMPSTER, Barrie; GOMILLION, David. **Construindo sistemas de telefonia com o Asterisk.** Birmingham: Packt Publishing Ltd, 2005.

DIGIUM. **AsteriskNOW**. Massachusetts, 2008a. Disponível em: <http://dl1.digium.com/quickstart_asterisknow.pdf >. Acesso em: 07 abr. 2008.

DIGIUM. **Asterisk downloads**. Massachusetts, 2008b. Disponível em: <http://www.asterisk.org/downloads>. Acesso em: 15 ago. 2008.

DIGIUM. **Coding guidelines**. Massachusetts, 2008c. Disponível em: <http://svn.digium.com/view/asterisk/trunk/doc/CODING-GUIDELINES?view=markup>. Acesso em: 30 ago. 2008.

DIGIUM. **Developers documentation**. Massachusetts, 2008d. Disponível em: <http://www.asterisk.org/doxygen/1.4/>. Acesso em: 02 set. 2008.

DÓRIA, Hugo; JÚNIOR, Israel; OLIVEIRA, Tadeu. **Inter-Asterisk exchange**. Aracaju, 2006. Disponível em: <http://gpwm.devin.com.br/index.php/Inter-Asterisk_Exchange_(IAX)>. Acesso: 18 set. 2008.

FERREIRA, Ainda A.; BRANDÃO, Glória A. V. C. Estudo das tecnologias de transmissão de voz sobre IP (VoIP) e desenvolvimento de uma aplicação VoIP. In: CONGRESSO DE PESQUISA E INOVAÇÃO DA REDE NORTE NORDESTE DE EDUCAÇÃO TECNOLÓGICA, 2., 2007, João Pessoa. **Anais eletrônicos...** João Pessoa: Centro Federal de Educação Tecnológica de Pernambuco, 2007. Disponível em: <http://www.redenet.edu.br/publicacoes/arquivos/20080227_094117_TELE-018.pdf>. Acesso em: 02 ago. 2008.

GONÇALVES, Flávio E. V. **Asterisk PBX guia de configuração.** Florianópolis: VOffice, 2005.

GONZALEZ, Felipe N. **Estudo e implantação de solução de voz sobre IP em softwares livres.** Joinville: TUPY, 2007.

HERSENT, Oliver; GURLE, David; PETIT, Jean P. **Telefonia IP.** São Paulo: Makron Books, 2002.

MADEIRA, Frederico T. T. **Segurança em redes de voz sobre IP**. 2007. 90 f. Trabalho de Conclusão de Curso (Pós-Graduação em Segurança em Redes de Computadores) – Associação de Ensino Superior de Olinda, Olinda.

MEGGELEN, Jim V.; SMITH, Jared; MADSEN, Leif. **Asterisk o futuro da telefonia.** Rio de Janeiro: Alta Books, 2005.

OZVOIP. **VoIP codecs clients compare codecs**. Surfers Paradise, 2008. Disponível em: <http://www.ozvoip.com/codecs.php>. Acesso em: 20 set. 2008.

RFC 2543. **SIP:** Session Initiation Protocol. Columbia, 1999. Disponível em: <http://www.ietf.org/rfc/rfc2543.txt>. Acesso: 18 set. 2008.

SOFT-SWITCH. **Building and installing the spandsp library**. Lantau Island, 2008. Disponível em: <http://www.soft-switch.org/installing-spandsp.html>. Acesso em: 02 out. 2008.

SYNGRESS. **Asterisk hacking.** Burlington: Syngress Publishing, 2007.

VOICEONE. **VoiceOne.** Verona, 2008. Disponível em: <http://www.voiceone.it>. Acesso em: 07 set. 2008.