Desenvolvimento de aplicação para transmissão de áudio sobre IP utilizando o protocolo SIP

Jadielson ALVES (1); Aida FERREIRA (2)
(1) Centro Federal de Educação Tecnológica de Pernambuco, Avenida Professor Luiz Freire, 500, Cidade Universitária, Recife-PE, CEP: 50740-540, 87943588, e-mail: dielson10@hotmail.com
(2) Centro Federal de Educação Tecnológica de Pernambuco, aidaaf@gmail.com

RESUMO

A sigla VOIP vem do inglês 'V'oice 'O'ver 'I'nternet 'P'rotocol. Como o próprio nome já diz "voz sobre IP", é uma tecnologia que consiste na digitalização e codificação de voz. Transformando a voz humana em pacotes IP e transmitindo-a através da grande rede, a qual requer a utilização do protocolo IP. Para que seja possível essa transmissão os usuários fazem uso de servidores que auxiliam na localização de outros usuários, como descreve o protocolo SIP. O trabalho apresenta o desenvolvimento de um protótipo de um SIP fone, onde é utilizada a biblioteca JAIN-SIP para o desenvolvimento do que é descrito no protocolo SIP, e a codificação está escrita na linguagem de programação Java. No protótipo encontra-se implementado o "user client", que consiste nas funcionalidades disponíveis para o cliente, tais como a realização de uma chamada para outro usuário e a visualização dos usuários disponíveis para serem chamados. Também se encontra implementado no protótipo o servidor de "Register", esse servidor é responsável pelo registro do usuário que entra ou sai do sistema, e também por enviar atualizações para todos os usuários. A metodologia utilizada no decorrer da pesquisa foi inicialmente fazer uma revisão bibliográfica sobre transmissão de áudio e vídeo sobre IP e sobre protocolo SIP. Com o embasamento teórico iniciou-se o desenvolvimento do software e em seguida um estudo de caso para avaliar a aplicação desenvolvida. A pesquisa encontra-se atualmente na fase de desenvolvimento do protótipo α do SIP fone CEFET-PE. Esse trabalho será muito importante para a educação profissional e tecnológica, pois ajudará tanto alunos como profissionais que tenham interesse na área de redes de computadores, e especificamente transmissão de voz sobre IP.

Palavras-chave: protocolo IP, digitalização, software, VoIP, SIP, JAIN-SIP

1. INTRODUÇÃO

Atualmente, é muito comum ouvir se falar sobre a sigla VoIP (voice over IP). Essa tecnologia que vem sendo aprimorada ao longo dos anos, e que hoje é capaz de transmitir dados, voz e vídeo em tempo real, fazendo uso do Protocolo IP.

A sigla VoIP significa o transporte da voz sob uma infra-estrutura IP. Esta infra-estrutura pode ser LAN ou WAN (Frame Relay/ATM/PPP, etc.). Geralmente, quando mencionamos VoIP, estamos falando da integração do PABX com um gateway (roteador ou switch), que faz a conversão da voz tradicional para Voz sobre IP, onde é empacotada transportada através da rede TCP/IP.

Para que ocorra a transmissão entre dois pontos é necessário um protocolo que faça toda sinalização antes da troca de voz. Nos primórdios da troca de voz sobre IP, utilizava-se para este fim a arquitetura H.323, que supria todas as necessidades de sinalização, mas ele era complexo e de difícil implementação.

O IETF especificou um protocolo de sinalização para negociação de abertura e fechamento de sessão entre muitos usuários. Esse protocolo é denominado Session Initiation Protocol (SIP). O SIP em sua versão mais nova está publicado na RFC 3261. Suas principais funcionalidades são de localizar o usuário, estabelecer chamadas e administrar as participações nessas chamadas.

2. SESSION INITIATION PROTOCOL

O protocolo SIP(Session Initiation Protocol) é um padrão da Internet Engineering Task Force(IETF). Ele é um modelo de "requisição e resposta", ou seja, o modelo cliente-servidor, parecido com o HTTP. Esse protocolo assim como o HTTP, também provê de sinalizações entre o cliente e o servidor. O protocolo SIP é usado para iniciar, modificar ou terminar sessões ou realizar chamadas multimídia de voz, vídeo e mensagens entre usuários, fazendo isso através de sinais trocados entre os participantes.

O desenvolvimento do SIP foi iniciado na década de 90, tendo sua primeira versão no ano de 1996, chamado inicialmente de *Session Invitation Protocol*. Essa versão apenas estabelecia a sessão, outras funcionalidades, como controles para conferencias, foram introduzidas no SIP versão 2.0 que foi lançada em 1997. Em 2001, com a grande popularidade do SIP, a IETF dividiu o grupo de desenvolvimento em três vertentes, o SIP, SIPPING e mais tarde o SIMPLE. O SIP era o grupo responsável pelas especificações fundamentais do protocolo e suas extensões. O SIPPING era responsável pelas aplicações que utilizavam o protocolo. E o SIMPLE se encarregava de padronizar a presença e Instant Messenger's utilizando o SIP.

Em fevereiro de 1999 o SIP foi proposto como um padrão, publicado como RFC 2543 e sua última versão, o SIPv2 foi proposto como padrão em 2002 e publicado como RFC 3261, substituindo a versão anterior.

As principais funcionalidades do protocolo SIP são:

- A localização do usuário: os usuários não têm endereços IP fixo, porque esses endereços podem ser gerados pelo DHCP, e também porque os usuários podem ter vários equipamentos com IP diferente. Para se estabelecer a chamada é necessário descobrir o endereço atual do usuário a ser chamado, assim podendo dar início a sessão.
- O estabelecimento de chamadas: existem mecanismos para que haja o estabelecimento de uma chamada entre usuários utilizando uma rede IP. O SIP permite que um usuário que chama avise ao usuário chamado, que ele está querendo estabelecer uma chamada. Permite também que os interlocutores negociem qual o tipo de codificação que será utilizada. Também prover a finalização da chamada.
- Administração na participação de chamadas: alguns mecanismos na administração das chamadas são propostos, tais como, adicionar novas mídias, negociar um novo tipo de codificação, convidar outros usuários, transferência de chamadas, tudo isso pode ser feito durante a chamada.

3. ARQUITETURA SIP

A arquitetura SIP apresenta duas sub-divisões de componentes que são: Clientes SIP e os Servidores SIP, que estão sub-divididos em Servidor de Localização SIP, Servidor Proxy SIP, Servidor de Redirecionamento SIP e Servidor de Registro SIP. Estes elementos da rede SIP são mostrados na figura 1:

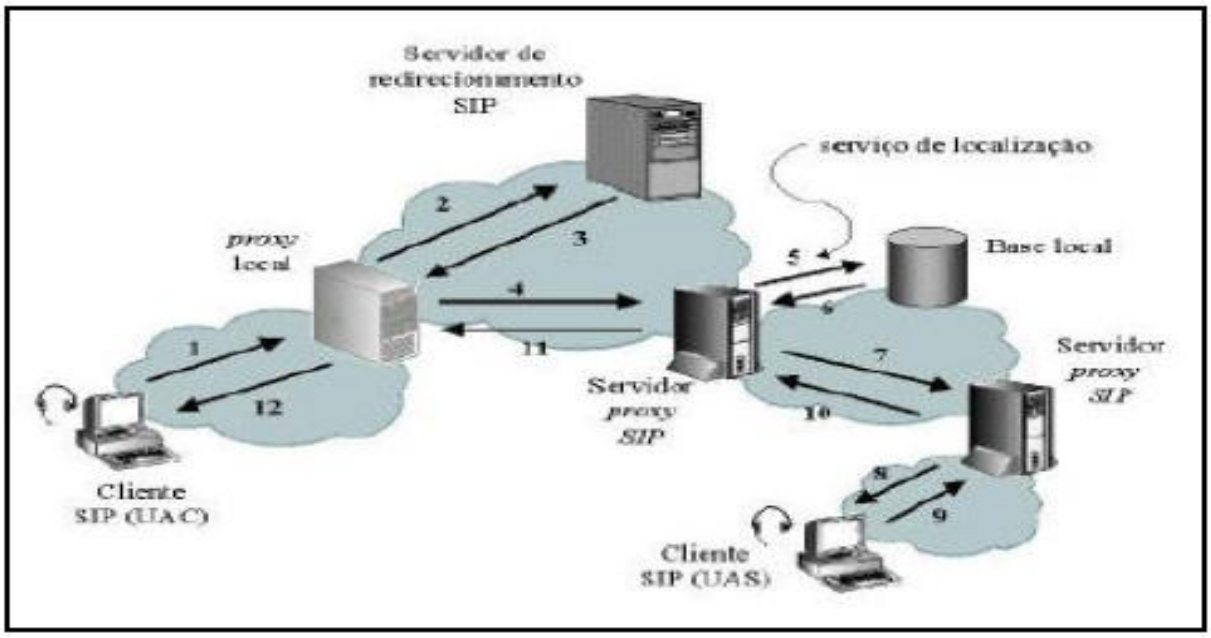

Figura 1 : componentes da arquitetura SIP

4. MENSSAGENS SIP

O SIP funciona numa arquitetura cliente/servidor. Assim como o HTTP ele precisa de alguns métodos para a sinalizações de requisição e respostas. Alguns desses métodos são: o INVITE, para fazer o convite ao outro usuário; o ACK, esse método é uma resposta do INVITE; o CANCEL, esse método é responsável pelo cancelamento de todas a requisições pendentes; o OPTIONS, ele faz uma pergunta sobre a capacidade e disponibilidade tanto ao servidor quanto ao cliente; o REGISTER, um cliente usa este método para registrar o "alias" (apelido) do seu endereço em algum servidor SIP; e o BYE, esse método é responsável pelo termino da sessão.

5. BIBLIOTECA JAIN-SIP

O conceito de biblioteca na ciência da computação é o de um conjunto de subrotinas, que servem para o desenvolvimento de software.

No inicio da implementação tentou-se utilizar a biblioteca Java Sound, mas não foi possível dar seqüência em sua implementação, já que ela não dá suporte para o desenvolvimento de aplicações SIP. Então foi escolhida a biblioteca JAIN-SIP v1.2, que foi desenvolvida pela Sun e pelo grupo NIST. Esta biblioteca possui classes para controle de sessão e mídia do SIP e também descrição de sessão SDP, tanto para desktop como para servidor. Esta biblioteca é uma interface Java-standard que define uma API que permite a rápida criação e implantação de serviços em uma telefonia dinâmica, através da plataforma Java.

O Jain-SIP pode ser utilizado para implementação dos user agents, proxy, registra.

A linguagem escolhida para implementação do projeto foi à linguagem de programação Java, porque além de toda a biblioteca Jain-SIP ser implementada em Java, foi escolhida também pela segurança, ferramentas para a programação gratuitas e pela sua portabilidade, pois o aplicativo será multi-plataforma.

6. ARQUITETURA DO JAIN-SIP

Para que se possa ocorrer à transmissão de pacotes entre dois pontos, chamador e chamado, é necessário a abertura de uma sessão entre eles. O conceito de sessão é simplesmente uma troca de dados em tempo real entre dois pontos.

Para o estabelecimento da sessão foi usado o JAIN SIP v1.2 que implementa as especificações RFC 3261 do SIP de iniciação de sessão. Ela é utilizada em conjunto com o protocolo SDP, especificado no RFC 2327, para descrição de sessão, e implementado também nas bibliotecas do JAIN NIST.

O SIP é um protocolo que é baseado em mensagens. Essa mensagens são definidas como request(pedido) que é feito pelo cliente e a response(resposta) que é a resposta dada pelo servidor. Para que se inicie, altere e termine a sessão, será necessária a troca de request e response entre o chamador e o chamado.

Para que haja qualquer tipo de troca de mensagens será necessária uma estrutura básica, que é mostrada na figura 2:

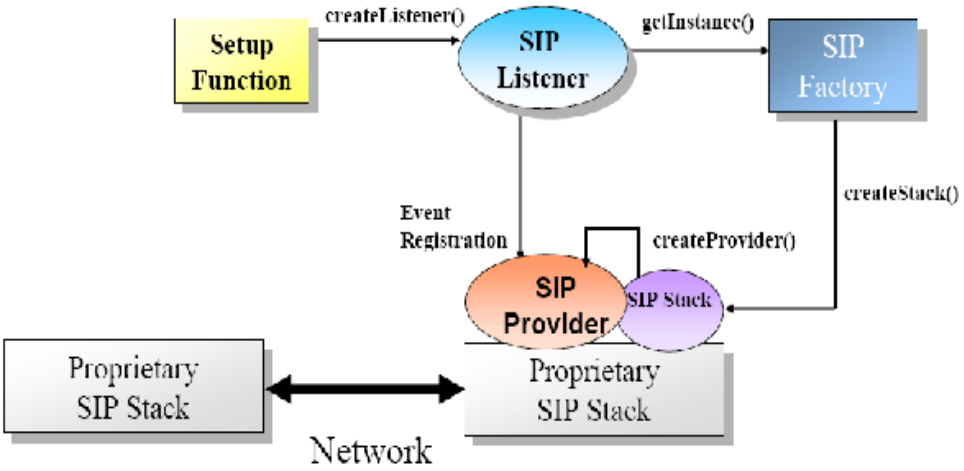

Figura 2: Arquitetura JAIN-SIP

Para que se possa criar toda arquitetura temos um objeto essencial, o **SipFactory**. O **SipFactory** é responsável pela criação do **SipStack**, que é a classe responsável por toda estrutura de recepção e envio de mensagens de pedido e resposta. **HeaderFactory** para criação de novos objetos cabeçalhos. **AddressFactory** que define métodos para criação de SipURI's e TelURL's. **MessageFactory** que define métodos para criação de novas requests e responses.

No JAIN SIP ainda existem as classe **ListeningPoint**, **SIPProvider**, **SIPListener,** que são essenciais para a transmissão e recepção de mensagens.

O **ListeningPoint** nada mais é do que um ponto único de escuta em uma rede IP, e composto também de uma porta e um protocolo de transporte. O **ListeningPoint** é a porta de comunicação utilizada pelo **SipProvider** para o envio e recebimento de mensagens.

Um objeto **SipStack** pode ter múltiplos **ListeningPoints**, enquanto uma entidade **SIPProvider** pode ter somente um **ListeningPoint**.

A interface **SIPProvider** é responsável pelo tratamento das menssagens e o componente das transações do **SipStack**. Ele é implementado por qualquer objeto **SipStack**.

A Interface **SIPListener** define o canal de comunicação da aplicação para o **SipStack**. Ela define métodos que serão chamados quando a aplicação receber e processar eventos, que são emitidos por um objeto que implementa a interface **SIPProvider**.

O **SIPListener** possui três eventos: **RequestEvents**, que são eventos de pedidos emitidos pelo **SIPProvider**, como os pedidos de INVITE. **ResponseEvets**, são eventos de resposta emitidos pelo **SIPProvider**, como as respostas 2xx. **TimeOuts**, são eventos de estouro de tempo emitidos pelo **SIPProvider**, esse estouros notificam que uma retransmissão é necessária ou o tempo da transmissão acabou.

A figura 3 abaixo mostra a arquitetura das mensagens na Jain-SIP:

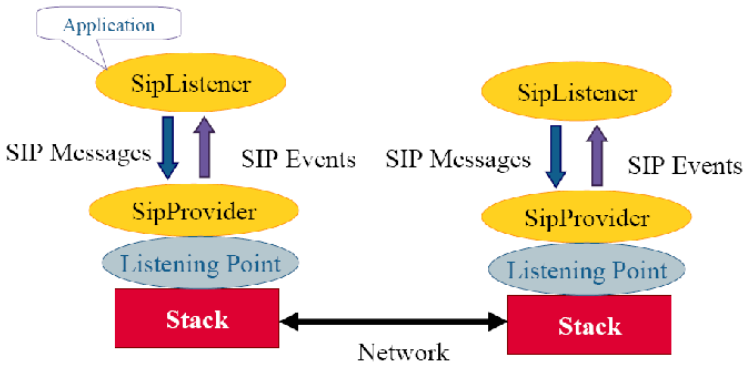

Figura 3: Arquitetura das mensagens JAIN-SIP

7. API JMF

JMF (Java Media Framework) é uma API que permite a manipulação de áudio, vídeo e outras mídias em aplicações Java como applications e applets. Com ela é possível capturar stream de áudio e vídeo e codificá-lo em diversos formatos como também a transmissão das mídias pelo padrão RTP (Real time Transport Protocol) para o desenvolvimento de aplicações que utilizem vídeo sob demanda.

Para criar e/ou executar um player em Java usando API JMF é necessário instalar o aplicativo *jmf-2_1_1e* , no caso do Windows é o *jmf-2_1_1e-windows-i586.exe* , o qual possui o papel de interfacear as funcionalidades do player com o Sistema Operacional.

Após a instalação, o JMF apresenta a seguinte estrutura de pastas:

- **bin** – contém scripts aplicativos para testes;
- **doc** – contém um arquivo de documentação;
- **lib** – contém as classes java e arquivos nativos para o funcionarmento de um player.

A API JMF define dois tipos de controle da mídia, Player e Processor. O Player é usado para a reprodução da mídia do stream e o Processor fornece a função básica para o controle e processo do fluxo de dados de mídia.

Um Player ou Processor, recebe stream de dados na entrada que pode ser feito através do DataSource e a forma de exibição vai depender do tipo de mídia a ser processada.

A figura 4 mostra um simples esquema de processamento de uma stream de dados:

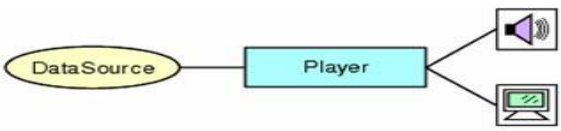

Figura 4: Processamento de uma stream de dados

8. IMPLEMENTANDO O CLIENTE COM JMF

Para a implementação com a biblioteca JMF deve-se primeiramente instalar o aplicativo *jmf*, como falado anteriormente. Após a instalação do aplicativo, deve-se ir na pasta **lib** para importar as biblioteca nativas disponíveis pela JMF através do ambiente de desenvolvimento, nesse projeto se faz uso do ambiente de desenvolvimento chamado **Eclipse,** desenvolvido pela **Sun**. Após importar as bibliotecas necessárias, é hora de iniciar o desenvolvimento da implementação.

Quando a classe para o gerenciamento da mídia é criada, deve-se implementar as bibliotecas **ControllerListener**, **ReceiveStreamListener**, **ActionListener**. Essas bibliotecas fornecem métodos para o tratamento, respectivamente, de eventos gerados pelo controle do player, pela recepção de eventos gerados pela sessão com o envio de uma stream, e pelos eventos de ação, que servem para analisar o desempenho do player.

Depois de toda a configuração e de importar as bibliotecas necessárias, deve-se iniciar a captura da voz. Mas para a captura da voz será necessário um dispositivo de captura de voz, que será encontrado através do método **CaptureDeviceManager.getDeviceList(null)** que retornará uma lista com todos os dispositivos que suportam um determinado formato. Após ter capturado a lista de dispositivos, deve-se criar um objeto onde receberá o local onde será a fonte de dados de mídia através do método **Manager.createDataSource(deviceInfo.getLocator())**. Agora é preciso criar um objeto **Processor**, o objeto que fornece a função básica para o controle e processo do fluxo de dados de mídia, através do método **Manager.createProcessor(origDataSource)**, passando a fonte de dados como parâmetro. Depois deve-se configurar o **Processor**, após toda configuração necessária o **Processor** é vinculado à fonte de captura, de modo que a mídia capturada seja processada e transformada em um fluxo de áudio, através do método **processor.getDataOutput()**. O trecho de código na figura 5 exemplifica esse processo inicial da captura da voz:

```
AudioFormat format = new AudioFormat(AudioFormat.LINEAR,
8000, 8, 1);
Vector devices = CaptureDeviceManager.getDeviceList(format);
origDataSource = Manager.createDataSource(di.getLocator());
processor = Manager.createProcessor(origDataSource);
// faz toda configuração necessária
origDataSource = processor.getDataOutput();
```

Figura 5: implementação inicial da captura de voz

No término da implementação inicial é necessário a criação de uma sessão para o envio de **stream**, gerenciamento dos participantes, entre outras funções. Essa sessão é criada pelo método **mymgr = new RTPSessionMgr()**, após criá-la adiciona-se um **listener** para escutar os eventos surgidas na sessão, pelo método **addReceiveStreamListener(this)**. Agora deve-se capturar o endereço local e endereço remoto da transmissão, onde eles serão transformados em endereço de sessão local e endereço de sessão remoto através do método **getByName()**. Na criação dos endereços de sessão através do construtor da classe **SessionAddress**, no qual é passado quatro parâmetros que são o IP da máquina local ou da máquina de destino, a porta para recepção dos pacotes **RTP**, o IP passado no primeiro parâmetro e a porta para controle dos pacotes recebidos, o número dessa porta de controle deve ser adicionado mais uma unidade. Agora deve-se inicializar a sessão através do método **initialize(localaddr)**, que recebe como parâmetro o endereço da sessão local. Após a inicialização da sessão é necessário abri-la utilizando o método **addTarget(outraAddr)**, que recebe como parâmetro o endereço de sessão do host remoto. O método **initialize()** e o **addTarget()** após serem chamados criam a sessão, onde através dela serão enviados os pacotes com relatórios RTCP e os pacotes RTP contendo a mídia. Para o envio das streams contendo o áudio, será necessário a invocação do método que cria essas streams, chamado **createSendStream(origDataSource, 0)**, esse método recebe como parâmetro o data source, ou seja, a fonte de dados de onde

está captando a mídia, juntamente com o parâmetro stream índex, que pode ser 0 ou 1, caso seja utilizado o 1, pode-se ter duas mídias em um único pacote RTP, e no caso seja utilizado 0, um único tipo de mídia é encapsulado no pacote RTP. O método **createSendStream** retorna um objeto do tipo **SendStream**, que será utilizado um método **star()** contido nesse objeto, para iniciar a transmissão de dados pelos pacotes RTP. É necessário também startar o **processor**, através do seu método **start()**, para que a sessão seja controlada por ele. Na figura 6 é mostrado o trecho de código para criar e iniciar o envio de voz sobre a sessão:

```
rtpManager = new RTPSessionMgr();
localAddress = new SessionAddress();
destaddr = InetAddress.getByName(destaddrstr);
outroAddr = InetAddress.getByName(outraAdd);
localaddr = new SessionAddress(destaddr, portstr);
sessaddr = new SessionAddress(destaddr, port+2, destaddr,
port + 3);
outraAddrem = new SessionAddress(outroAddr, portstre);
sessaddrem = new SessionAddress(outroAddr, outraPort, outroAddr,
outraPort + 1);
rtpManager.initialize( localaddr);
rtpManager.addReceiveStreamListener(this);
rtpManager.addTarget(outraAddrem);
sendStream = rtpManager.createSendStream( origDataSource, 0);
sendStream.start();
processor.start();
```

Figura 6: implementação para criação e inicialização de uma sessão

9. IMPLEMENTANDO O SERVIDOR DE REGISTRO

Na utilização de um SIP fone é de fundamental importância uma lista com os endereços dos usuários disponíveis, porque quando for necessário realizar uma ligação para outro SIP fone deve-se saber seu endereço atual do dispositivo em que a pessoa desejada se encontra. Ficaria inviável se ter uma agenda para anotar esses endereços, mesmo porque alguns dispositivos têm seu endereço dinâmico, o qual fica mudando de tempos em tempos. Para solucionar esse problema, seria necessário um servidor que recebesse todos os endereços correntes de todos os dispositivos que fizessem uso dele. Esse servidor também deve atualizar os endereços quando alguém faz login no sistema e quando alguém se desconecta, mantendo a lista de endereços sempre atualizada. O servidor responsável por essas funções é chamado de "Servidor de Registro".

O protocolo SIP além de dar suporte a implementação dos agentes usuários servidores e agentes usuários clientes, ele também fornece métodos para a implementação de servidores, entre eles o servidor de registro, que oferece as funcionalidades que foram descritas no parágrafo anterior. Ele obtém os endereços dos dispositivos que se logam a ele e distribui esses endereços para todos os endereços que estão on-line, o mesmo acontece quando algum dispositivo se desconecta do sistema.

A obtenção de um registro para um agente de usuário será feita através de uma implementação de uma **request** com a mensagem REGISTER, a qual o servidor de registro vai recebê-la e tratá-la, mandando em seguida uma **response** com a confirmação do seu login no sistema. Logo em seguida o servidor irá enviar uma lista de endereços que estão logados no sistema, e essa lista será atualizada a cada login feito no servidor.

No lado do servidor, quando a **request** é recebida, ela será encaminha para um método tratá-la. Nesse método será extraído o cabeçalho da requisição, onde contém o método da requisição, nesse caso o REGISTER, o nome e endereço do cliente, a porta, entre outros identificadores. Após colher todas as informações necessárias, o servidor vai validar o login e a senha do agente usuário, através de um banco de

dados que contém o login e senha de todos os usuários cadastrados a ele. Quando a consulta ao banco é feita o servidor fica esperando a resposta, que se for positiva ele enviará uma **response** para o cliente com a confirmação do seu login no sistema, e caso seja negativa ele enviará uma **response** contendo uma mensagem com um erro.

Na figura 7 é mostrado uma parte do código para implementar o tratamento do REGISTER, o armazenamento do endereço em uma lista dinâmica, o pedido para um método de atualização dos endereços e a **response** enviada para o usuário.

```
Request req = Event.getRequest();
FromHeader from =
(FromHeader)req.getHeader("From");
lista.add(end);

Response response = null;
Dialog dialogo = ServerId.getDialog();
this.currentDialog = dialogo;
// restante do cabeçalho
ServerId.sendResponse(response);
this.atualizarLista();
```

Figura 7: Tratamento do REGISTER

Quando uma **request** com o pedido de registro ”REGISTER” chega ao servidor, além da validação do usuário, será necessário guardar esses endereços de modo dinâmico, ou seja, quando alguém se logar no sistema seu endereço seja guardado em uma lista e quando alguém sair do sistema seu endereço seja excluído da mesma lista. A lista de endereços deve ser enviada para todos os usuários, no momento em que alguém entra ou que alguém sai do sistema, desse modo a lista de usuários on-line sempre vai estar atualizada para todos os clientes logados.

No lado do Agente Usuário, toda vez que o sistema é iniciado haverá uma tela de login e senha para que possa acessar o sistema. Após o preenchimento da tela de login, as informações serão enviadas para o servidor através de uma **request** com o pedido de registro “REGISTER”, onde essas informações contidas na **request** serão extraídas e validadas pelo servidor. Enquanto isso o cliente fica aguardando a resposta que será enviada pelo servidor, se caso a resposta for positiva, abrirá uma nova tela que será a tela principal do SIP fone, onde contém os nomes dos usuários que estão on-line.

Para conseguir preencher a lista com os nomes que estão on-line, os usuários aguardarão uma mensagem que será enviada pelo servidor, vale salientar que ela só será enviada caso a senha e o login do usuário tenha sido validado pelo servidor. A mensagem que contém os endereços dos usuários logados será enviada através de uma **request** com a requisição “MESSAGE”, a qual conterá uma string com todos os usuários. Uma vez que chegou a **request** com os endereços, a mensagem será extraída e a string que ela contém será tratada por um método a parte, para a extração dos endereços que a string detém.

Na figura 8 mostra um trecho de código para a implementação de um pedido REGISTER.

```
SipURI from =
addressFactory.createSipURI(getUsername(), getHost()
+ ":" + getPort());
Address fromNameAddress =
addressFactory.createAddress(from);
fromNameAddress.setDisplayName(getUsername());
//restante do cabeçalho
Request request =
messageFactory.createRequest(requestURI,
Request.REGISTER, callIdHeader, cSeqHeader,
fromHeader,
toHeader, getLocalViaHeaders(),
getMaxForwardsHeader());
regTrans.sendRequest();
```

Figura 8: Implementação de um pedido REGISTER

10. INTERFACE GRÁFICA DO CLIENTE E SERVIDOR

A interface gráfica do SIP fone CEFET-PE e do SIP Servidor foi desenvolvida no ambiente de desenvolvimento Eclipse. A interface do SIP fone apresenta inicialmente uma tela de login, onde o usuário entrará com sua senha e login, que deverá estar cadastrado anteriormente. Após se logar o usuário entrará em uma tela onde mostrará os usuários que estão ativos no sistema, onde o usuário deverá clicar na pessoa desejada e logo em seguida seleciona chamar. Para finalizar a conversa, o cliente pode selecionar o botão encerrar a qualquer momento.

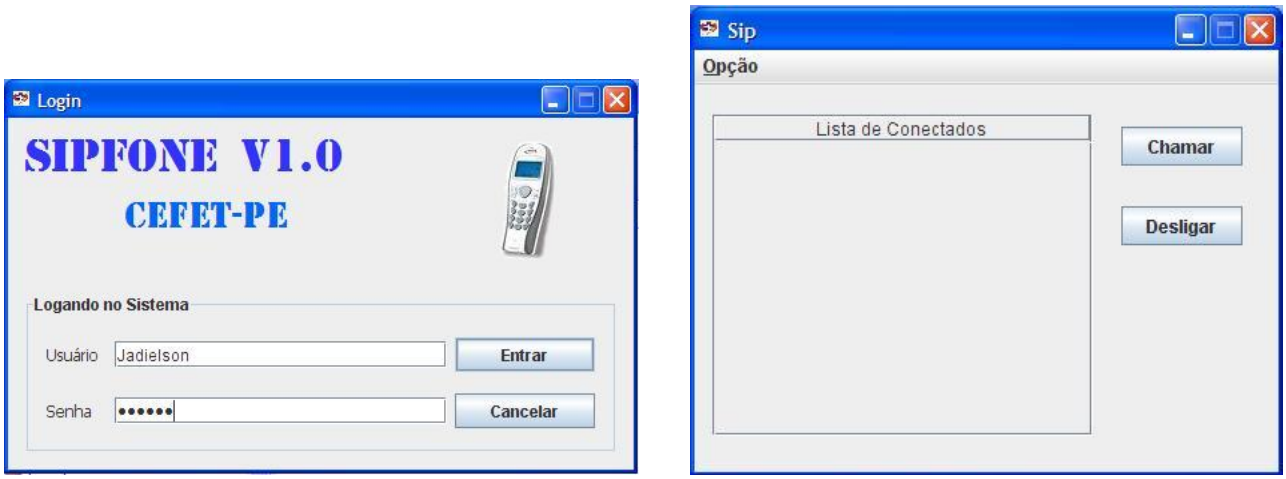

Figura 9: Interface gráfica do Sip Fone Cefet-PE

A interface do servidor contém uma tela inicial com a opção de cadastrar usuário, onde o administrador deverá cadastrar o usuário com um login, que deverá ser único no sistema, e uma senha. Na tela inicial também contém uma opção para paralisação do servidor, essa opção não exclui os usuários ativos no sistema. Essa opção pode ser utilizada para eventuais manutenções no servidor. Na tela inicial também contém a opção para ativar o servidor, que por default vêm paralisado. Essa opção ativa as funcionalidades do servidor de registro.

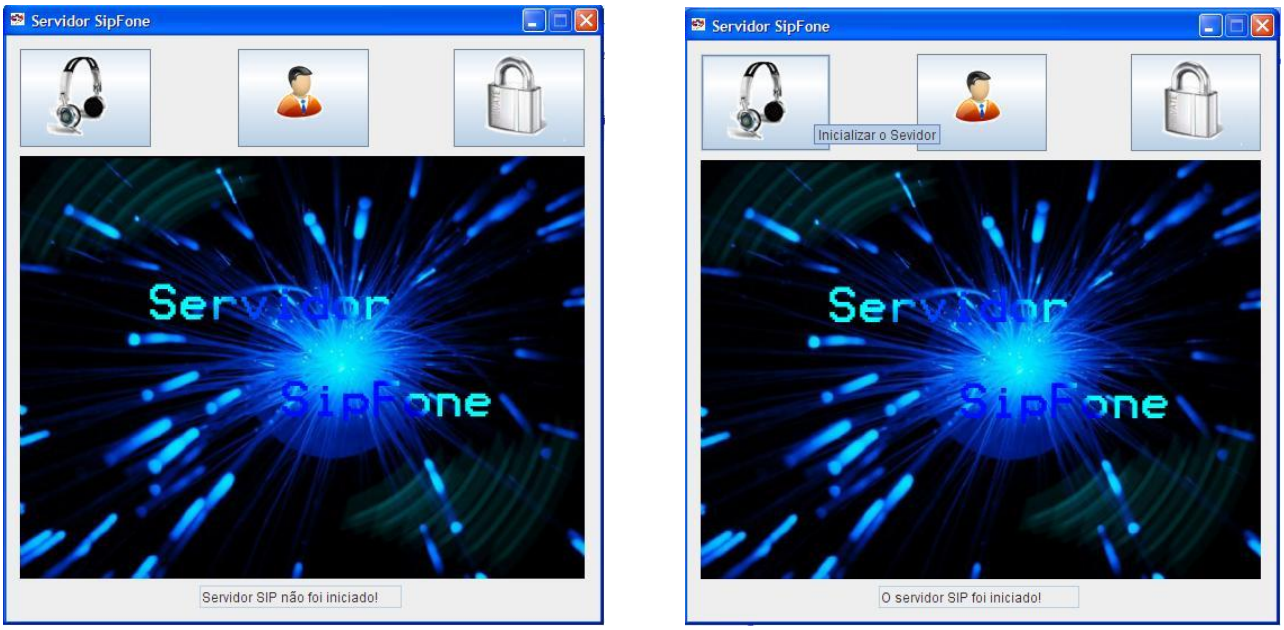

Figura 10: Interface gráfica do Servidor Sip

11. CONCLUSÃO

Neste trabalho de iniciação científica foram aprimorados os conhecimentos sobre a transmissão de voz sobre IP utilizando o protocolo SIP. Esses conhecimentos foram de fundamental importância para o desenvolvimento do protótipo do Sip Fone, assim como o protótipo do Servidor Sip, onde ambos encontra-se na versão α.

A interceção do conhecimentos da linguagem de programação Java, do protocolo SIP, da API JMF e da biblioteca Jain-SIP, resultou no desenvolvimento do aplicativo Sip Fone e no Servidor SIP. A interceção desses conhecimentos abre novos horizontes, tais como o desenvolvimento de aplicativos que além da transmissão de voz, podem transmitir vídeo e fazer sessões multicast em tempo real.

REFERÊNCIAS

SUN MICROSYSTEMS. **Javaª Media Framework API Guide**. California, U.S.A: Garcia Avenue, Mountain View, 1999.

TANENBAUM, Andrew S. **Redes de Computadores**. Rio de Janeiro: Campus, 1997.
DANG, Luan. JENNINGS, Cullen. KELLY, David. **Pratical VoIP Using Vocal**. 1. ed. O.Reilly & Associates, Inc. 2002.

MEDEIROS, Ernani. **Desenvolvendo Software com UML 2.0**. 1. ed. São Paulo: Pearson Makron Books, 2004.

Honório, P.E.. **Projeto de desenvolvimento de telefonia IP utilizando SIP**. Trabalho de conclusão de curso – Universidade Estadual de Londrina, Londrina, 2004.

MANSUR, Andre Gonçalves; HOMMERDING, Roberto. **Implementação didática de telefone VoIP por software utilizando protocolo SIP.** 2006. 62 p. Monografia (Graduação) – Curso de Tecnologia em Eletrônica, UTFPR, Curitiba, 2006.

SAMPAIO, Cleuton. **TCP/IP e Intranets**. 1. ed. Rio de Janeiro: Brasport, 1997.