Serviço Robusto de *Web-To-Dial* Baseado em SIP e Java Applet

Cesar A. C. Marcondes[1], **Paulo H. de Aguiar Rodrigues**[2]

Departamento de Ciência da Computação/IM e NCE/UFRJ
Caixa Postal 2324, CEP 20001-970
Rio de Janeiro – RJ – Brasil

Resumo

O futuro das telecomunicações está na convergência das redes de dados e das redes de voz. Para isso, as comunicações de voz baseada na tecnologia IP vêm se tornando uma alternativa cada vez mais viável, dado o desenvolvimento de novos protocolos de rede que permitem dar suporte à transmissão de áudio em tempo real, garantir a qualidade de serviço na transmissão, e estruturar arquiteturas de telefonia baseadas na Internet. Apresentamos a implementação e implantação de um serviço *Web-to-Dial*, um telefone SIP básico baseado na tecnologia *Java Applets* sob ambiente de execução *Java Plug-in*, tendo, como características importantes, a robustez no tratamento de mensagens de sinalização e a qualidade da captura, recepção e transmissão da mídia. Testes de interoperabilidade com outros clientes e servidores SIP são apresentados.

Palavras-chave: Internet: protocolos, serviços e aplicações; Telefonia IP; Serviços de Telefonia IP; Web-to-Dial, Java Applet, SIP.

Abstract

The future of telecommunications resides in the convergence of data and voice networks. Voice communications based on IP technology are becoming a viable alternative, supported by the development of new network protocols allowing real-time audio transmissions and deployment of IP telephony architectures with QOS guarantees. We present the implementation and deployment of a Web-to-Dial telephony service, a SIP telephone based on Java Applets running under Java Plug-in, and having special features as robust signaling message handling and extended quality in capture, reception and media transmission. Interoperability tests with other SIP clients and servers are presented.

Keywords: Internet: protocols, services and applications; IP Telephony Services; Web-to Dial, Java Applet, SIP.

1. INTRODUÇÃO

Hoje, no contexto internacional, as telecomunicações vêm experimentando uma enorme mudança estrutural, onde as vantagens da tecnologia de comutação de pacotes têm despertado grande interesse tanto nas comunidades da Internet como no ambiente corporativo das operadoras telefônicas convencionais. Dentre as razões para esta convergência, podemos

[1] Mestrando do Programa de Pós-Graduação em Informática do IM/NCE, e-mail: cesar@posgrad.nce.ufrj.br.

[2] Professor do Departamento de Ciência da Computação/IM e analista do Núcleo de Computação Eletrônica/UFRJ, e-mail: aguiar@nce.ufrj.br.

citar a possibilidade de uma melhor utilização dos meios físicos devido à multiplexação estatística e à viabilidade do transporte integrado de dados, vídeo e voz.

No Brasil, a tecnologia ATM tem sido largamente usada para viabilizar a implementação de backbones IP de alta velocidade, sendo exemplos a Intelig, ATL e RNP [1]. Pelas próprias características do ATM que suporta nativamente QoS para diferentes fluxos, obtém-se uma plataforma tecnológica capaz de agregar dados, telefonia e televisão, e propiciar para os provedores de serviços novas demandas de aplicações como telefonia IP, vídeo sob demanda e aulas virtuais. A evolução do suporte a QoS ao nível do TCP/IP irá permitir que estas mesmas aplicações sejam viabilizadas diretamente pela arquitetura IP. No caso de voz sobre IP (VOIP), esta nova forma de implementar a telefonia, que foge do esquema tradicional da taxação de pulsos da telefonia pública, é muito interessante seja pela economia de custos seja pelos novos serviços inteligentes e versáteis que ela propicia.

Com a agregação de fluxos sobre uma rede única, teremos uma evolução do cenário atual, aonde os navegadores irão gradativamente assumindo o papel de agentes de comunicação e informação. Em pouco tempo, o uso de telefonia será transparentemente incorporado aos navegadores com capacidade multimídia, com os quais poderemos comunicar com outros usuários que estejam conectados tanto na Internet, quanto na malha de telefonia convencional, ou na rede de telefonia celular. A arquitetura de suporte a telefonia heterogênea terá a tarefa árdua de oferecer a qualidade de serviço que os usuários almejam.

Para a completa transposição da telefonia convencional para a telefonia VOIP, é necessário atingir níveis adequados de disponibilidade, confiabilidade e qualidade fim-a-fim (influenciada, principalmente, por atraso, variância do atraso e perda de pacotes). A telefonia IP também pode oferecer serviços adicionais como roteamento de chamada personalizado, o que na Internet poderia estar agregado a Web, ou e-mail, ampliando a versatilidade e a sofisticação do ambiente telefônico.

Com várias vantagens estratégicas, a indústria da telefonia IP vem crescendo bastante nos últimos anos. Um negócio que vem chamando bastante atenção são os ITSPs (*Internet Telephony Service Providers*), que são provedores destinados a oferecer serviços de voz sobre IP para usuários, principalmente a partir da Web (ex. *Net2Phone*, *DialPad*). Estes serviços de voz sobre IP incluem desde fax internacional a serviços de *gatewaying* com a telefonia convencional em vários países, propiciando um novo tipo de operadora telefônica virtual. Dentre os serviços oferecidos por estes provedores, o mais difundido é o serviço *Web-to-Dial*, sendo este um software acessado via interface Web, que permite a um usuário usar telefonia IP diretamente de dentro do site do provedor, fazendo desnecessária a instalação de um cliente VOIP no computador do usuário.

Um software de telefonia IP agregado a uma página Web oferece uma enorme flexibilidade. Do ponto de vista do usuário, ele poderia acessar um telefone deste tipo em qualquer local, mesmo em viagens, e realizar a ligação diretamente da Internet. Do ponto de vista do provedor, este poderia montar uma central de atendimento ao cliente totalmente voltada para a Internet, sem a necessidade da aquisição de 0800. Outras vantagens incluem o gerenciamento inteligente de usuários, facilidade da atualização online da versão de software, descoberta e coleta automática de *bugs*, e utilização da infra-estrutura de rede montada pela operadora virtual para que o serviço IP na Web possa acessar pontos de troca de tráfego com telefonia convencional, otimizando custos envolvidos em ligações interurbanas.

Este artigo descreve a implementação de um serviço *Web-to-Dial*, tendo como características importantes: a robustez no tratamento de mensagens de sinalização, a interoperabilidade com a arquitetura SIP, e a qualidade da captura, recepção e transmissão da mídia. O artigo está organizado em oito seções. Na Seção 2, descrevemos detalhes da implementação deste serviço pelo uso de *Java Applets* [2] e questões de segurança envolvidas. Na seção 3 detalhamos os módulos do *software* e os paradigmas de programação usados. Discussão sobre a robustez do segmentador de protocolo SIP é apresentada na Seção 4. Na Seção 5, discutimos sobre formas de captura de mídia pelo navegador. Na Seção 6, descrevemos os testes de conformidade SIP, e finalmente na Seção 7, mostramos uma arquitetura SIP montada no laboratório que permite a um usuário navegando na Internet realizar ligações aos ramais telefônicos da UFRJ. As conclusões são apresentadas na Seção 8.

2. IMPLEMENTAÇÃO DE SERVIÇO WEB TO DIAL EM JAVA

Uma das primeiras decisões de projeto foi optar pelo uso do protocolo de sinalização de chamada padronizado pela IETF, o SIP [3], que permite ao usuário interagir diretamente com qualquer outra implementação - gateway, servidor ou cliente - baseada em SIP. Outro ponto favorável para a escolha do SIP é o fato de ser baseado no protocolo HTTP, que possibilita o uso de mime-types e, com isso, o transporte em suas mensagens de qualquer tipo de sinalização, mesmo de outras páginas Web ou arquivos quaisquer, tornando o SIP facilmente integrável a aplicações na Internet.

Para embutir o software de telefonia IP numa página Web, utilizamos a tecnologia de *Java Applets* sob o ambiente de execução *Java Plug-in* [4]. A vantagem do *Java Plug-in* é que sua instalação é simples, transparente, e feita sob demanda, quando o usuário acessa a página contendo o serviço. Além disso, ele permite usar no navegador todas as novas bibliotecas do Java. A outra possível opção, o uso da própria máquina virtual Java (JVM) embutida nos navegadores, não permitiria trabalhar com bibliotecas de programação Java 2 (como a Swing), presentes nas novas versões do Java Runtime Environment, limitando certas funcionalidades.

Quando a applet é executada, ela automaticamente entra no ambiente de execução Java-Plugin ao invés de ativar a máquina virtual padrão embutida no navegador. Para que isso aconteça, é necessário inserir algum código HTML especial naquela página Web. Para gerar este código HTML especial, usamos a ferramenta de conversão *Java Plugin Converter,* distribuída com o kit de desenvolvimento Java.

2.1 Solução para o Problema de Segurança no uso de Java Applets

O uso da tecnologia *Java Applets* tem muitas implicações nos níveis de segurança do software embutido na página e que roda em cooperação com os navegadores. Realizar uma chamada partindo da Web para outro IP qualquer requer que a applet tenha permissões especiais dadas pelo navegador. O modelo de segurança utilizado pelo *Java Plug-in* (ambiente de execução da *applet*) faz restrições ao desenvolvimento de programas que precisam acessar os recursos de I/O e de rede, além de limitar o uso amplo de *threads* nas *applets*.

Para que o serviço Web-to-Dial pudesse ultrapassar estas restrições, optamos pelo uso de certificados eletrônicos autogerados [5], como forma de assinar o código da Java Applet e, então, ter completa liberdade para acessar os recursos locais. A opção do uso pela applet de objetos de segurança da JVM embutidos nos navegadores, em oposição ao uso dos

certificados, tornaria a programação totalmente dependente do navegador, o que não é adequado ao nosso ambiente. Por outro lado, o uso de certificados autogerados é a forma usada por desenvolvedores para evitar o custo de obtenção de um certificado eletrônico da *Verisign* ou *Thawte*, nas fases iniciais de um projeto.

Uma vez que o código Java desenvolvido foi incorporado a um *container* de objetos JAR (*Java Archive*) e devidamente assinado (usando as ferramentas *keytool* e *jarsigner* do kit de desenvolvimento Java), seu uso dentro de uma página HTML, convertida para usar tags de *Java Plug-in*, faz com que o navegador carregue uma máquina virtual externa e seja apresentada uma caixa de diálogo sobre a assinatura deste código (Fig. 1). O usuário deve escolher a opção "*Grant*", que liberará a applet de todas as restrições. Um cuidado especial que deve ser tomado no uso de certificados é verificar a ausência no arquivo de políticas de segurança do JRE (*Java Runtime Environment*) da entrada: *permission java.lang.RuntimePermisssion "usePolicy"*. Se ela existir, os certificados não terão efeito de liberação algum. Como o *Java Plug-in* é uma tecnologia independente de navegador, não é necessário prover certificados diferentes de assinatura de código para cada navegador.

Fig. 1 – Decisão sobre liberar os recursos usando Certificado Autogerado

3. DETALHAMENTO DOS MÓDULOS DO CLIENTE SIP USANDO APPLET

O serviço *Web-to-Dial* é na verdade um telefone IP SIP básico embutido no site. Seguindo a classificação de cliente básico SIP descrita por [6], nossa implementação em applet (Fig. 2) suporta, rigorosamente, a lista de capacidades abaixo:

1- Envia/recebe INVITE sobre UDP,
2- Gera resposta ACK apropriadamente,
3- Dá opção ao usuário de aceitar e rejeitar chamadas telefônicas IP,
4- Monta cabeçalho SDP [7] contendo pelo menos um codificador de mídia,
5- Trata apropriadamente os cabeçalhos *To*/ *From*/ *Call-ID*/ *Cseq*/ *Via*/ *Content-Length*/ *Content-Type*,
6- Realiza a geração de tag contendo o descritor da chamada no cabeçalho *To*,
7- Envia e recebe mensagem básica de terminação de chamada BYE via UDP,
8- Permite o uso da forma compactada dos cabeçalhos SIP,
9- Rejeita métodos desconhecidos com respostas 501 (*Not Implemented*), e
10- Envia e recebe mídia de áudio via RTP, possivelmente sem o uso de RTCP.

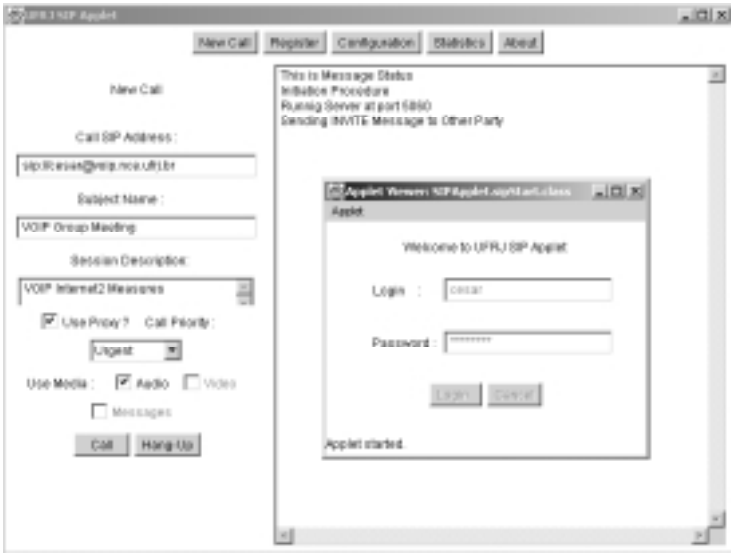

Fig. 2 – Implementação UFRJ SIP Client

Podemos dividir nossa implementação em 6 módulos básicos: Módulo de Conexão ou Transporte da Sinalização, Módulo Gerador de Mensagens SIP, Módulo Segmentador de Mensagens SIP, Módulo de Transmissão e Recepção de Mídia via RTP, Módulo Gerenciador da Máquina de Estados SIP e Módulo de Interface com o Usuário.

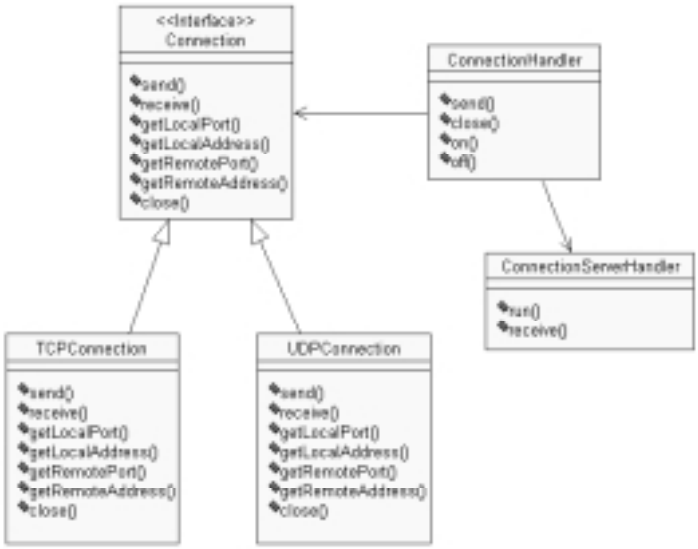

Fig. 3 – Diagrama de Classes do Módulo de Transporte da Sinalização SIP

Na implementação dos módulos houve uma preocupação constante em utilizar as melhores práticas na programação Java. Assim, na programação do módulo de transporte de sinalização SIP, o uso da interface *Connection* (Fig. 3) faz com que a sinalização possa ser feita sobre UDP ou TCP. Neste módulo, temos os threads (*ConnectionHandler* e *ConnectionServerHandler*) de envio e recepção de mensagens. Os métodos *On( )* e *Off ( )* de gerenciamento da conexão são usados para evitar a transmissão e recepção simultânea pela *Connection*, operando como semáforos no acesso concorrente. Uma possível extensão deste modelo para o gerenciamento de mais de uma chamada, interessante do ponto de vista de um

servidor SIP, seria a criação de um *pool* de *Connections*. É relevante comentar que a classe *BufferedReader*.é a mesma estrutura em Java usada para receber tanto datagramas UDP quanto dados TCP, pacotes que são armazenados e lidos de maneira diversa.

Os módulos de geração e segmentação de mensagens foram implementados com codificação distinta. Para a geração foi usado um codificador de mensagens rápido baseado em vetor, contendo todos os cabeçalhos SIP em posições fixas. Com esse vetor é possível montar eficientemente a mensagem SIP e também fazer uso de compactação de cabeçalhos (Fig. 4). Mas na segmentação (*parsing*) das mensagens, preferimos usar a biblioteca desenvolvida pelo NIST [8], parte da implementação oficial do módulo SIP-JAIN [9]. Esta biblioteca propicia uma forma mais robusta e flexível de tratamento de mensagens SIP, pois está baseada na gramática ABNF do SIP (veja Seção 3), e opera corretamente frente a cabeçalhos alterados e/ou inexistentes.

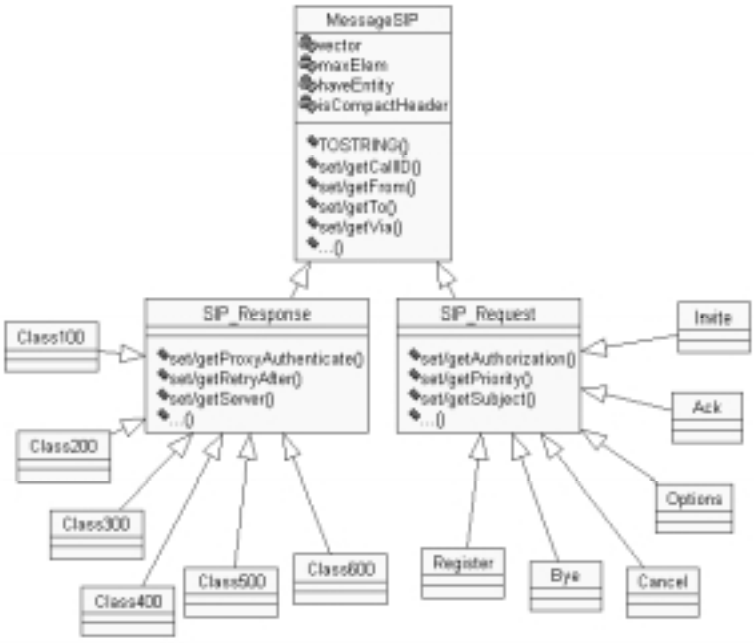

Fig. 4 – Diagrama de Classes do Módulo de Geração de Mensagens SIP

O módulo de transmissão e recepção de Mídia via RTP/UDP usa a biblioteca de manipulação de mídia *Java Media Framework* versão 2.1.1 [10]. Esta biblioteca tem uma ampla variedade de codificadores/decodificadores de mídia (como o G.711, GSM e MPEG Layer III Audio), permite captura de áudio a partir de um applet, possibilita configurar os buffers de compensação de jitter e realizar o *playout* de mídia gravadas (como o som de alerta do telefone IP). A biblioteca ainda oferece a possibilidade de se adicionar um componente de exibição gráfica, que permite visualizar estatísticas encaminhadas pelos pacotes RTCP, o protocolo de controle do RTP. Estas estatísticas contêm relatórios periódicos sobre: variação do atraso (jitter), número de pacotes recebidos, número de pacotes perdidos, número de pacotes perdidos por atraso, entre outros.

Um possível módulo estatístico baseado nas informações RTCP obtidas pelo *JMF* ainda não foi finalizado, mas pretendemos completá-lo e seguir os padrões de RTP MIB para

armazenar as estatísticas obtidas, como o realizado pela ferramenta de monitoramento descrita em [11]. Um ambiente de gerência poderia ser implementado usando um applet remoto que recolheria as estatísticas localmente da máquina do cliente, as quais seriam, posteriormente, enviadas ao servidor onde reside o applet, de modo a termos um panorama da qualidade das ligações que os usuários estão experimentando.

O módulo gerenciador da máquina de estados SIP (Fig. 5) recebe, das classes de transporte e da interface gráfica, eventos que disparam os métodos *ProcessRequest* e *ProcessResponse*, sem conhecimento de qual máquina de estados estará atuando sobre estes métodos. Usamos o padrão de projeto *Command* [12] para encapsular a requisição, e o padrão de projeto *Mediator* [12] para mediar a comunicação entre a interface gráfica, o gerenciador de máquina de estados, e o módulo de transporte da sinalização. Internamente aos objetos de FSMUA (*Finite State Machine User Agent*) temos uma máquina de estados finita, representando as transações de estado do SIP. No caso do *InviteFSMUA*, por exemplo, cobrimos todos os possíveis estados dados pela seção *INVITE Client Transaction* da RFC 2543 [3].

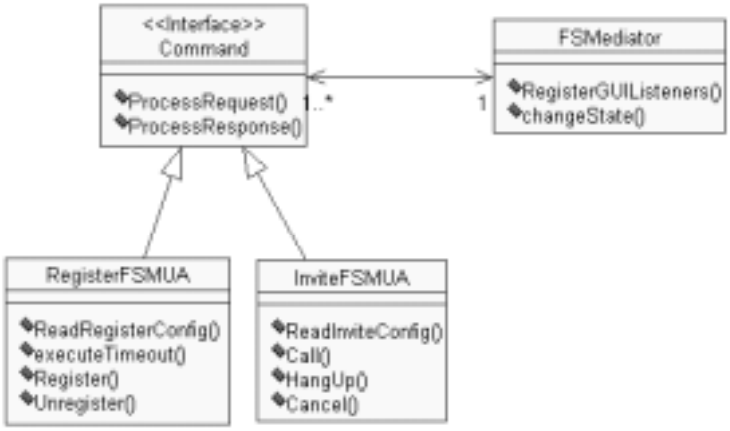

Fig. 5 – Gerenciamento da Máquina de Estados SIP

Por fim, o módulo da interface com o usuário (vide Fig. 2) tem 4 painéis integrados, cada um com seus próprios conjuntos de botões, campos de formulário e opções. São eles: New Call (Nova Chamada), Register (Registro), Configuration (Configurações do Usuário) e Statistics (Estatísticas). A captura dos eventos de clique de botões e preenchimento de campos é feita pelo uso de interfaces *Listeners.* Estas interfaces estão automatizadas no construtor de interface gráfica da ferramenta *JBuilder* da *Inprise*, que foi usada no desenvolvimento deste projeto. Outra ferramenta utilizada foi a *Rational Rose* [13] de modelagem de diagramas da UML (Unified Modeling Language), de onde foram extraídas as figuras mostradas nesta seção.

Dentre os módulos implementados, alguns, como o segmentador de mensagens (*parser*) e o de transmissão e recepção de mídia, têm um impacto muito grande sobre o tempo de comunicação total. Por isso, nas próximas seções, analisaremos detalhadamente questões de performance relacionadas com estes módulos.

4. SEGMENTADOR DE MENSAGENS GERADO PELA GRAMÁTICA SIP ABNF

O SIP usa, assim como muitas especificações técnicas da IETF, uma versão modificada da notação BNF (*Backus-Naur Form*), chamada ABNF [14] (*Augmented BNF*), para formalizar matematicamente seus protocolos. Esta notação, similar à EBNF (Extended BNF), tem como objetivo descrever sem ambigüidade a sintaxe de uma linguagem. Ela é usada para definir uma gramática do protocolo, e possibilitar a construção mecânica de ferramentas de segmentação (*parsing*), específicas para este protocolo.

A biblioteca NIST-SIP foi montada sob uma estrutura de compilador de compiladores (semelhante ao YACC – *Yet Another Compiler Compiler*, popular gerador de *parser* em C++ para Unix). A base para a construção do segmentador (*parser*) SIP desta biblioteca é oriunda do projeto ANTLR [15], que possibilita a construção de *parsers*, *tree-parser*, e *lexers***,** suportados por gramáticas EBNF (Extended BNF) do tipo *top-down LL* (a maneira mais fácil de fazer *parsing*) [16]. Isso torna o processo de *parsing* bastante robusto, embora com custo computacional elevado. Para os propósitos do JAIN [9], que é o desenvolvimento de aplicações por terceiros para redes de operadoras, e para qual finalidade a NIST-SIP foi construída, era preciso realmente garantir esta robustez superior.

Quando iniciamos o desenvolvimento do cliente SIP, a biblioteca NIST-SIP realizava apenas a segmentação das mensagens, não suportando a sua geração, que teve que ser feita em codificação própria. Em 2001, o projeto NIST-SIP também disponibilizou sua versão do montador de mensagens SIP a partir de seus objetos (*SIPHeaders* e *SIPMessage*). Constatamos, no entanto, que o atraso das mensagens geradas no início da chamada pela NIST-SIP era pior do que o observado no esquema simplificado que havíamos desenvolvido. Os tempos medidos estão mostrados na Tabela 1.

Tipo de Mensagem SIP	Algoritmo de Construção de Mensagem usando Vetor e preenchendo os seus índices com as *Strings* das linhas de cabeçalho (UFRJ)	Algoritmo de Construção de Mensagem usando criação de objetos *SIPHeader* e posterior codificação de todos os *headers* em sua forma canônica (NIST-SIP)
INVITE inicial	300 iterações : tempo de 548,04 ms	300 iterações : tempo de 1443,29 ms
ACK após o recebimento de 200 OK	300 iterações : tempo de 670,03 ms	300 iterações : tempo de 661 ms

Tab. 1 – Resultados do Uso do Montador de Mensagens NIST-SIP em comparação com o montador de mensagens simplificado da implementação SIP Applet (AMD K7/256 MB RAM/média de 30 experimentos)

Na tabela acima, a diferença maior observada entre desempenhos na montagem da mensagem INVITE inicial pode ser explicada pela maneira com que os algoritmos trabalham. No nosso módulo gerador de mensagens, preenchemos os cabeçalhos SIP em um vetor de tamanho fixo, cada cabeçalho tendo uma posição única no vetor (Fig. 4). Obtivemos o tempo de 548,04 ms em 300 iterações de geração (usamos 300 iterações para ressaltar a diferença, que é pequena no nível de uma mensagem, mas pode se tornar um problema se estivermos trabalhando dentro de um servidor). Já no algoritmo usado pela NIST-SIP, cujo tempo em 300 iterações foi de 1443,29 ms, é preciso criar primeiro os objetos que formam cada cabeçalho e configurá-los apropriadamente. Depois, estes cabeçalhos são juntados para formar o objeto da mensagem SIP, que é então codificada para sua forma canônica (em texto), passando por muitos processos de criação de objetos. Por outro lado, em nossa implementação somente um

objeto é criado. Com o uso de um vetor fixo, portanto, se ganha em velocidade, mas perde-se em robustez e validação de texto.

Em Tab. 1, a vantagem na construção da mensagem SIP usando nossa abordagem, ocorre somente quando estamos trabalhando com mensagens iniciais SIP. No caso do processamento de uma mensagem intermediária (ACK após o recebimento do 200 OK) que passa pelo *parsing* NIST-SIP, a construção da mensagem de resposta fica bastante facilitada, pois os objetos dos cabeçalhos (*SIPHeaders*) já foram construídos pelo próprio processo de segmentação, estando prontos e configurados. A opção de copiar estes objetos e incorporá-los à nova mensagem SIP, como feito pela NIST-SIP, é mais eficiente do que extrair campo por campo em formato texto e preencher o vetor com a mensagem SIP. A NIST-SIP levou cerca de 661 ms para montar uma mensagem ACK, enquanto que nosso algoritmo, contando o tempo de extração dos textos dos cabeçalhos da NIST-SIP, levou 670,03 ms. A reutilização dos cabeçalhos SIP nas mensagens intermediárias ocorre constantemente, durante toda a transação da chamada telefônica IP.

Como as duas formas tem suas vantagens e desvantagens, optamos por usar o nosso Montador apenas para mensagens iniciais (INVITE, REGISTER), e usar o Montador de Mensagens NIST-SIP para montar mensagens que são intermediárias ao estabelecimento da chamada (200 OK, ACK, BYE). Bem recentemente, o projeto NIST-SIP adotou um esquema mais rápido de codificação de mensagens iniciais, que diminui o número de objetos criados, e usa um padrão de projeto *Factory* [12] para criar, de uma só vez, um cabeçalho e preenchê-lo com uma *string*, ao invés do método anterior de configuração de subcampos de cada cabeçalho.

5. CAPTURA DE MÍDIA PELO NAVEGADOR

Outro ponto crítico na implementação do cliente SIP foi resolver a questão de como capturar som a partir de um applet. A *Sun Microsystems*, mantenedora do Java, trabalha atualmente com duas APIs de manipulação de som: *JavaSound* e *JMF*. Estas duas bibliotecas têm soluções para a captura de áudio na Internet. Além da performance da biblioteca, a questão da segurança também foi um fator que influenciou nossa decisão.

A API *JMF* na sua versão 2.1.1a [10] provê ferramentas próprias para a configuração das permissões de segurança para a gravação de áudio pela Internet sem precisar trabalhar com certificados eletrônicos de terceiros. No caso da *JavaSound*, incorporada ao kit de desenvolvimento Java versão 1.2+, seria possível a captura de áudio pelo uso de outros certificados eletrônicos. Entretanto, ela não tem qualquer mecanismo para transmissão RTP, nem possibilita a conversão de codificadores/ decodificadores de áudio com a mesma facilidade que a *JMF*. Existe uma tendência de que em breve a Java Sound e a *JMF* se tornem uma só biblioteca. Para ilustrar esta tendência podemos mencionar que atualmente temos métodos nas duas bibliotecas de classes com mesmo nome, e que podem causar conflitos.

Outras diferenças entre a *JMF* e a *Java Sound* recaem na velocidade e nos tipos da captura suportados. Como a *JavaSound* (*javax.sound*) é uma biblioteca incorporada ao JRE, foi desenvolvido um esquema via máquina virtual para possibilitar a captura de áudio. Já a *JMF* usa rotinas de código nativo em C++ (para Linux, Windows e Solaris) para realizar esta mesma captura diretamente do hardware, seja de áudio, de vídeo, ou outros dispositivos, passando por cima do processamento da máquina virtual, aumentando a performance.

Posteriormente, a *JMF* trabalha com os dados capturados, encapsulando em uma interface JNI (*Java Native Interface*), e preparando este fluxo de mídia para ser transmitido via RTP/UDP.

Em nossa implementação, desenvolvemos um transmissor e receptor de áudio baseado na *JMF* (Fig. 6). A classe *AudioBox* é quem dispara a execução dos objetos *AudioTransmit* e *AudioReceive* simultaneamente, assim que na máquina de estados atingirmos o estado de chamada estabelecida. Isso acontece logo após a fase de negociação de mídias SIP (envio de ACK e recebimento de 200 OK), onde os parâmetros da mídia passados pelos cabeçalhos SDP [7] *Connection (c)* e *Media (m)* negociados, contemplam endereço IP, porta UDP e tipo de mídia que estará sendo transmitida e recebida.

Atualmente, estamos suportando apenas áudio, mas as classes foram projetadas para receber outros tipos de mídia como vídeo. A classe *AudioReceive* também provê a capacidade de detectar, ao nível de fluxo RTP, mudança de formato de codificação pela parte remota. Por exemplo, se o outro cliente baseado em estatísticas RTCP resolver trocar o codificador de mídia para preservar a qualidade da transmissão, então o applet detectará esta mudança no cabeçalho Payload Type RTP, e iniciará um processo de re-INVITE, onde ocorrerá nova negociação das capacidades, sem que o estado da chamada seja alterado.

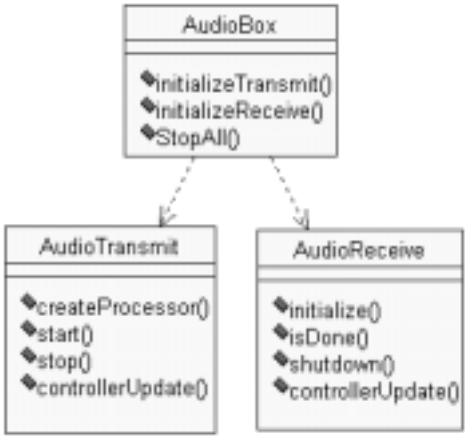

Fig. 6 – Classes de Transmissão e Recepção de Áudio

Nossa opção de desenvolvimento conforme visto anteriormente é pelo uso da *JMF* no módulo de transmissão e recepção de áudio. A *JMF* possui várias vantagens, como maior facilidade de manipulação da mídia (configuração de *buffers* de compensação de jitter, *buffers* de playout, codificadores, multiplexação de mídia) e de manipulação do RTP (captura de eventos como a detecção de mudança de payload e parâmetros RTCP). Entretanto, esta decisão tem uma desvantagem: os engenheiros da *Sun* não desenvolveram ainda um instalador silencioso e por demanda da *JMF*. A recomendação deles é realizar o *download* e instalar diretamente a *JMF* com código nativo (chamado de *performance pack*) na máquina local do cliente. O usuário deve, então, habilitar, através do aplicativo *JMFRegistry*, a opção de captura de áudio pelas *applets*, conforme Fig. 7.

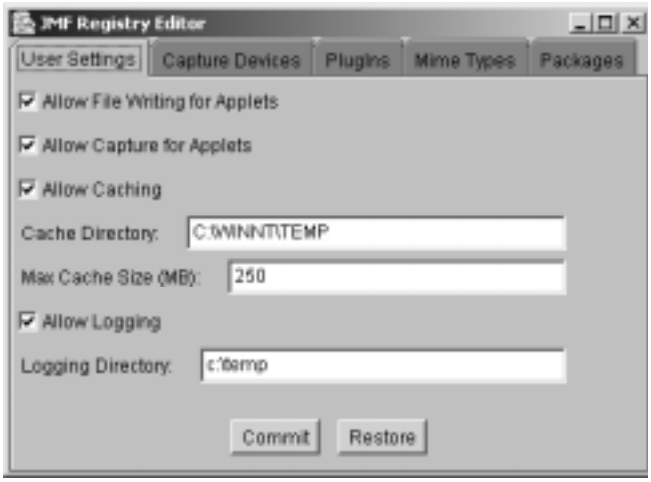

Fig. 7 – Utilização do JMF Registry para habilitar captura de áudio pela Applet

Em agosto de 2001, a *Sun* disponibilizou o código fonte da biblioteca *JMF* e suas aplicações, incluindo o *JMFRegistry*. Desenvolvemos uma análise sobre o processo de captura via *applet* e verificamos que existe uma dependência muito grande das classes do performance *pack JMF* com os códigos nativos de cada plataforma operacional (Linux, Solaris e Windows). Seria portanto necessário copiar este código nativo correspondente, por demanda (como o *Java Plug-in*), para a máquina do cliente. Estamos pesquisando maneiras de fazer isso. A opção usada no projeto até agora é a instalação direta do pacote *performance pack JMF*. Em breve estaremos comprovando experimentalmente se a solução com uso da javax.sound.*, comparativamente com a *JMF*, tem desempenho suficiente para integração com a telefonia convencional.

Realizamos algumas medidas para identificar gargalos no nosso código, como o tempo de preparação dos objetos da JMF no início da transmissão e da recepção de áudio. Os parâmetros básicos destes testes foram o uso do capturador de mídia *DirectAudio* ("*dsound*" ao invés de "*javasound*"), com taxa de amostragem 44100Hz, montando um fluxo RTP/µlaw de 8000 Hz (taxa de bits de 64 kbps), num computador AMD K-7 com 256 MB RAM.

Observamos que na classe *AudioTransmit* houve um atraso da ordem de 2 a 3 segundos entre o tempo de recebimento do 200 OK do estabelecimento de chamada e o início da transmissão de áudio. Este atraso é dependente de fatores como velocidade do processador, quantidade de memória e tempo na resolução de nomes (DNS), utilizado pelo método *addTarget* da classe *RTPManager*. Este último é usado para resolver o parâmetro RTCP *Canonical Name* (CNAME) que descreve os participantes de uma sessão multimídia. Entre estes tempos, o que mais contribui para o atraso do início da transmissão é o bloqueio no estado *Realizing/Realized* do objeto *processor*, onde é preparada para transmissão a mídia capturada e convertida para o formato PCM. Na primeira vez que a *applet* é carregada e se prepara para transmitir, este bloqueio tem maior duração, com cerca de 10 segs., até que sejam criados todos os objetos de codificação e o buffer de transmissão.

No caso da classe *AudioReceive*, o início da recepção ficou em torno de 2 segundos, também após o 200 OK. Apesar do método *addTarget* também estar presente nesta classe (método que contribui para o atraso na transmissão), ele só é acionado após o início da recepção do fluxo RTP, não impactando diretamente sobre o tempo entre o recebimento do 200 OK e o início da recepção de áudio.

Foram desenvolvidas algumas otimizações no serviço Web-to-Dial, para que o usuário minimizasse o tempo de transmissão de áudio fim-a-fim na Internet. Utilizamos, por exemplo, os objetos *DirectAudio, DirectAudioRenderer* (*JMF*) para diminuir o impacto da utilização da JVM neste processo, pois usam código nativo. Em seguida, implementamos um buffer de recepção com tamanho variável (em milissegundos) para que, de acordo com os requisitos da comunicação, pudéssemos otimizar em relação ao tempo de apresentação ou à fração de perda de pacotes por *jitter*.

Por fim, também é possível o usuário escolher qual codificador de mídia de áudio usar, de modo a observar requisitos de limitação de banda. Uma maior compressão de áudio pode ser conseguida em detrimento de um tempo maior na comunicação interativa, como no caso do uso do codificador MPEG Áudio Layer III.

6. TESTES DE CONFORMIDADE ENTRE IMPLEMENTAÇÕES SIP

A interoperabilidade de clientes e servidores SIP tem especial importância no grupo de trabalho SIP da IETF, sendo periodicamente realizadas conferências somente para testes entre várias implementações de empresas e universidades [17].

Fizemos uma bateria de testes de conformidade SIP entre nosso cliente SIP applet e vários clientes (Cliente SIP da Universidade Columbia v. 1.6, Cliente SIP da Ubiquity v. 2.0, Cliente SIP da Hughes v1.0) [18], servidores (Servidor SIP da Universidade Columbia e Servidor SIP do NIST) [18] e gateway SIP (Cisco 2600 [19]), usando computadores rodando Windows 2000. O objetivo destes testes foi mapear problemas de interoperabilidade entre nossa implementação e os demais *softwares*, usando a metodologia de testes estabelecida pelo IMTC (*International Multimedia Teleconferencing Consortium*) [20]. Estas normas descrevem cenários de compatibilidade entre entidades SIP. Cada cenário é descrito por: entidades envolvidas, detalhamento da seqüência de testes, e critérios de avaliação para completar o cenário com sucesso. Segundo esta metodologia [20] são cinco os cenários definidos para o SIP.

O primeiro cenário é definido como: comunicação ponto-a-ponto de áudio sem nenhum servidor. Nele são colocados dois clientes SIP em comunicação direta na mesma rede local. Eles devem seguir o ciclo do estabelecimento de chamada SIP (INVITE inicial, respondido com sinalização de respostas provisórias como 180 Ringing, negociação final da mídia com 200 OK e o ACK), e logo após deve ser avaliada a qualidade da mídia e a terminação da chamada por um dos clientes. Tab. 2 mostra os resultados de nossos testes.

	Cliente SIP Columbia – versão 1.6	Cliente SIP Ubiquity	Cliente SIP Hughes
Cliente SIP Applet (Web to Dial)	Funcionou bem nas várias tentativas diferentes. Entretanto apresentou instabilidade principalmente com a transmissão de mídia via RAT e recepção de BYE.	Funcionou bem nas várias tentativas. Tivemos, entretanto, que resolver um problema, que não deveria acontecer neste cliente, ao ser usado um caractere espaço a mais no cabeçalho *origin* do SIP.	Funcionou perfeitamente frente a vários testes diferentes.

Tab. 2 – Resultados de cinco observações dos testes no Cenário 1 de Interoperabilidade entre Clientes SIP (Ponto-a-Ponto) (segundo IMTC)

Uma observação a ser feita com relação ao teste acima é que o cliente de Columbia atualmente está na sua versão 1.7 (versão comercial) e não tínhamos licença para usá-lo. Portanto alguns dos erros encontrados podem ter sido resolvidos nesta nova versão.

Nos cenários 2, 3 e 4 que utilizam um servidor SIP como entidade de interoperação, nós usamos o Servidor SIP de Columbia 1.17, que armazena os registros em servidor *MySQL*, e outro servidor SIP escrito em Java usando a biblioteca NIST-SIP, que armazena os registros na memória. Ambos os servidores suportam as atividades de registro (alvo do cenário 2), de redirecionamento (alvo do cenário 3) e de proxy (alvo do cenário 4). Os resultados podem ser vistos em Tab. 3. A seqüência de testes de registro usando o endereço multicast 224.0.1.75 não pode ser concluída, pois o nosso *software* ainda não suporta esta característica. O endereço multicast, neste contexto, serve para que os clientes enviem mensagens REGISTER para qualquer servidor SIP associado ao endereço multicast, facilitando a busca do mesmo.

	Servidor SIP Columbia versão 1.17	Servidor SIP NIST
Cliente SIP Applet (Web to Dial)	Funcionou bem para o cenário local.	Funcionou perfeitamente com a applet, entretanto em testes complementares feitos com o SIP Columbia, estranhamente o cabeçalho *Contact* não pode ser segmentado pelo NIST-SIP.

Tab. 3 – Resultados de cinco observações dos testes nos Cenários 2, 3, 4 de Interoperabilidade entre Cliente SIP e Servidor SIP (segundo IMTC)

O último teste de interoperabilidade, o cenário 5, descreve uma seqüência de ligação entre o telefone IP e um telefone convencional. Utilizamos um roteador Cisco 2600 para realizar este teste. O *applet* fez a ligação com sucesso, entretanto notamos que o Cisco 2600 não está enviando mensagem BYE ao final da chamada, como recomendado pelo padrão SIP, quando desligamos o telefone na rede telefônica convencional. Logo algumas implementações de clientes não souberam tratar este comportamento.

7. USO DA APPLET COMO ENTRADA DE LIGAÇÕES AOS RAMAIS DA UFRJ VIA INTERNET

Como cenário macro, integramos o serviço *Web-To-Dial* apresentado ao nosso ambiente heterogêneo de telefonia IP (SIP/H.323) [21]. Fig. 8 mostra esta integração, que permite a realização de chamadas telefônicas para ramais da UFRJ a partir da Internet.

Um dos desafios na montagem desta arquitetura foi quanto à questão da segurança, pois o laboratório de VOIP é protegido por um filtro de pacotes (*firewall*) e poderíamos estar usando em nossa rede interna endereços IP não válidos, uma prática comum nas universidades. Para resolver esta situação, utilizamos a implementação do servidor SIP Proxy em *Java* desenvolvido pela NIST-SIP.

Este servidor tem a capacidade de lidar com aspectos de autenticação SIP, e, mediante esta autenticação, realizar o roteamento da chamada acionando mecanismos para atravessar o *firewall*, e possibilitando também o uso de Network Address Translation (*NAT*). Esses mecanismos são implementados neste servidor, como *scripts* em *Perl*, que se comunicam com o servidor e onde são enviados comandos para um *firewall* linux do tipo *iptables* [22]. Fizemos uma pequena alteração nestes *scripts Perl* para trabalharem com nosso *firewall* baseado em *ipchains*, traduzindo os comandos.

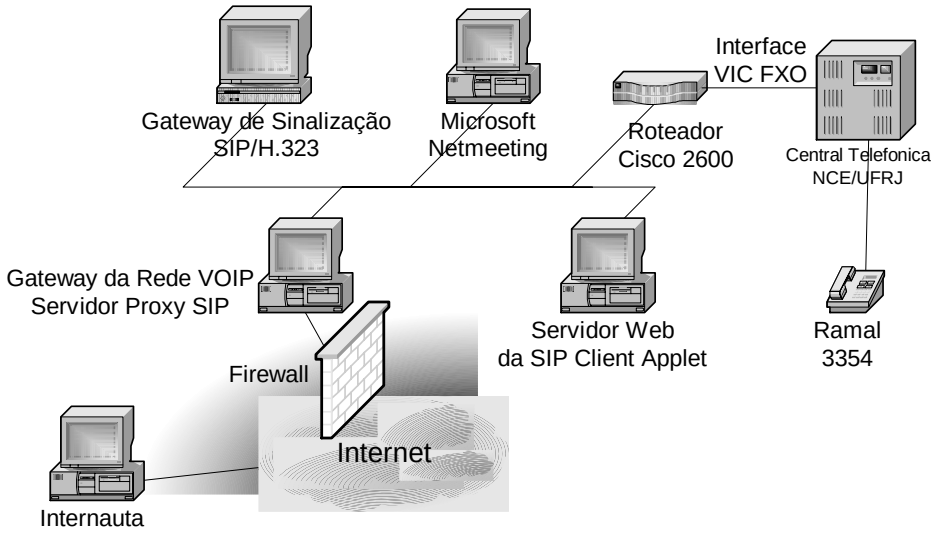

Fig. 8 – Arquitetura do Laboratório VOIP/UFRJ

O funcionamento é o seguinte: o servidor, quando usado no modo proxy por um usuário externo, obtém os parâmetros das portas de mídia UDP a serem abertas, através da segmentação do payload SDP das mensagens redirecionadas por ele, repassando essas informações para os *scripts Perl*, que inserem regras temporárias de abertura. Estas portas permanecem abertas no *firewall* durante a ligação, ou após um tempo máximo configurado no servidor.

O roteador Cisco 2600 (com interfaces de telefonia FXO e FXS e usando IOS versão 12.2 com suporte as mensagens SIP), presente na Fig. 8, permite fazer a interoperação com a rede telefônica convencional (PSTN), atuando como um Gateway SIP/PSTN. No Cisco, foi configurada uma interface FXO (interface de linha telefônica), conectada um ramal da central telefônica NCE/UFRJ (NEC NEAX 2400), para que pudessem ser roteadas ligações vindas da Internet para esta central. Criamos um plano de discagem simples (chamado pela Cisco de *dial-peer*), no qual uma mensagem SIP contendo uma URL 55212598-xxxx pode combinar com uma expressão regular de redirecionamento para uma das interfaces FXO da central telefônica (Fig. 9). Essa expressão regular dada por *destination-pattern*, restringe as chamadas telefônicas via Internet apenas para ramais da UFRJ, sendo os últimos quatro dígitos, o número do ramal.

```
voice rtp send-recv
!
voice-port 1/0/1
cptone BR
description Linha Telefonica FXO
!
dial-peer voice 6 pots
destination-pattern 55212598....
port 1/0/1
!
sip-ua
!
```

Fig. 9 – Configuração Básica do Cisco 2600 para Suporte SIP

Desta arquitetura, usando o serviço *Web-to-Dial*, é possível um internauta ligar para um ramal telefônico convencional via Web. Também é possível interoperar com um ambiente H.323, através do uso de nosso Gateway de Sinalização SIP/H.323 [21] para fazer a tradução da sinalização SIP para H.323, e permitir realizar uma ligação para um cliente H.323 como o *Netmeeting* da Microsoft.

8. CONCLUSÕES E TRABALHOS FUTUROS

Este artigo descreve a implementação de um serviço *Web-to-Dial*, baseado na tecnologia *Java Applets* sob ambiente de execução *Java Plug-in*, tendo como características importantes a robustez no tratamento de mensagens de sinalização, a interoperabilidade com a arquitetura SIP e a qualidade da captura, recepção e transmissão da mídia.

A implementação, apresentada na Seção 2, consiste de seis módulos básicos, onde houve a preocupação em se usar as melhores práticas para a programação Java. Para solucionar o problema de segurança no acesso aos recursos locais de rede e I/O, optou-se pelo uso de auto-certificados. O módulo de transmissão e recepção de mídia via RTP/UDP usa a biblioteca de manipulação de mídia *Java Media Framework* versão 2.1.1, que oferece um componente gráfico para exibição de estatísticas do RTCP e monitoração da qualidade de serviço. O transporte da sinalização foi implementado de modo a permitir que a sinalização ocorra tanto sobre UDP como sobre TCP, indistintamente. Para a montagem de mensagens SIP foi utilizada uma codificação própria baseada em vetor, que permite o uso de compactação nos cabeçalhos. Na segmentação das mensagens (parsing) intermediárias, optou-se pelo uso da biblioteca desenvolvida pelo NIST e que faz parte do módulo SIP-JAIN. Esta biblioteca propicia uma forma mais robusta e flexível de tratamento de mensagens SIP, pois está baseada na gramática ABNF e opera corretamente frente a cabeçalhos alterados e/ou inexistentes. O desempenho da segmentação de mensagens é avaliado na Seção 3 e justificado nossa opção de usar um montador próprio para as mensagens iniciais, do tipo INVITE e REGISTER, enquanto mensagens intermediárias são montadas usando o montador NIST-SIP. Na captura de mídia pelo navegador, nossa opção foi pelo uso da API *JMF* em oposição a Java Sound, pelos mecanismos de transmissão RTP e facilidade de conversão de codificadores/decodificadores presentes na JMF.

Os cinco cenários apresentados na Seção 4 mapeiam problemas de interoperabilidade entre nossa implementação de SIP applet *Web-to-Dial* e os clientes SIP de Columbia, Ubiquity, e Hughes, e com os servidores SIP de Columbia e NIST. Interoperação com gateway Cisco foi também testada. Os resultados mostram que alguns destes clientes apresentam deficiências. Na Seção 5 descrevemos o uso da applet em nosso ambiente heterogêneo de telefonia, demonstrando o completo sucesso da implementação.

No aspecto de captura de áudio falta avaliar se a solução com uso da javax.sound.*, comparativamente com a JMF, tem desempenho suficiente para integração com a telefonia convencional.

Como trabalho futuro, temos intenção de desenvolver um ambiente de gerência de VOIP que usaria o applet remoto para recolher estatísticas da máquina do cliente e enviá-las para um servidor central, armazenando as estatísticas segundo os padrões RTP MIB.

9. BIBLIOGRAFIA

[1] Adailton José Santos Silva, et al. **Modelos de Referência para Redes Metropolitana de Alta Velocidade ATM – Rumo à Internet2**. Relatório Técnico. Rede Nacional de Pesquisa. Rio de Janeiro 1998.

[2] B. Eckel. **Thinking in Java, 2nd Edition**, MindView Inc. Junho 2000.

[3] H. Schulzrinne, J. Rosenberg, et al. **IETF RFC 2543 – SIP: Session Initiation Protocol.** Outubro 2001.

[4] **Java Plug-in Home Page** (http://java.sun.com/products/plug-in/).

[5] M. Gallant. **Working Example Signed with Self-signed Certificate** (http://204.191.136.6 /~neutron/ testsscert / index.html). 2001.

[6] H. Schulzrinne. **Classification for SIP Interoperability Test Event**. (http://www.cs.columbia.edu/ ~hgs/ sip/ sipit/ classification.html). 2001.
[7] M. Handley, V. Jacobson. **IETF RFC 2327 – SDP: Session Description Protocol**. Abril 1998.
[8] **NIST-SIP API**. (http://snad.ncsl.nist.gov/proj/iptel/). 2001.
[9] R. Bhat e R. Gupta. **JAIN Protocols APIs**. IEEE Communications Magazine, Janeiro 2000.
[10] **Java Media Framework API Guide**. (http://java.sun.com/products/java-media/jmf/). Novembro 1999.
[11] F. C. Afonso. **Virtual Reality Transfer Protocol (VRTP). Implementing a Monitor Application for the Real-Time Transport Protocol (RTP) using the Java Media Framework (JMF)**. Escola Naval de Pós-Graduação, Tese de Mestrado em Ciência da Computação, Monterey, California, EUA, Setembro 1999.
[12] E. Gamma, et al. **Design Patterns – Elements of Reusable Object-Oriented Software**. Addison Wesley Longman, 1995.
[13] M. Fowler e K. Scott. **UML Essencial: um breve guia para a linguagem padrão de modelagem de objetos**. 2º Edição. Editora Bookman. Porto Alegre, 2000.
[14] D. Crocker, P. Overell. **IETF RFC 2234 – ABNF: Augmented BNF for Syntax Specifications**. Novembro de 1997.
[15] **ANTLR Website**. (http://www.antlr.org/)
[16] A. V. Aho, R. Sethi, e J. D. Ullman. **Compilers: Principles, Techniques and Tools**. Addison-Wesley, 1986.
[17] **SIPIT-SIP Interoperability Test Events.** (http://www.cs.columbia.edu/~hgs/sip/sipit/). 2001.
[18] H. Schulzrinne. **SIP Implementations**. (http://www.cs.columbia.edu/~hgs/sip/ implementations.html). 2001.
[19] **Voice Over IP for the Cisco 2600 Series**. (http://www.cisco.com/ univercd/ cc/ td/ doc/ product/ software/ ios113ed/ 113t/ 113t_3/ voip2600.htm)
[20] S. Dhulipalla, G. Meyer, C. Ross. **SIP Interoperability Scenarios Test Plan**. International Multimedia Teleconferencing Consortium. SIP SIG Activity Group. Junho 2000.
[21] B. Ribeiro, P. Rodrigues, C. Marcondes. **Implementação de Gateway de Sinalização entre Protocolos de Telefonia IP SIP/H.323**. SBRC 2001, Florianópolis, Maio 2001.
[22] **Linux iptables HOWTO**. (http://www.linuxguruz.org/iptables/howto/iptables-HOWTO.html)