# VoViWLAN

Aplicação de Transmissão de Áudio e Vídeo em Redes sem Fios

João Pedro Azevedo Soares

Universidade do Porto
Faculdade de Engenharia
FEUP

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Electrotécnica e de Computadores
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

Julho de 2006

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# VoViWLAN

João Pedro Azevedo Soares

Aluno do 5º ano da Licenciatura em Engenharia Electrotécnica e de Computadores pela
Faculdade de Engenharia da
Universidade do Porto

Trabalho realizado no âmbito do Projecto de Fim de curso na disciplina de PSTFC, do 2º semestre, do 5º ano, da Licenciatura em Eng. Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto, leccionada por João José Pinto Ferreira.

Projecto realizado sob a orientação do
Professor Doutor João Canas Ferreira,
do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto
e de
Engenheiro António Almeida
Mactek

Porto, Julho de 2006

# Resumo

A transmissão de voz sobre IP é cada vez mais utilizada hoje em dia, prevendo-se que aumente ainda mais nos próximos anos.

Pretendia-se desenvolver uma aplicação de voz e vídeo orientada às redes sem fios.

Para transmitir áudio e vídeo, é necessário capturá-los primeiro e comprimi-los. Utilizou-se, para isso, bibliotecas já existentes, nomeadamente o *EMIPLIB*, que por sua vez utiliza uma grande quantidade de bibliotecas.

Para desenvolver a aplicação, utilizaram-se protocolos considerados abertos, como o SIP e o SDP. Foi necessário a utilização de bibliotecas já existentes para implementar estes protocolos. As bibliotecas utilizadas foram o *eXoSIP* e o *oSIP*, que mostraram ser capazes de implementar a sinalização sem grandes problemas.

Achou-se importante adicionar algumas funcionalidades à aplicação, como a possibilidade de pôr um utilizador em espera, a implementação de um livro de endereços, etc.

Finalmente, sugere-se a continuação do desenvolvimento desta aplicação para melhorar alguns tipos de aspectos, como o funcionamento por detrás de NATs.

# Summary

Voice transmission is rising nowadays, and predictions tell us that it will have a boom very soon.

With this project the intention was to develop a software application to transmit audio and video oriented to wireless networks.

It is required to capture and encode (compress) audio and video to transmit them. So, it was used already made libraries, like *EMIPLIB* witch uses a big amount of other libraries.

To make the application, it was used open protocols, like SIP and SDP. It was necessary to use libraries to implement these protocols. The libraries used were *eXoSIP* and *oSIP* witch turned out to be capable of implementing signalization without any problems.

It was important to add other functionalities to the application, like the possibility to put a user on hold, the implementation of an Address Book, etc.

Finally, it is suggested the continuation of the development of this application to enhance some aspects, such as the working behind NATs.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Lista de Código

# Glossário

*ALSA – Advanced Linux Sound Architecture* – Componente do *kernel* do *Linux* que fornece drivers para os dispositivos de som.

API – *Application Programming Interface* – É uma interface que um programa ou biblioteca disponibilizam para outros programas poderem usar os serviços.

Codec – Compressor/Descompressor – Programa capaz de comprimir e descomprimir.

Diálogo – Ver Secção 4.1.1.2.

*DirectShow* – API para capturar e reproduzir média.

*ESD – Enlightened Sound Daemon* – Servidor de som para o *Gnome* e *Enlightenment.*

fps – *frames por segundo* – Número de imagens por segundo.

H.263 – *Codec* de vídeo ideal para videoconferências.

H.323 – Protocolo de sinalização de chamadas.

Host name – Nome único de um dispositivo ligado a uma rede.

IETF – *Internet Engineering Task Force* – Organismo da Internet que promove os standards da Internet.

IP – *Internet Protocol* – É um protocolo que permite entregar pacotes de uma fonte a um destino, através de várias redes interligadas.

IPv4 – Versão 4 do IP.

IPv6 – Versão 5 do IP.

ITU – *International Telecommunication Union* – Organização que responsável pela estandardização e alocação do espectro.

IYUV – Formato de representação da imagem.

*Jack* – Servidor de som que corre no MAC OS X e no *Linux.*

*Jitter* – Variação indesejada das características de um sinal.

*Kbps* – Kilobits por segundo – Taxa de transmissão de bits, normalmente associado a velocidade de transmissão.

LAN – *Local Area Network* – Rede Local

NAT – *Network Address Translation* – Técnica que consiste em reescrever os endereços de IP de origem de um pacote. Esta técnica é utilizada porque era previsto o fim dos endereços IP disponíveis para a Internet.

*OSS – Open Sound System* – Interface de som portátil.

RTP – *Real-time Transport Protocol* – Protocolo de transporte em tempo real.

SDP – *Session Description Protocol* – Protocolo de descrição de sessões.

SIMPLE – *Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions* – Protocolo que complementa o SIP para a utilização de mensagens instantâneas.

SIP – *Session Initiantion Protocol* – Protocolo de sinalização de sessões.

Speex – *Codec* de audio.

STUN – *Simple Traversal of UDP over NATs* **– P**rotocolo de rede que permite a clientes por trás de NATs descobrir o seu endereço público de Internet.

*Tranform coding* – Tipo de compressão de dados normalmente associado ao formato JPEG.

TU – *Transmission Unit* – Unidade de Transmissão.

URI - *Uniform Resource Identifier* – Identificador Universal de Recursos

*Video4Linux* – API de captura de vídeo para *Linux*.

*WinMM* – API de captura de áudio para *Windows*.

XCON – Extensão do protocolo SIP que permite ter conferências centralizadas.

# 1. Introdução

Com o crescente aumento da largura de banda das ligações à Internet, tem-se verificado um aumento de tráfego IP, aproveitando a velocidade das ligações para transmissão de dados em tempo real.

O volume de tráfego de dados já ultrapassou o volume de tráfego de voz. Uma vez que transportar dados tem um menor custo que transportar voz, há uma tendência para que o tráfego de dados aumente, visto que é possível (e barato) transportar voz como dados.

Nos últimos anos tem havido um grande aumento de utilizadores de banda larga, bastando para isso analisar os dados fornecidos pela ANACOM (1): no 1º trimestre de 2004 o número de clientes de banda larga por 100 habitantes era de 5,5% passando no 1º trimestre de 2005 para 9.0% e no 1º trimestre de 2006 já era de 12.5%.

Para além deste crescimento do número de acessos de banda larga, a própria largura de banda também tem aumentado significativamente, passando dos tradicionais 512 Kbps para os 2 Mbps ou mesmo para os 16 Mbps. Isto permite aos utilizadores aproveitarem a sua largura de banda para transmitir grandes quantidades de dados.

Com estas velocidades pode-se transmitir vários tipos de dados em tempo real, tais como voz e vídeo. Logo, o número de prestadores de serviços de voz sobre IP (VoIP) tem aumentado significativamente, bem como o número de clientes.

Nos últimos anos, tem-se assistido a um aumento da quantidade e da utilização de redes sem fios, devido, sobretudo, à evolução das tecnologias existentes e à redução dos preços, podendo-se cada vez mais transmitir a maiores velocidades.

Este projecto, insere-se no âmbito da disciplina Projecto, Seminário ou Trabalho de Final de Curso e foi sugerido pela empresa *Mactek*.

Assim, o projecto pretendia que se construísse uma aplicação de transmissão de voz e vídeo sobre IP, de forma a ser possível transmitir numa rede sem fios (WLAN). Os requisitos deste projecto eram pouco detalhados, por isso, foi necessário a determinação destes requisitos tendo em conta que se pretendia a melhor solução possível. Para isso, foi necessário pesquisar e estudar protocolos existentes e tecnologias de compressão de dados em tempo real.

Utilizando o *Microsoft Visual Studio.NET 2003*, foi desenvolvida a aplicação requisitada, que permite transmitir e receber áudio e vídeo em tempo real. Estes dados são comprimidos, podendo-se controlar aspectos como a qualidade de áudio e do vídeo. O controlo das chamadas também foi implementado de forma a ser possível realizar chamadas em conferência e pôr chamadas em espera. Pode-se também fazer transferências de ficheiros

A aplicação contém um livro de endereços para os utilizadores poderem organizar os seus contactos e regista as chamadas efectuadas e recebidas. Por fim, foi implementada

a reprodução de ficheiros de som durante uma chamada para que os utilizadores possam ouvir.

Com este relatório pretende-se mostrar os protocolos e tecnologias envolvidas nesta aplicação, com uma descrição das bibliotecas utilizadas e apresentação da arquitectura e da aplicação desenvolvida.

Este relatório está dividido em 7 capítulos:

- O Capítulo 1 é este que e representa a Introdução;
- O Capítulo 2 descreve os requisitos gerais determinados para a realização da aplicação;
- O Capítulo 3 descreve como é feita a captura e transmissão dos dados, subdividindo em captura (Secção 3.1), compressão (Secção 3.1.1) e reprodução (Secção 3.2) de áudio, captura (Secção 3.3) e compressão (Secção 3.3.1) de vídeo e protocolo de transporte em tempo real (Secção 3.4). Há também uma introdução à biblioteca utilizada (Secções 3.5 e 3.6);
- O Capítulo 4 descreve como é feita a sinalização, fazendo uma introdução aos protocolos SIP (Secção 4.1) e SDP (Secção 4.2). Este capítulo introduz também a biblioteca utilizada nas Secção 4.1.2 e 4.1.3;
- O Capítulo 5 descreve como foi implementado o livro de endereços;
- O Capítulo 6 mostra a arquitectura da aplicação desenvolvida (Secção 6.1) e um pequeno manual de utilização (Secção 6.3);
- Finalmente, o Capítulo 7 apresenta as conclusões e sugestões para a possível prossecução do trabalho.

# 2. Requisitos Gerais

De acordo com a descrição do projecto foi necessário determinar os requisitos tendo em conta o tipo de aplicação que se queria desenvolver.

Pretendia-se uma aplicação de transmissão de áudio e vídeo em tempo real e para isso uma parte vital óbvia e crítica seria a captura dos dados a transmitir, nomeadamente o áudio e vídeo. A partir desta parte vital, foi necessário proceder-se a uma análise de requisitos, levando à especificação de requisitos mais detalhados.

Assim, como parâmetros associados à captura, definiu-se para o vídeo uma taxa de refrescamento (*frame rate*) de 15 frames por segundo (fps), visto ser uma taxa *standard* suportada pelas *webcams* convencionais.

Foi definido como 10 o número máximo de utilizadores numa conferência, uma vez que o tempo de processamento aumenta com o número de utilizadores e a largura de banda é limitada e reduzida.

Foi necessário também especificar uma *bitrate* por omissão para cada utilizador numa chamada. Tendo em consideração que grande parte dos utilizadores de Internet de banda larga tem uma velocidade máxima de *upload* de 128 kbps, definiu-se este valor como o valor máximo a atingir. Para isso, como seria de esperar, foram utilizados *codecs* (compressor/descompressor) que permitiam limitar as *bitrates*:

- Como *codec* de áudio foi utilizado o *Speex* (Secção 3.1.1) que tem uma *bitrate* a variar entre 2 e 44 kbps;

- Para o vídeo utilizou-se a compressão H.263+ que permite definir uma *bitrate* fixa.

Logo, admitindo que o áudio transmite com um *bitrate* máximo de 44 kbps, foi definido para o vídeo 84 kbps.

Os valores anteriores são atribuídos por omissão, mas foi definido que os utilizadores da aplicação poderia modificá-los, de forma a obter um maior controlo.

Para implementar a parte de captura e transmissão de áudio e vídeo, escolheu-se uma biblioteca auxiliar chamada EMIPLIB (Secção 3.5). Esta biblioteca caracteriza-se por simplificar bastante a captura, compressão e transmissão de áudio e vídeo

Outra parte de extrema importância para o projecto seria a implementação da sinalização de chamadas de forma a poder iniciar, terminar e modificar sessões com outros utilizadores.

O protocolo SIP (Secção 4.1) foi o escolhido para fazer a sinalização. Este protocolo serve-se de outros protocolos existentes para controlar as sessões, nomeadamente do SDP (Secção 4.2) e o RTP (Secção 3.4). O SDP serve para negociar o tipo de média e

*codecs* a utilizar, enquanto que o RTP serve para transmitir dados em tempo real, tais como áudio e vídeo.

Para implementar a sinalização, foi escolhida a biblioteca *eXoSIP* (Secção 4.1.2), uma biblioteca que foi especificamente construída para a sinalização de chamadas de Voz sobre IP (*VoIP*).

Passando para os requisitos de mais baixa prioridade, pensou-se que seria interessante manter um registo das chamadas efectuadas e recebidas, para que os utilizadores pudessem controlar facilmente as suas chamadas.

Um livro de endereços também seria interessante para manter os contactos organizados.

Os utilizadores muitas vezes sentem a necessidade de trocar ficheiros entre si, por isso, também se sugeriu como requisito.

Decidiu-se também implementar o *streaming* de ficheiros de som, visto que a biblioteca EMIPLIB permitia fazer isso e traria à aplicação uma funcionalidade útil que acrescenta valor.

Já que se escolheu bibliotecas que funcionam em diversos sistemas operativos, foi definido como um requisito possível a maximização da portabilidade para outros sistemas operativos.

Mostra-se de seguida os diagramas de caso de uso para a aplicação (Figura 1 e Figura 2):

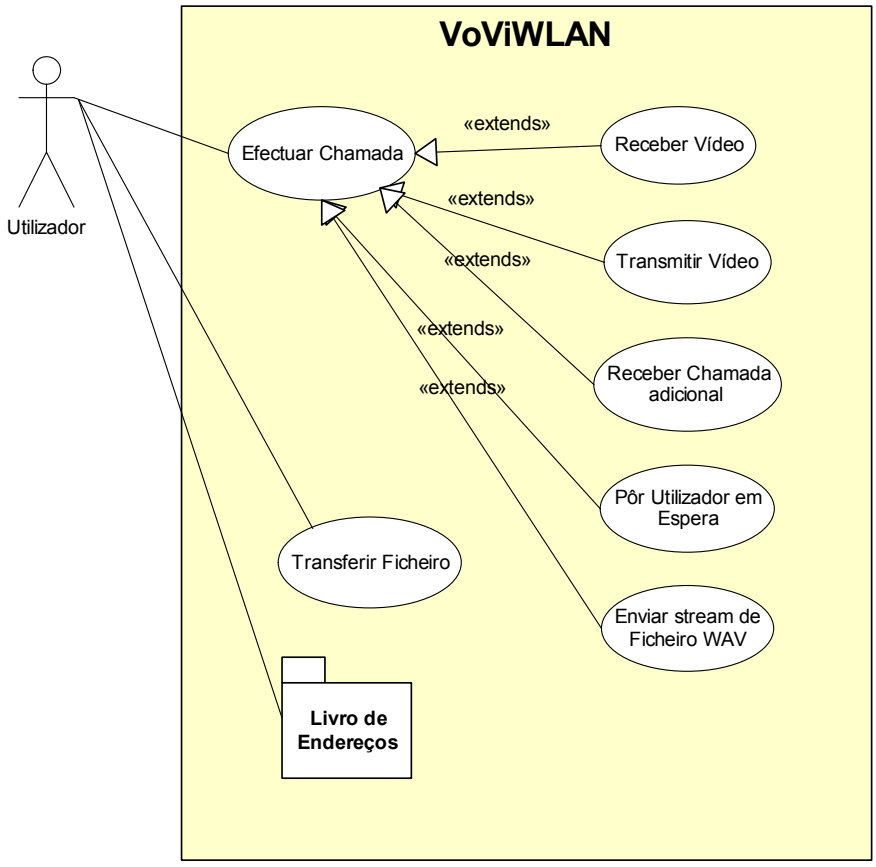


**Figura 1 –** Diagrama de casos de uso da aplicação

**Tabela 1 –** Descrição dos casos de utilização do sistema principal

| Casos de Utilização | Descrição |
|---|---|
| Efectuar Chamada | O utilizador quando deseja fazer uma chamada deve colocar o endereço no local certo ou seleccionar de uma mini-agenda e chamar esse utilizador.<br>A chamada é terminada quando só estiverem dois utilizadores e um deles a terminar, ou quando o utilizador receptor rejeita a chamada. |
| Receber Vídeo | Os utilizadores podem ou não receber vídeo. Podem receber vídeo automaticamente, ou decidem no momento que o emissor tenta enviar. |
| Transmitir Vídeo | No caso do utilizador possuir uma *webcam* pode transmitir vídeo. Existem duas maneiras de o começar a transmitir: logo que começa uma chamada, sendo para isso necessário activar a opção para mandar automaticamente, e durante uma chamada de áudio. |
| Receber Chamada Adicional | Durante uma chamada, pode-se receber uma chamada adicional. Para isso, o utilizador recebe um pedido para aceitar ou rejeitar a chamada. Se aceitar a nova chamada junta-se à antiga e é iniciada uma conferência. |
| Pôr Utilizador em Espera | Durante uma chamada, pode ser necessário colocar um utilizador em espera para, por exemplo, iniciar uma nova chamada, etc. Pôr uma chamada em espera significa deixar de mandar e receber dados de som (e vídeo se activo). |
| Enviar *stream* de Ficheiro WAV | Durante uma chamada, pode-se tocar um ficheiro para o(s) outro(s) utilizadores ouvirem. |
| Transferir Ficheiro | Ao seleccionar um utilizador, pode-se transferir um ficheiro seleccionado. |
| Livro de Endereços | Este módulo inclui os casos de uso para o livro de endereços. |

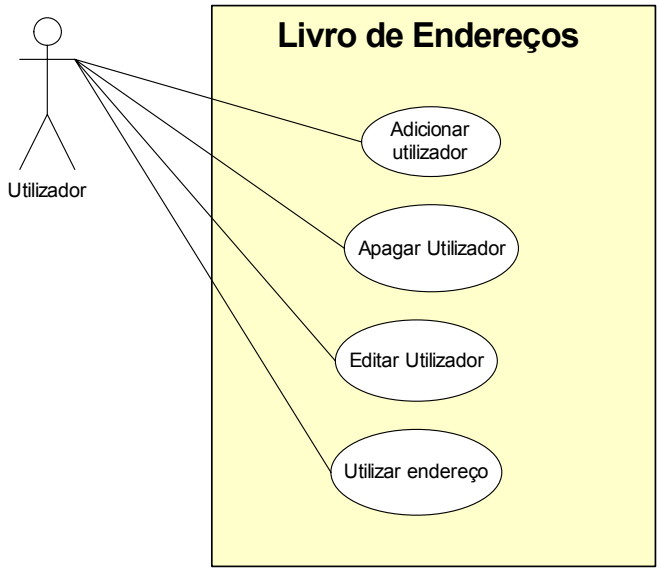


**Figura 2 –** Diagrama de casos de uso para o pacote Livro de Endereços

**Tabela 2 –** Descrição dos casos de uso do Livro de Endereços

| Casos de Utilização | Descrição |
|---|---|
| Adicionar Utilizador | Adiciona um utilizador ao livro de endereços. |
| Apagar Utilizador | Apaga o utilizador seleccionado do livro de endereços. |
| Editar Utilizador | Pode ser necessário mudar o endereço dos utilizadores. |
| Utilizar Endereço | Permite a aplicação utilizar o endereço para iniciar uma chamada. |

Em conclusão, foi definido que se faria a captura de áudio (Secção 3.1) e vídeo (Secção 3.2) comprimindo-os, respectivamente, com *Speex* (Secção 3.1.1) e H.263+ (Secção 3.3.1), bem como mandar estes dados, usando o protocolo de transporte RTP (Secção 3.4). Para implementar isto, utilizar-se-ia a biblioteca auxiliar *EMIPLIB* (Secção 3.5). Foi escolhida a sinalização SIP (Secção 4.1) com a utilização do protocolo SDP (Secção 4.2) para negociar os tipos de media, fazendo uso da biblioteca *eXoSIP* (Secção 4.1.2). Por fim, foram definidos possíveis incrementos à aplicação, tais como um livro de endereços (Capítulo 5), registo de chamadas, *streaming* de ficheiros de som, portabilidade, etc.

# 3. Captura de Dados e Transmissão

Para transmitir áudio e vídeo torna-se, obviamente, necessário fazer a sua captura. Para isso, utiliza-se os recursos do PC. O áudio é capturado pela placa de som, que é comum em quase todos os PCs, e o vídeo é capturado por uma *webcam* que esteja ligada.

Nas secções seguintes vamos falar da captura do áudio (Secção 3.1) e sua compressão (Secção 3.1.1), reprodução de áudio (Secção 3.2) e captura (Secção 3.3) e compressão (Secção 3.3.1) de vídeo.

Na secção 3.4 vamos fazer uma introdução ao protocolo de transporte RTP revelando os aspectos mais importantes.

Para terminar, as secções 3.5 e 3.6 descrevem alguns detalhes sobre biblioteca *EMIPLIB* e como a utilizar.

## *3.1. Captura de Áudio*

A biblioteca *EMIPLIB* implementa a captura de áudio utilizando o Interface de Programação de Aplicativos (API) *Microsoft® Win32®* que permite adicionar a uma aplicação funções multimédia através do *winMM*.

A captura de áudio é feita de uma forma muito simples:

- Primeiro abre-se o dispositivo de áudio desejado através da função `waveInOpen`;
- Depois prepara-se os *buffers* para o áudio através da função `waveInPrepareHeader`;
- Manda-se os *buffers* para o dispositivo de áudio através da função `waveInAddBuffer`;
- Finalmente, inicia-se a captura com a função `waveInStart`.

Para terminar a captura usa-se a função `waveInStop`.

### 3.1.1. Compressão do Áudio

Para transmitir o áudio numa rede é necessário comprimi-lo de forma a não ocupar uma grande largura de banda.

Foi utilizado um *codec* com o nome *Speex*. Trata-se de um *codec Open Source* feito para comprimir áudio de voz. Fazendo uma tradução livre do seu site (2) "é bem adaptado para aplicações de Internet e fornece características úteis que não estão presentes noutros *codecs*".

Este *codec* foi feito para comprimir voz com *bitrates* a variar entre 2 e 44 kbps. Dá a possibilidade de comprimir em banda estreita (*Narrowband*) (8kHz), banda larga (*Wideband*) (16kHz) e banda ultra-larga (*Ultra-wideband*) (32kHz), tem detecção de voz (*Voice Activity Detection -VAD*), transmissão descontínua (*DTX*), etc.

## 3.2. Reprodução de Áudio

Tal como a captura do áudio, a biblioteca *EMIPLIB* utiliza o Interface de Programação de Aplicativos (API) *Microsoft® Win32®*, nomeadamente o *winMM*.

A reprodução de áudio é semelhante à captura, embora seja um pouco mais complexo pois é necessário receber o áudio e colocá-lo nos *buffers* para ele ser reproduzido:

- O dispositivo de saída de áudio é inicializado pela função `waveOutOpen`;
- É construído um buffer com a estrutura `WAVEHDR`;
- Quando se recebe um pacote RTP de áudio, o conteúdo é colocado no *buffer*;
- Prepara-se os *buffers* para reprodução;
- Finalmente, utiliza-se a função `waveOutWrite` para o dispositivo de som reproduzir os dados que foram recebidos.

## 3.3. Captura de Vídeo

A *EMIPLIB* já implementa a captura de vídeo, usando, para isso, no sistema operativo *Microsoft Windows*, um Interface de Programação de Aplicativos (API) chamado *DirectShow*. Este API é o largamente mais utilizado pelos programadores no *Windows*.

Foi criado pela *Microsoft* para os programadores fazerem operações com ficheiros de média. Para além disso, permite a captura a partir de dispositivos Multimédia, como *Webcams*.

Um componente básico do *DirectShow* é os filtros (*filters*), que correspondem a um componente de *software* que faz algum tipo de operação numa trama (*stream*) multimédia. Por exemplo, podem ler ficheiros, descodificar formatos, capturar vídeo, etc. Os filtros têm entradas e saídas, constituindo pinos que podem ser ligados a outros pinos de outros filtros, que formam, assim, gráficos de filtros (*Filter Graphs*).

O diagrama seguinte mostra um gráfico de filtros para tocar um ficheiro de vídeo AVI:

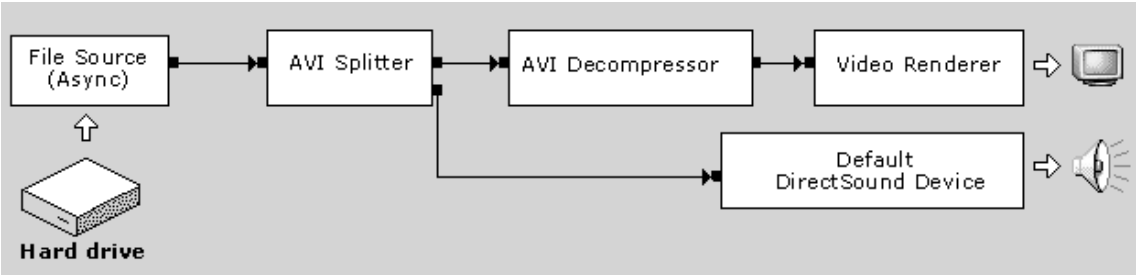


**Figura 3 –** Exemplo de um gráfico de filtros para tocar um ficheiro AVI (3)

O *File Source* lê o ficheiro AVI do disco rígido, depois separa o áudio de vídeo no *AVI Splitter*. O *AVI Decompressor* descomprime o vídeo e o *Video Renderer* desenha as *frames* no ecrã. O *Default DirectSound Device* toca o áudio no dispositivo de som.

Todas as aplicações que usam o *DirectShow* devem criar um *Filter Graph Manager* e utilizá-lo para criar e controlar filtros de gráficos.

Os gráficos de filtros que capturam dados chamam-se gráficos de captura (*capture graphs*). O *DirectShow* disponibiliza um construtor de gráficos de captura, que facilita a criação deste tipo de gráficos.

Assim, para capturar vídeo é necessário criar um gráfico do género da Figura 4:

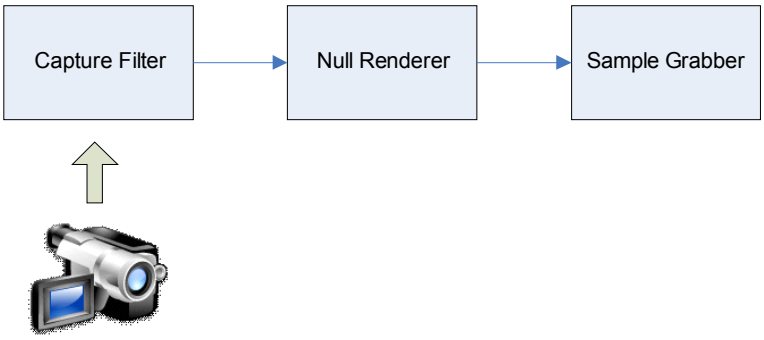


**Figura 4 –** Gráfico de filtro para capturar vídeo

O *Capture Filter* captura dados do dispositivo de vídeo (*Webcam*), passa no *Null Renderer* e, finalmente, fica no *Sample Grabber*, que fornece as amostras à aplicação, para depois serem comprimidas.

### 3.3.1. Compressão de Vídeo

Tal como o áudio, há a necessidade de comprimir o vídeo, de forma acrescida, visto que é preciso uma maior de quantidade de dados para representar uma imagem.

Para fazer a compressão utilizou-se o *codec H.263+*. Trata-se de um *codec* feito para comunicações com baixo débito, próprio para a Internet.

Este *codec* utiliza um algoritmo híbrido de previsão entre cada imagem, *Tranform coding* e compensação do movimento:

- A previsão entre cada imagem remove a redundância temporal entre pixeis;
- *Transform coding* remove redundância espacial;
- Vectores de movimento ajudam a compensar o movimento. Estes vectores têm a precisão de meio pixel.

## 3.4. Transmissão RTP

O protocolo RTP é documentado pelo RFC 3550 (4). Este protocolo fornece transporte entre dois pontos para transporte em tempo real de dados como áudio e vídeo.

O RTP não garante qualidade de serviço. Este protocolo não tem mecanismos que asseguram a entrega de pacotes, nem a entrega de pacotes na ordem correcta, mas também não assume que a rede entrega-os na sequência certa: os números de sequência ajudam ao receptor reconstruir a ordem certa de pacotes.

Os pacotes de áudio e vídeo são transmitidos em separado, em diferentes portas. Isto serve para que o utilizador possa escolher receber apenas um tipo de média.

Para perceber como é um pacote RTP é necessário introduzir alguns conceitos:

- Pacote RTP – Um pacote de dados, constituído por um cabeçalho RTP e os dados que o pacote transporta;
- *RTP payload* – Os dados transportados pelo pacote RTP. Existem vários formatos de *payload*, como o PCMU, PCMA, etc.

De seguida mostra-se como é o formato genérico do cabeçalho RTP:

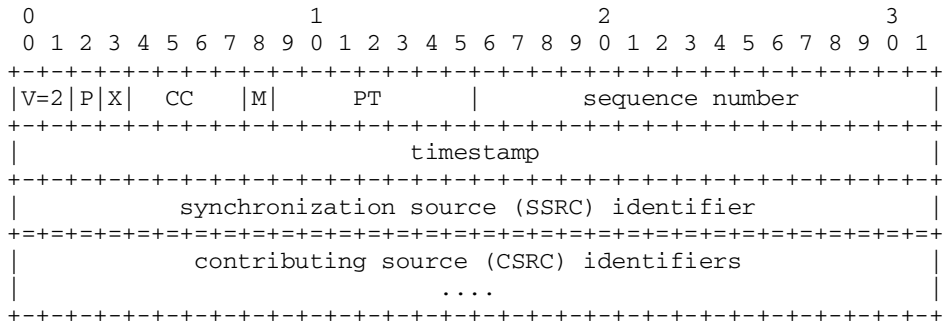


**Figura 5 –** Cabeçalho de um pacote RTP

Os campos têm o seguinte significado:

- *Version (V)* – versão – Indica a versão do protocolo;
- *Padding (P)* – Se bit estiver a 1 indica que o pacote tem um ou mais octetos que não pertencem ao *payload*. O número de octetos a mais está indicado no último octeto;
- *Extension (X)* – Extensão – Indica que há uma extensão no header;
- *CSRC count (CC)* – Indica o número de identificadores CSRC que estão a seguir ao cabeçalho;
- *Payload Type (PT)* – Tipo de *payload* – Indica o formato do payload (PCMU, GSM, etc.);

- *Sequence number* – Número de sequência – Este número é incrementado por cada pacote RTP que é mandado. É normalmente utilizado pelo receptor para ordenar os pacotes recebidos;

- *Timestamp* – Indica o instante de amostragem do primeiro octeto do pacote RTP. Este campo serve para sincronização e cálculo do *jitter*. A frequência do relógio para a geração do *timestamp* depende do formato que se está a transmitir;

- *SSRC* – Identifica a fonte de sincronização. Este valor deve ser escolhido aleatoriamente de forma que as duas fontes de sincronização da mesma sessão de RTP tenham o mesmo identificador SSRC;

- *CSRC* – Identifica a fontes que contribuem para o *payload*.

Mostra-se de seguida, um extracto de um pacote RTP que transmite áudio com o formato GSM:

```
Real-Time Transport Protocol
    Stream setup by SDP (frame 10)
        Setup frame: 10
        Setup Method: SDP
    10.. .... = Version: RFC 3550 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count: 0
    0... .... = Marker: False
    Payload type: GSM 06.10 (3)
    Sequence number: 44497
    Timestamp: 958
    Synchronization Source identifier: 3801817858
    Payload: D9A28B19AB6D6036D44DB6D2BDE038D375B71D6D200B1C91...
```

**Figura 6 –** Pacote RTP

## 3.5. Introdução ao EMIPLIB

Foi utilizada uma biblioteca que implementa o que é descrito nas secções anteriores. Esta biblioteca chama-se *EMIPLIB* e tem a grande vantagem de simplificar a captura e transmissão dos dados. Esta biblioteca pode ser utilizada em *Windows*, *Windows CE*, *Linux* e *Mac OS X* tornando uma aplicação facilmente portável para outro sistema operativo.

Esta biblioteca permite:

- Capturar som (através de *OSS*, *WinMM* ou *Jack*);

- Tocar som (através de *OSS*, *ALSA*, *ESD*, *WinMM* e *Jack*);

- Entrada de ficheiros WAV (através de *libsndfile, libaudiofile* ou de um leitor de WAV interno);

- Saída de ficheiros WAV (através de *libsndfile*);

- Captura da *webcam* (através de *Video4Linux* e *DirectShow*);

- Compressão *Speex*;
- Compressão *H.263+* (através de *libavcodec*);
- Mistura de áudio recebido;
- Envio de tramas RTP;
- Sincronização de tramas RTP;
- Efeitos de som 3D.

O EMIPLIB está organizado em pequenos componentes que se ligam em cadeia (Figura 7). Por exemplo, um componente para capturar áudio, outro para comprimir e outro para transmitir. Ao activar uma cadeia as mensagens são distribuídas entre componentes.

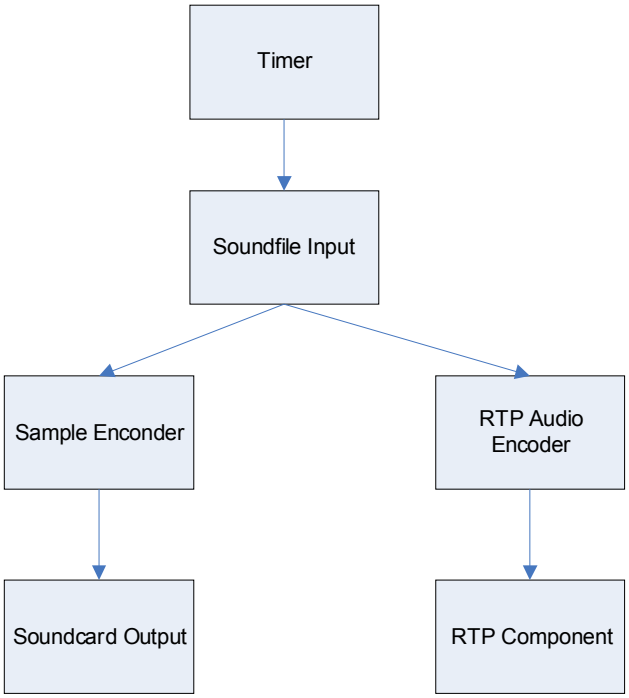


**Figura 7 –** Exemplo de uma cadeia de componentes

### 3.5.1. Código Exemplo

No conjunto de código seguinte, mostra-se como criar uma simples aplicação, cuja finalidade única é reproduzir o áudio capturado pelo microfone.

Como as cadeias usam *threads,* é necessário derivar a classe `MIPComponentChain`, para o caso de ocorrer um erro na *thread*.

```
class MyChain : public MIPComponentChain
 {
 public:
        MyChain(const std::string &chainName) :
         MIPComponentChain(chainName)
        {
        }
 private:
        void onThreadExit(bool error, const std::string
         &errorComponent, const std::string &errorDescription)
        {
                if (!error)
                        return;

                std::cerr << "An error occured in the background
         thread." << std::endl;
                std::cerr << "    Component: " << errorComponent <<
         std::endl;
                std::cerr << "    Error description: " <<
         errorDescription << std::endl;
        }
 };
```

**Código 1 –** Exemplo de uma classe que representa cadeias (*Chains)*

Depois, deve-se inicializar os componentes:

```
bool returnValue;
int samplingRate = 16000;
int numChannels = 1;

MIPWinMMInput * sndCardInput = new MIPWinMMInput();
returnValue=sndCardInput->open(samplingRate, numChannels,
        interval);
checkError(returnValue, sndCardInput);

MIPWinMMInput * sndCardOutput = new MIPWinMMOutput();
returnValue = sndCardOutput->open(samplingRate, numChannels,
        interval);
checkError(returnValue, sndCardOutput);
MyChain * chain("Sound file player");
```

**Código 2 –** Exemplo da inicialização dos componentes

A função `checkError` é uma função que deve ser criada para verificar se ocorreram erros, e em caso afirmativo, imprime a mensagem de erro retornada.

Agora, é necessário criar as cadeias (*Chains)*, ligar os componentes e começar a trocar mensagens entre componentes.

```
MyChain * chain("Sound file player");
returnValue = chain->setChainStart(sndCardInput);
checkError(returnValue, chain);

returnValue = chain->addConnection(sndCardInput, sndCardOutput);
checkError(returnValue, chain);

returnValue = chain->start();
checkError(returnValue, chain);
```

**Código 3 –** Exemplo da ligação dos componentes numa cadeia (*Chain)*

A Figura 8 mostra como são ligados os componentes em cadeia:

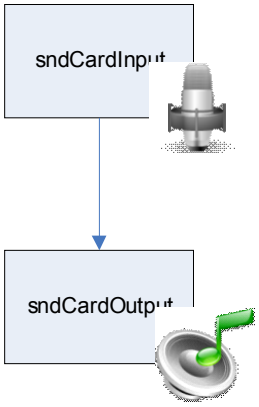


**Figura 8 –** Cadeia de componentes

### 3.5.2. Dificuldades na utilização da biblioteca

Esta biblioteca ainda se encontra numa versão inicial (0.11), o que dificultou a sua utilização, uma vez que depois de algumas experiências foram encontrados alguns erros, os quais foram corrigidos.

Primeiro, a biblioteca disponibilizava uma classe que implementava cadeias para transmitir e receber áudio, o que facilitava imenso a utilização. Esta classe não funcionava devido à componente de RTP. Por este motivo, foi decidido criar outra classe manualmente. O mesmo aconteceu com outra classe, desta vez, para a transmissão e recepção de vídeo.

Depois, a classe que capturava vídeo através do *DirectShow* não conseguia detectar a *webcam*, visto que estava a tentar encontrar uma câmara que captasse com o formato IYUV. Este formato não era suportado pela *webcam* que se estava a utilizar. No entanto, ela capturava num formato equivalente, mais comum, o I420. Estes dois formatos têm um modo de representação da cor YUV, ou seja, têm uma luminância e duas crominâncias. Este modelo é utilizado nos sistemas de televisão PAL e NTSC. Y representa a luminância e U e V a crominância. Estes componentes são criados a partir dos valores vermelho (R), Verde (G) e Azul (B). Para corrigir este problema foi criada uma cópia desta classe e substituído o formato.

## 3.6. Obtenção e detalhes sobre o EMIPLIB

Neste projecto, foi utilizada a versão 0.11 do EMIPLIB descarregada na página do projecto em (5).

Para compilar em Windows, esta biblioteca apenas disponibilizava um ficheiro de solução para o *Microsoft Visual Studio 2005*. Como se estava a utilizar o *Microsoft Visual Studio .NET 2003* foi necessário criar uma nova solução com os mesmos ficheiros e propriedades do projecto par a versão de 2005.

Antes de se compilar, deve-se descomentar no ficheiro `mipconfig_win.h` a linha que diz `#define MIPCONFIG_SUPPORT_QT`.

Para utilizar esta biblioteca era necessário compilar outras bibliotecas que o *EMIPLIB* dependia, tais como o *JRTPLIB*, o *JThread*, *Speex*, *ffmpeg* e Qt. As secções seguintes mostram como se obtém e como compilar estas bibliotecas e secção.

### 3.6.1. JRTPLIB

Esta biblioteca oferece suporte ao protocolo de transporte RTP (Secção 3.4) para *Windows* e *Linux*. Foi utilizada a versão 3.5.0 disponível em (6).

Eram disponibilizados ficheiros de solução para o *Microsoft Visual Studio 6* e para o *Microsoft Visual Studio 2005*, versão 8, mas não disponibilizava para a versão que estava a ser utilizada, o *Mircrosoft Visual Studio .NET 2003*, versão 7.1. Logo, converteu-se o projecto da versão 6 para a 7.1, abrindo este ficheiro com a versão do *Visual Studio* que se estava a utilizar.

Finalmente, para se compilar, utilizou-se o mesmo método que nos projectos do *Visual Studio*.

### 3.6.2. JThread

Esta biblioteca disponibiliza classes para a criação de *Threads* e *Mutexes*, escondendo a forma como se criam threads em *Windows* ou *Linux*. Foi utilizada a versão 1.2.0 disponível em (7).

Tal como na JRTPLIB, eram também disponibilizados ficheiros de solução para o *Microsoft Visual Studio 6* e para o *Microsoft Visual Studio 2005*, versão 8, não disponibilizando para a versão que estava a ser utilizada, o *Mircrosoft Visual Studio .NET 2003*, versão 7.1. Logo, converteu-se o projecto da versão 6 para a 7.1 abrindo este ficheiro com a versão do *Visual Studio* que se estava a utilizar.

Na compilação, utilizou-se o mesmo processo que nos projectos do *Visual Studio*.

### 3.6.3. Speex

O *Speex* (Secção 3.1.1) corresponde a um *codec* de áudio necessário para compilar o *EMIPLIB*. Foi utilizada a versão 1.0.5 disponível para descarregar em (2).

Só era disponibilizado o ficheiro de solução para o *Microsoft Visual Studio 6*, por isso, ao abrir este ficheiro com o *Microsoft Visual Studio .NET 2003*, procedeu-se à sua conversão.

Para se compilar, fez-se da mesma forma como se compila os projectos no Visual Studio.

### 3.6.4. ffmpeg

Ffmpeg é uma solução para gravar converter áudio e vídeo. O EMIPLIB apenas utiliza a *libavformat*, *libavcodec* e *libavutil* pertencentes a esta solução para comprimir o vídeo em H.263+.

Na página desta solução está indicado que deve ser usada sempre a versão mais recente disponível pelo sistema de controlo de versões SVN. Logo, foi descarregada a versão de 5 de Abril de 2006 às 15 horas 32 minutos e 54 segundos através do comando:

```
svn checkout svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg
```

Para compilar o *ffmpeg* é necessário ter instalado o *MinGW* e o *MSYS* que, basicamente, correspondem a uma *shell* mínima e a um sistema mínimo que simula o *Linux*.

Na *shell* deve-se introduzir `./configure --enable-memalign-hack –disable-static –enable-shared` e quando terminar fazer `make`.

O EMIPLIB utiliza os ficheiros `avcodec.lib`, `avformat.lib` e `avutil.lib`.

### 3.6.5. Qt

*Qt* é um sistema de interfaces gráficas multiplataforma. O EMIPLIB usa o *Qt* para apresentar uma janela com o vídeo recebido. Como o EMIPLIB não suporta a versão 4 do *Qt*, foi usada a versão 3.3.5 disponível em (8).

Para compilar o *Qt* deve-se seguir a extensa lista de passos descritos no ficheiro `INSTALL` excepto no passo onde diz `./configure`, pois deve-se substituir isto com `./configure –platform win-msvc.net –shared –debug`.

### 3.6.6. Dependências do EMIPLIB

Quando se utiliza esta biblioteca deve-se utilizar no *linker* os seguintes ficheiros:

- Quartz.lib;
- Strmiids.lib;
- wsock32.lib;
- ws2_32.lib;
- Iphlpapi.lib;
- eXosip.lib;

- osip2.lib;
- osipparser2.lib;
- jthread.lib;
- jrtplib.lib;
- libspeex.lib;
- avformat.lib;
- avcodec.lib;
- avutil.lib;
- qt-mt3.lib;
- O ficheiro gerado pela compilação do *EMIPLIB*
- ftol2.obj;
- winmm.lib.

# 4. Sinalização

Para estabelecer uma chamada com um utilizador, é necessário haver sinalização entre as chamadas, para que os utilizadores possam aceitar, rejeitar, ou efectuar pedidos de chamadas.

Logo, foi necessário escolher um protocolo de sinalização, tendo como possibilidades a criação de um protocolo próprio, a utilização do protocolo H323, ou o SIP.

Criar um protocolo próprio estava fora de questão, uma vez que era previsível que um protocolo criado por uma só pessoa podia não ser suficientemente estável ou ter muitas falhas que dificultassem a realização do trabalho. Também, dificilmente seria um protocolo compatível com outras aplicações existentes. Outra razão importante era o tempo disponível para realizar o projecto, que não era suficiente, se fosse necessário criar este protocolo.

O H323 é um protocolo criado pelo ITU e é considerado um protocolo pouco flexível, pois especifica um sistema total e único, ou seja um conjunto de protocolos fixos e obrigatórios, para efectuar as suas funções. Tem uma codificação binária que dificulta a detecção de erros durante a programação. Foi criado a pensar em LANs.

Então, optou-se pelo SIP por ser mais flexível, escalável e, acima de tudo, simples. A seguir faz-se uma descrição do protocolo. O SIP permite várias funções, como a transferência de chamadas, a possibilidade de se poder ter vários telefones com o mesmo endereço, a possibilidade de se pôr utilizadores em espera, etc.

## *4.1. SIP*

O SIP é um protocolo proposto como *standard* para iniciar, modificar e terminar sessões criado pelo IETF e descrito no RFC 3261 (9).

Uma sessão normalmente está associada a uma chamada, ou seja, corresponde à ligação que se mantém entre utilizadores.

Este protocolo foi feito a pensar na Internet de forma a utilizar outros já existentes, visto que não é um protocolo de descrição de sessões e não faz controlo de conferências. Para isso é utilizado normalmente o SDP (Secção 4.2). Isto torna este protocolo muito flexível e escalável.

Normalmente estão associados ao SIP outros protocolos como o RTP (Secção 3.4) quando se quer transmitir dados em tempo real e o SDP (Secção 4.2), que negoceia os detalhes da sessão como, por exemplo, os tipos de media a transmitir, os *codecs*, endereços, portas, etc.

Existem algumas extensões que permitem expandir as funcionalidades do SIP, como por exemplo:

- SIMPLE – permite o suporte de mensagens instantâneas e presença (notificação de utilizadores se estão ligados ou desligados, por exemplo), definindo uma arquitectura para a implementação de *buddy lists*, mecanismos de confirmação de entrega de mensagens, etc;

- XCON – permite suportar conferências centralizadas.

Uma das características deste protocolo é a sua simplicidade, pois é baseado em texto, similar ao HTTP e SMTP, facilitando a extensão, a depuração e o processamento.

De seguida, aborda-se os aspectos mais técnicos deste protocolo que são descritos no seu RFC (9).

### 4.1.1. Arquitectura

O SIP é parecido com o HTTP, pois tem um modelo de transacção de requisição e resposta. Uma requisição representa um método ou uma função e tem sempre uma ou mais respostas.

A Figura 9 mostra um exemplo de uma sessão estabelecida entre dois utilizadores, sem utilização de qualquer servidor intermédio, mostrando as mensagens mais utilizadas neste protocolo.

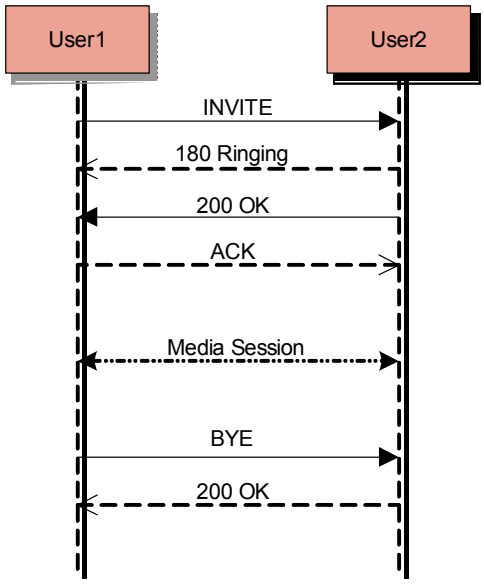


**Figura 9 -** Exemplo das mensagens SIP para o estabelecimento de uma sessão

Primeiro, o User1 manda um INVITE para o User2 de forma a iniciar uma sessão. O User2 pode responder a esta mensagem de várias formas. Neste caso está a responder com um *Ringing* e com um OK. A mensagem *Ringing* é considerada uma resposta provisória e avisa o User1 que o User2 foi alertado e que está à espera que este aceite ou rejeite a chamada. O OK é uma mensagem definitiva e quer dizer que o User2 aceitou a chamada. Estas mensagens podiam ser do tipo das que não deixam estabelecer a sessão (Secção 4.1.1.3), tais como Falha de Requisição (*Request Failure*) ou Falhas Globais (*Global Failure*).

Antes de continuar, é necessário introduzir o conceito de SIP URI. Representa um endereço global de um utilizador e contém informação suficiente para iniciar e manter a uma sessão. Normalmente tem um formato do género:

```
sip:user@host
```

Onde *user* representa um cliente do *host* e o *host* representa o endereço do prestador de serviços que, normalmente, sabe para onde mandar os pedidos e respostas. Quando um cliente não está associado a um prestador de serviços, o *host* corresponde a um endereço IP.

```
INVITE sip:user2@192.168.0.2 SIP/2.0
Via: SIP/2.0/UDP 192.168.0.1:5060;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: User2 <sip:user1@192.168.0.2>
From: User1 <sip:user1@192.168.0.1>;tag=1928301774
Call-ID: a84b4c76e66710@192.168.0.1
CSeq: 314159 INVITE
Contact: <sip:user1@192.168.0.1>
Content-Type: application/sdp
Content-Length: 142

(SDP do User1 não mostrado)
```

**Figura 10 –** Exemplo de um cabeçalho de um pacote SIP

Acima mostra-se a típica mensagem SIP, onde a primeira linha indica o nome do método (INVITE) e as linhas seguintes mostram os cabeçalhos obrigatórios em todas as mensagens SIP, destacando:

- *Via* – Representa o endereço onde o User1 está à espera para receber a(s) resposta(s) a esta requisição;
- *To* – Contém o nome e endereço SIP (URI) do destinatário da mensagem;
- *From* - Contém o nome e endereço SIP (URI) do remetente da mensagem. Contém também um campo *tag* usado para identificação;
- *Call-ID* – É um identificador único para esta chamada (sessão). É gerado combinando uma *string* aleatória com o endereço IP do utilizador ou o seu *host name*;
- *CSeq* – Contém um número inteiro e o nome do método. O número é incrementado sempre que é gerada uma requisição dentro de um diálogo;
- *Contact* – É constituído por um URI que representa a rota directa para o contacto User1;
- *Content-Type* – Descreve o tipo do corpo da mensagem.

Porque o SIP não descreve os detalhes da sessão, como o tipo de média, *codecs*, etc. é então colocado no seu conteúdo a mensagem SDP (Secção 4.2), que é considerado um anexo, assimilando-se aos anexos transportados numa mensagem de correio electrónico.

#### 4.1.1.1. UAC e UAS

Até agora tem-se falado apenas de utilizadores, mas agora torna-se necessário dividir esta definição em cliente (UAC) e servidor (UAS). Basicamente, o UAC é a parte que gera os pedidos (requisições) e recebe respostas, enquanto que o UAS, ao receber estes pedidos, gera as respostas de acordo com os dados que dispõe.

#### 4.1.1.2. Diálogos

Um diálogo corresponde a uma relação entre dois UAs que permanece durante algum tempo.

É criado quando um pedido recebe uma resposta com códigos ente 100 e 299, ou seja, respostas provisórias ou respostas que aceitam o pedido.

#### 4.1.1.3. Tipos de resposta

Uma requisição pode ter vários tipos de respostas, sendo elas provisórias ou definitivas. De seguida, mostra-se os tipos de respostas que se pode receber juntamente com exemplos das mais utilizadas:

- 1xx – É uma resposta provisória que indica que o UAS recebeu o pedido, mas continua a processá-lo. O UAC pode receber zero ou mais respostas deste tipo;
    - 100 Trying – Indica que o pedido foi recebido e que estão a ser tomadas acções para esta chamada ter uma resposta definitiva;
    - 180 Ringing – Mostra que o UA que recebeu o INVITE está a tentar alertar o utilizador.
- *2xx* – Resposta definitiva e indica que o pedido foi recebido, percebido e aceite;
    - *200 OK* – Foi aceite o pedido e normalmente transporta em anexo mais detalhes.
- *3xx* – Indica que o utilizador tem outra localização, indicando o(s) novo(s) endereço(s). Cabe ao UAC decidir se deve voltar a contactar e escolher outro endereço;
- *4xx* – Resposta definitiva e diz ao UAC que o pedido tem erros na sintaxe ou não pode ser aceite no servidor. O UAC não deve tentar o mesmo pedido sem o modificar.
    - *400 Bad Request* – O pedido não foi percebido devido à sua sintaxe;
    - *415 Unsupported Media Type* – Indica que os formatos sugeridos no pedido para início da sessão não são suportados. Deve mandar em anexo os formatos suportados;
    - *486 Busy Here* – Diz que consegui contactar o utilizador, mas este não pode ou não quer receber mais chamadas.

- *5xx* – Resposta definitiva que indica falha no servidor;
    - *501 Not Implemented* – O servidor não suporta o pedido.
- *6xx* – É uma resposta definitiva, e é considerada uma falha global. Indica que o pedido não pode ser aceite em todos os servidores.
    - *603 Decline* – O utilizador indica que não quer aceitar a chamada.

#### 4.1.1.4. Máquinas de Estado

Uma transacção corresponde ao conjunto de um pedido com as respostas entre os UAs. De acordo com o RFC 3261 (9), para tratar das transacções, é obrigatório o uso de máquinas de estado, sendo necessário a separação entre as transacções que são *INVITE* e as que não são, visto que há uma importância acrescida na entrega da resposta *200 OK* (o protocolo UDP não garante a entrega dos pacotes), para que o UAC responda com um *ACK* e inicie a troca de dados. Ao todo, existem quatro máquinas de estado, duas do lado do cliente e duas do lado do servidor.

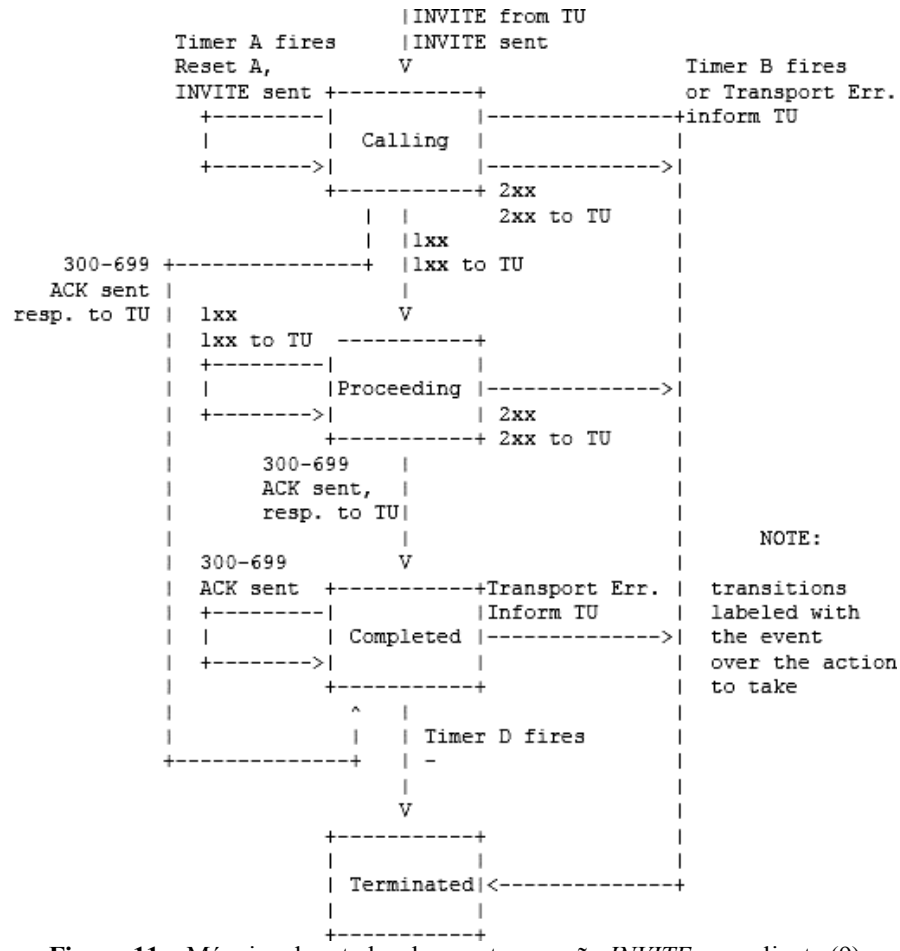


**Figura 11** – Máquina de estados de uma transacção *INVITE* num cliente (9)

Para protocolos de transporte sem confirmação, como é o caso do UDP, os pedidos são retransmitidos a cada T1 segundos, que duplica depois de cada retransmissão. O valor por omissão de T1 é de 500ms. Para os protocolos de transporte (TCP) fiáveis não há retransmissão.

A máquina de estados de transacções *INVITE* num cliente (Figura 11) entra no estado *Calling* quando é iniciada uma nova transacção, ou seja, na criação do pedido *INVITE*. O *timer A* é iniciado com T1 segundos e sempre que dispara é feita a retransmissão do *INVITE* reiniciando o *timer* com 2*T1 segundos. Quando o *timer B* dispara e a transacção ainda se encontra no estado *Calling*, quer dizer que ocorreu um *timeout*. Assim, o estado do diagrama passa para *Terminated*. O timer B é iniciado com um valor de 64*T1 segundos.

Se o cliente recebe uma resposta provisória (com o código 1xx), enquanto está em *Calling*, então o estado transita para *Proceeding*. Neste estado, já não deve haver retransmissão do pedido.

Ao receber uma resposta com o código entre 300 e 699 e se o diagrama de estados se encontrar nos estados *Calling* ou *Proceeding*, então este deve passar para o estado *Completed* e mandar um *ACK* para o mesmo endereço e porta que mandou a resposta. De seguida deve passar para o estado *Terminated* quando o *timer D* disparar. Este *timer* é iniciado com pelo menos 32 segundos no caso da utilização do UDP e 0 segundos se se usar o TCP. Este *timer* corresponde ao tempo que a máquina de estados do Servidor (Figura 13) permanece em *Completed*, pois podem haver retransmissões da resposta e é necessário mandar novamente o *ACK*.

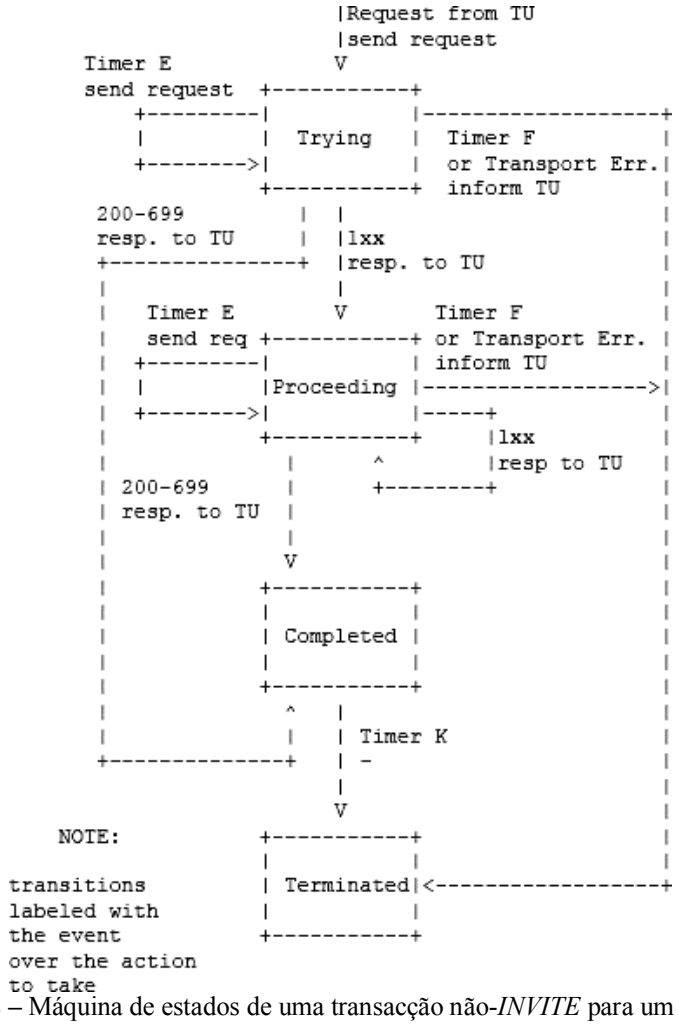


**Figura 12 –** Máquina de estados de uma transacção não-*INVITE* para um cliente (9)

A grande diferença entre as transacções *INVITE* e as não-*INVITE* é o facto das não-*INVITE* não usarem o *ACK*.

Tal como as transacções *INVITE*, os pedidos também são retransmitidos num intervalo que começa em T1 segundos e vai duplicando até chegar T2. No entanto, ao receber uma resposta provisória os pedidos continuam a ser retransmitidos, mas a um intervalo de T2 segundos. Estas retransmissões devem ser respondidas novamente pelo servidor com a mesma mensagem que mandou anteriormente.

Logo, a máquina de estados (Figura 12) é iniciada com o estado *Trying* quando é iniciada uma transacção através de um pedido. O *timer F* é iniciado com 64*T1 segundos. Ao utilizar UDP o *timer E* deve ser iniciado com T1 segundos e quando dispara deve sempre retransmitir o pedido. O *timer F* deve fazer a máquina de estados transitar para o estado *Terminated.*

Quando é recebida uma mensagem provisória, o diagrama deve transitar para *Proceeding*, e, quando recebe uma resposta com o código entre 200 e 699, deve passar para o estado *Completed*.

Do estado *Completed* só deve passar para *Terminated* depois de disparar o *timer K* que é iniciado com T4 segundos (UDP) ou 0 segundos (TCP), pois há a possibilidade de se receber retransmissões de respostas. T4 representa a quantidade de tempo que a rede demora a despachar as mensagens entre cliente e servidor, sendo o seu valor por omissão igual a 5 segundos.

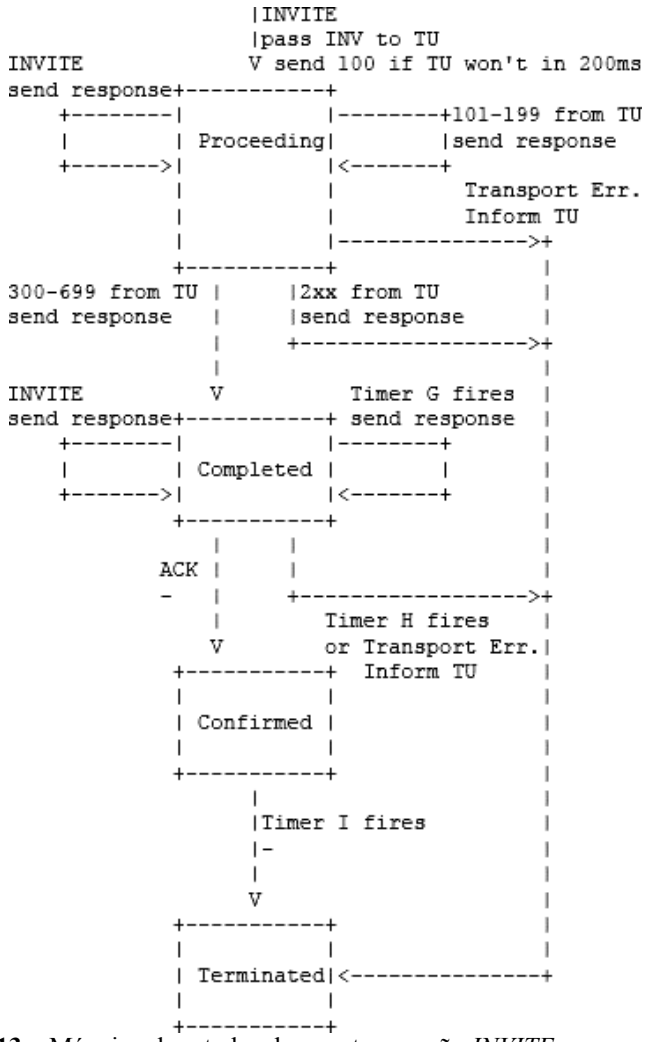


**Figura 13 –** Máquina de estados de uma transacção *INVITE* num servidor (9)

Uma transacção *INVITE* num servidor começa quando este recebe um pedido *INVITE*. Entra, assim, no estado *Proceeding*, podendo ou não mandar uma mensagem provisória *100 Trying*, se souber que não vai gerar uma mensagem nos próximos 200 ms.

Enquanto a máquina de estados se encontrar em *Proceeding*, podem ser enviadas um número infinito de respostas provisórias.

No caso do pedido ser aceite, manda-se uma mensagem do tipo *2xx*, passando o diagrama de estados imediatamente para *Terminated.* Mas se o tipo de resposta for com o código compreendido entre 200 e 699, a máquina de estados passa para *Completed*. O *timer G* dispara a T1 segundos para retransmitir a resposta no caso do protocolo de transporte ser UDP.

Quando se entra no estado *Completed*, o *timer H* deve ser iniciado com 64*T1 segundos para qualquer tipo de transporte. Este *timer* define quando o servidor deve parar de retransmitir a resposta indicando que houve falha na transmissão.

Se for recebido um *ACK* enquanto a máquina de estados se encontrar no estado *Completed*, deve haver uma transição para o estado *Confirmed*, que absorve qualquer *ACK* adicional, enquanto o *timer I* não disparar. Este *timer* é iniciado com T4 segundos para UDP ou 0 segundos para TCP.

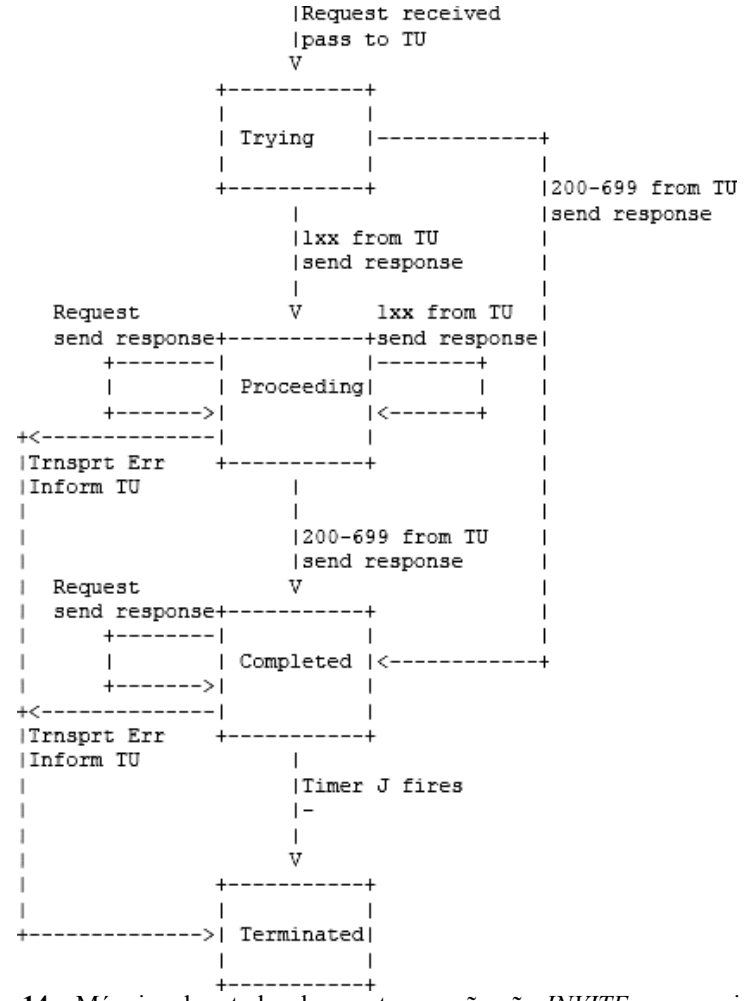


**Figura 14 –** Máquina de estados de uma transacção não-*INVITE* num servidor (9)

A máquina de estados para pedidos não-*INVITE* num servidor é iniciada no estado *Trying*, descartando quaisquer outras retransmissões do mesmo pedido.

Se o servidor passar mensagens provisórias, a máquina deve entrar no estado *Proceeding*. Se for recebida alguma retransmissão do pedido neste estado, deve ser mandada a última resposta provisória.

Quando é passada uma resposta final, o estado deve mudar para *Completed* e deve-se iniciar o *timer J* com 64*T1 segundos para UDP ou 0 segundos para TCP. Depois do *timer* disparar, o diagrama passa para o estado *Terminated*.

#### 4.1.1.5. Terminar Sessão

Para terminar uma sessão é necessário ter em atenção o estado dos pedidos aplicando as seguintes regras:

- Quando um diálogo já está criado, a sessão deve ser terminada com um *BYE*;
- Um UA nunca pode mandar um *BYE* fora de um diálogo;
- O utilizador que inicia a sessão pode mandar *BYE* em diálogos confirmados e não confirmados, mas o utilizador que é chamado só pode mandar *BYEs* em diálogos confirmados, ou seja, depois de receber o *ACK* do utilizador que chama;
- Para cancelar um pedido que ainda não recebeu resposta, deve-se mandar um *CANCEL*.

### 4.1.2. Introdução ao eXoSIP

Esta biblioteca implementa a sinalização SIP. Corresponde a uma extensão de outra biblioteca chamada *oSIP*, tornando-se muito mais fácil de utilizar, pois contém uma camada acima do *oSIP*. Tem a vantagem de se poder utilizar funções do *oSIP* para partes mais complexas e menos comuns da sinalização.

Para os requisitos desta aplicação esta biblioteca só não suporta a negociação SDP, que é feita através do *oSIP*.

A informação indicada nos pontos seguintes trata de aspectos mais técnicos da biblioteca, indicando exemplos básicos de utilização.

#### 4.1.2.1. Inicialização

A primeira coisa a fazer no *eXoSIP* é inicializá-lo juntamente com o *oSIP* e abrir um *socket* para receber pedidos. O exemplo a seguir indica como deve ser feito:

```
#include <eXosip2/eXosip.h>
int i;
TRACE_INITIALIZE (6, stdout);
i=eXosip_init();
if (i!=0)
  return -1;
i = eXosip_listen_addr (IPPROTO_UDP, NULL, port, AF_INET, 0);
if (i!=0)
  {
    eXosip_quit();
    fprintf (stderr, "could not initialize transport layer\n");
    return -1;
  }
```

**Código 4 –** Inicialização do *eXoSIP*

De seguida, é necessário tratar dos eventos (pedidos ou respostas) recebidos:

```
  eXosip_event_t *je;
  for (;;)
    {
      je = eXosip_event_wait (0, 50);
      eXosip_lock();
      eXosip_automatic_action ();
      eXosip_unlock();
      if (je == NULL)
        break;
      if (je->type == EXOSIP_CALL_NEW)
        {
        ....
        ....
        }
      else if (je->type == EXOSIP_CALL_ACK)
        {
        ....
        ....
        }
      else if (je->type == EXOSIP_CALL_ANSWERED)
        {
        ....
        ....
        }
      else .....
      ....
      ....
      eXosip_event_free(je);
    }
```

**Código 5 –** Tratamento de eventos no *eXoSIP*

Como se pode ver, para cada tipo de mensagem deve haver um `if`. Por exemplo, `EXOSIP_CALL_NEW` corresponde ao código de um *INVITE*.

A partir de um evento (`eXosip_event_t`) pode-se extrair os cabeçalhos da mensagem SIP, tais como *Call-ID, From*, etc.

#### 4.1.2.2. Iniciar e terminar chamadas

Para iniciar uma chamada, é necessário construir uma mensagem *INVITE* com todos os cabeçalhos e no conteúdo a mensagem SDP. O exemplo seguinte mostra como se faz.

```
osip_message_t *invite;
int i;
i = eXosip_call_build_initial_invite (&invite,
                          "<sip:to@antisip.com>",
                          "<sip:from@antisip.com>",
                          NULL, // optional route header
                          "This is a call for a conversation");
if (i != 0)
  {
    return -1;
  }
osip_message_set_supported (invite, "100rel");
{
  char tmp[4096];
  char localip[128];
  eXosip_guess_localip (AF_INET, localip, 128);
  snprintf (tmp, 4096,
            "v=0\r\n"
            "o=josua 0 0 IN IP4 %s\r\n"
            "s=conversation\r\n"
            "c=IN IP4 %s\r\n"
            "t=0 0\r\n"
            "m=audio %s RTP/AVP 0 8 101\r\n"
            "a=rtpmap:0 PCMU/8000\r\n"
            "a=rtpmap:8 PCMA/8000\r\n"
            "a=rtpmap:101 telephone-event/8000\r\n"
            "a=fmtp:101 0-11\r\n", localip, localip, port);
  osip_message_set_body (invite, tmp, strlen (tmp));
  osip_message_set_content_type (invite, "application/sdp");
}
eXosip_lock ();
i = eXosip_call_send_initial_invite (invite);
if (i > 0)
  {
    eXosip_call_set_reference (i, reference);
  }
eXosip_unlock ();
return i;
```

**Código 6 –** Exemplo de envio de um *INVITE*

O *INVITE* é construído com a função `eXosip_call_build_initial_invite` e depois é anexada uma *string* com a mensagem SDP. Finalmente, a função `eXosip_call_send_initial_invite` envia o convite.

```
eXosip_lock ();
eXosip_call_send_answer (ca->tid, 180, NULL);
eXosip_unlock ();
```

**Código 7 –** Exemplo de resposta a um pedido

Esta é uma forma genérica de enviar uma resposta. Neste caso, está a mandar um *RINGING*, visto que indica o código 180. Para mandar outros tipos de mensagens

substitui-se o número pelo correspondente à mensagem desejada. Por exemplo, um 400 manda um *Bad Request* (Secção 4.1.1.3). No entanto, para uma mensagem de *OK* é necessário anexar a mensagem SDP, por isso deve-se fazer da seguinte forma:

```
osip_message_t *answer = NULL;
eXosip_lock ();
i = eXosip_call_build_answer (ca->tid, 200, &answer);
if (i != 0)
{
   eXosip_call_send_answer (ca->tid, 400, NULL);
}
else
{
   i = sdp_complete_200ok (ca->did, answer);
   if (i != 0)
   {
      osip_message_free (answer);
      eXosip_call_send_answer (ca->tid, 415, NULL);
   }
   else
      eXosip_call_send_answer (ca->tid, 200, answer);
}
eXosip_unlock ();
```

**Código 8 –** Exemplo de uma mensagem *200 OK*

Neste caso, se não conseguir construir as mensagens manda um *Bad Request* ou um *Unsupported Media Type*, se não suportar o tipo de media ou o tipo de compressão propostos no *INVITE*.

### 4.1.3. Obtenção e detalhes sobre o eXoSIP

O *eXoSIP* é uma extensão do *oSIP*, por isso é necessário ter as duas bibliotecas. Foram utilizados versões *release*: para o *eXoSIP* a versão 2.2.3 e para o *oSIP* a versão 2.2.2.

Pode-se obter esta versão do eXoSIP em (10) e o oSIP em (11).

Para compilar em Windows, o directório `platform/vsnet`, em qualquer uma das bibliotecas, contém um ficheiro de projecto do *Microsoft Visual Studio .NET 2003* versão 7.1 que se pode compilar facilmente.

É necessário ter em atenção e colocar o conteúdo da biblioteca *oSIP* num directório com nome `osip`. Este directório deve-se encontrar no directório pai do do *eXoSIP*.

Para utilizar estas bibliotecas no projecto, deve-se especificar nas propriedades do projecto no *Microsoft Visual Studio .NET 2003*, na parte dos directórios de *include* adicionais, o caminho para os directórios de *include* do *eXoSIP* e do *oSIP*. Também se deve incluir na parte do *Linker* os ficheiros lib `eXosip.lib`, `osip2.lib` e `osipparser2.lib`.

## 4.2. Protocolo SDP

Este protocolo é documentado pelo RFC 2327 (12) e descreve informação necessária para criar sessões. Uma mensagem SDP deve ter informação necessária para se decidir se se pode ou não participar numa sessão.

É utilizado na sinalização para informar os utilizadores o tipo de media e *codecs* que transmitem e recebem.

Assim, o SDP descreve:

- O tipo de media (áudio, vídeo, etc.);
- O protocolo de transporte (RTP/UDP/IP, H.320, etc);
- Endereço e portas remotas para transmitir media.

Uma mensagem SDP é constituída por campos do tipo <tipo>=<valor>, em que existem campos que são obrigatórios:

- v= (versão do protocolo);
- o= (dono/criador e identificador da sessão);
- s= (nome da sessão);
- t= (tempo que a sessão está activa);
- m= (nome de media e endereço de transporte);

Relativamente aos campos não obrigatórios temos:

- i= (informação sobre a sessão);
- u= (URI das descrição);
- e= (endereço e-mail);
- p= (número de telefone);
- c= (informação sobre a ligação);
- b= (informação sobre a largura de banda);
- z= (ajustamentos de fuso horário);
- k= (chave de encriptação);
- a= (atributos da sessão);
- r= (número de repetições);

De seguida, podemos ver como se constrói uma mensagem típica SDP num telefone SIP:

```
char sdp[4096];
snprintf (sdp, 4096,
      "v=0\r\n"
      "o=voviwlan 0 0 IN IP4 %s\r\n"
      "s=conversation\r\n"
      "c=IN IP4 %s\r\n"
      "t=0 0\r\n"
      "m=audio %d RTP/AVP 100 97 112\r\n"
      "a=rtmap:97 speex/8000\r\n"
      "a=rtmap:100 speex/16000\r\n"
      "a=rtmap:112 speex/32000\r\n"
      "m=video %d RTP/AVP 99\r\n"
      "a=rtpmap:99 H263-2000/9000\r\n"
        ,localip, localip, audioport, videoport);
```

**Código 9 –** Exemplo de uma mensagem SDP

Pode-se ver na mensagem acima, através do campo `c`, que se trata de uma ligação IPv4, onde o destino do áudio e vídeo é `localip` para as portas `audioport` e `videoport`. Os campos `a` indicam que os *codecs* aceites são o S*peex* com diversos *bitrates* para a transmissão de áudio e o *H263* para a transmissão de vídeo.

Quando este protocolo é utilizado na sinalização, normalmente o *INVITE* dá uma lista dos *codecs* que podem ser utilizados na sessão. Se este *INVITE* é aceite, então é mandado em anexo da mensagem *OK* o SDP com um só *codec* para cada tipo de media.

# 5. Livro de Endereços

Achou-se que seria útil incluir um livro de endereços na aplicação, para que os utilizadores podessem organizar os seus contactos.

Assim, foi decidido utilizar o livro de endereços do *Windows*, mas com uma interface gráfica integrada com a aplicação. Para isso, utilizou-se a biblioteca *Enhanced Managed .NET WAB Address Book Library*.

O acesso ao livro de endereços implementado está feito de forma a se poder adicionar, modificar e remover endereços. Também é possível a utilização do endereço para realizar uma chamada.

A secção seguinte faz uma introdução a esta biblioteca, dando alguns exemplos de funções para fazer operações com os contactos.

A secção 5.2 mostra onde se obtêm e como se compila esta biblioteca.

## 5.1. Introdução ao Enhanced Managed .NET WAB Address Book Library

A *Enhanced Managed .NET WAB Address Book Library* é uma biblioteca muito simples para código "gerênciado" (*managed*) *.NET* e facilita a integração do livro de endereços do *Windows* numa aplicação.

O que se pretendia com esta biblioteca era adicionar, remover ou editar contactos do livro de endereços do *Windows*.

A função seguinte mostra como se pode listar os contactos em C++:

```
void GetAllContacts()
{
  WAB::AddressBook * addr = new WAB::AddressBook();
  WAB::WABObjectCollection * col =
            addr->GetAllByObjectKind(WAB::WABObjectKind::Contacts);

  for(int i=0;i<col->get_Count();i++)
  {
    WAB::Contact * contact = (WAB::Contact)col[i];
    Console::WriteLine(contact->DisplayName);
  }
}
```

**Código 10 –** Função que lista os contactos do Livro de Endereços do *Windows*

Primeiro, é criado um objecto do tipo `AddressBook`, e depois são seleccionados todos os contactos incluídos no livro de endereços com a função `GetAllByObjectKind`. Seguidamente, é atribuído o contacto a `contact` e impressos na consola, um de cada vez.

O código seguinte mostra como se pode criar contacto:

```
WAB::AddressBook *addr = new WAB::AddressBook();
WAB::Contact * c = addr->CreateContact();
c->set_FirstName("Jose");
c->set_Surname("Silva");
c->EmailAddresses->Add("jose@www.com");
c->Save();
```

**Código 11 –** Exemplo de como criar um contacto no Livro de Endereços do *Windows*

Primeiro, é criado o contacto onde, depois, se introduz o primeiro e último nome através das funções `set_FirstName` e `set_Surname`, respectivamente. De seguida, introduz-se o endereço de e-mail e, finalmente, guarda-se o contacto com a função `Save`.

A aplicação desenvolvida utiliza o campo do *Telefone de casa* para guardar o endereço SIP. Para isso, basta usar a função `set_HomePhone`, tal como as anteriores.

Para apagar um endereço, basta usar a função `Delete` que pertence à classe `Contact`.

## 5.2. Obtenção e detalhes da Enhanced Managed .NET WAB Address Book Library

Foi utilizada a versão 2.0 desta biblioteca e pode-se descarregá-la no seu site em (13).

Para a utilizar, pode-se compilar utilizando o ficheiro do projecto do *Microsoft Visual Studio .NET 2003* versão 7.1 e obter um ficheiro dll para utilizar com as aplicações.

Pode-se também incluir os ficheiros da biblioteca na nossa aplicação, visto que são apenas 5 e têm um tamanho reduzido.

Foi a última opção que se escolheu para este projecto, visto que simplificava a utilização desta biblioteca.

# 6. Aplicação Desenvolvida

A aplicação é constituída por uma parte que utiliza a biblioteca EMIPLIB (Secção 3.5) que faz a captura, transmissão e reprodução de dados de áudio e vídeo em tempo real e por uma parte que faz toda a sinalização das chamadas. Existe também módulos adicionais como livro de endereços e transferência de ficheiros que foram adicionados à aplicação.

## 6.1. Arquitectura

A partir dos requisitos, obteve-se uma aplicação com o diagrama de Arquitectura Lógica representado na Figura 15, que mostra como as diferentes partes da aplicação se interligam:

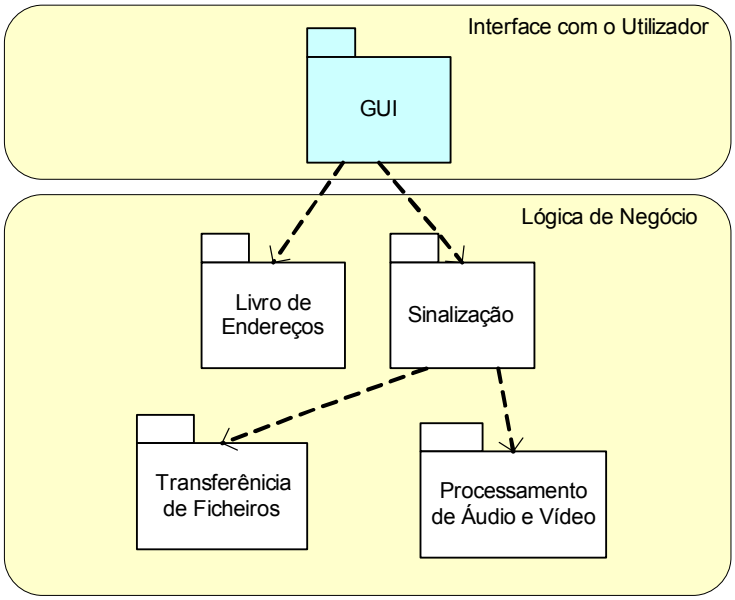


**Figura 15 –** Diagrama da Arquitectura Lógica

**Tabela 3** – Descrição dos pacotes do Diagrama de Arquitectura Lógica

| Pacotes | Descrição |
|---|---|
| GUI | Interface gráfica. Nesta parte, o utilizador terá todos os controlos para os casos de uso. |
| Livro de Endereços | Controla a utilização do livro de endereços. |
| Sinalização | Responsável pela sinalização das chamadas. *Ex. Iniciar chamada, terminar chamada, etc.* |
| Transferência de Ficheiros | Responsável pela transferência dos ficheiros, indicando o seu progresso e possíveis erros. |
| Processamento de Áudio e Vídeo | Captura, comprime, transmite, recebe e descomprime os dados de áudio e vídeo. |

A figura seguinte mostra a Arquitectura física da aplicação desenvolvida.

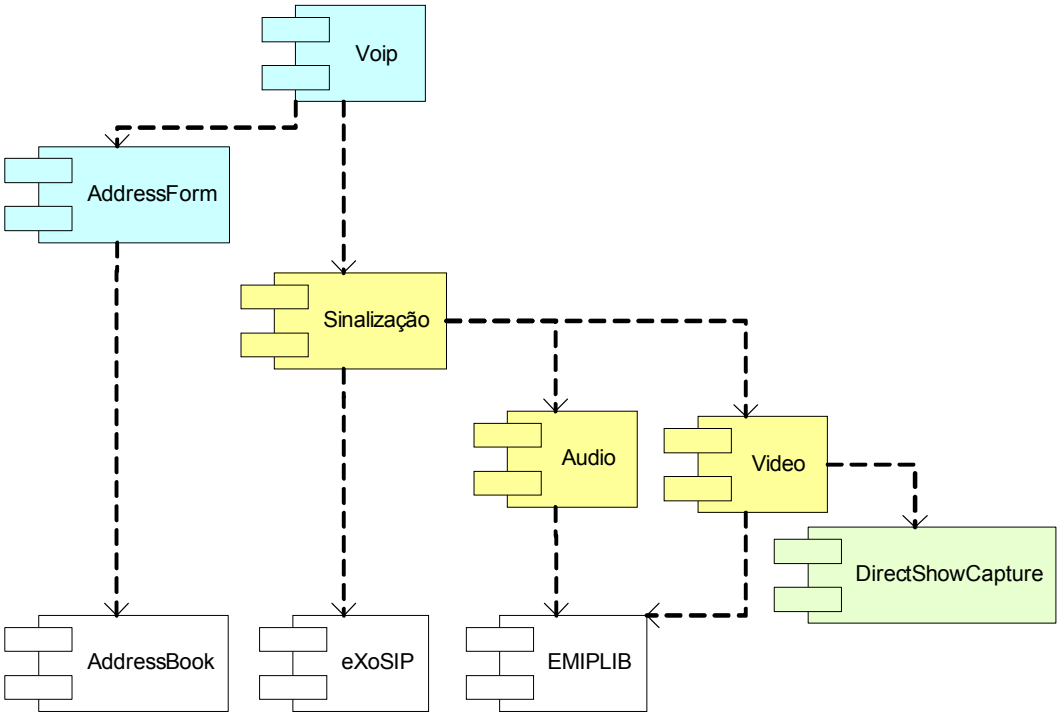


**Figura 16 –** Diagrama de Arquitectura Física

Pode-se notar no diagrama que os pacotes estão organizados por camadas, estando a azul os da interface gráfica, a amarelo os do controlo de dados e mensagens e a branco os das bibliotecas. Pode-se afirmar que o pacote a verde faz parte das bibliotecas, visto ser uma adição ao EMIPLIB.

**Tabela 4 –** Descrição dos pacotes do Diagrama de Arquitectura Física

| Pacote | Descrição |
|---|---|
| Voip | Interface gráfica da aplicação principal |
| AddressForm | Interface gráfica do Livro de Endereços |
| Sinalização | Neste pacote estão incluídas todas as classes utilizadas para fazer a sinalização SIP. |
| Áudio | Pacote com classes que iniciam o dispositivo de captura de áudio e fazem a transmissão, compressão, reprodução, descompressão, etc. do vídeo. |
| Vídeo | Pacote com classes que iniciam o dispositivo de captura de vídeo e fazem a transmissão, compressão, recepção, descompressão, etc. do vídeo. |
| DirectShowCapture | Adição à biblioteca EMIPLIB para poder capturar no formato I420 |
| AddressBook | Biblioteca para utilizar o Livro de Endereços do *Windows* |
| eXoSIP | Biblioteca de sinalização SIP |
| EMIPLIB | Biblioteca de captura transmissão e reprodução de áudio e vídeo |

## 6.2. Capturas de Ecrã

Mostra-se, de seguida, uma captura de ecrã da aplicação principal.

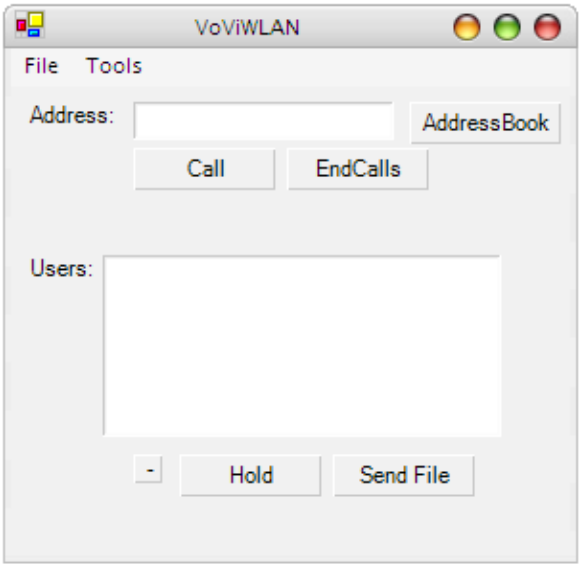


**Figura 17** – Captura de ecrã da aplicação principal

O livro de endereços da aplicação.

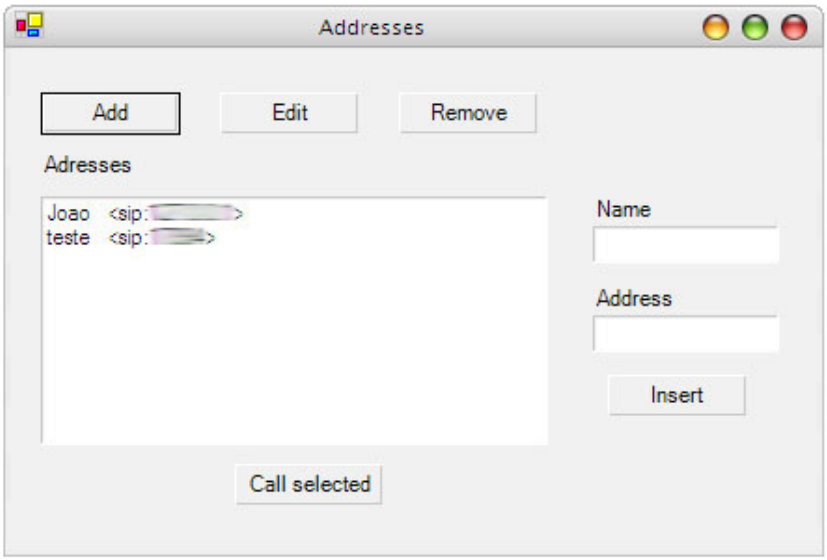


**Figura 18 –** Captura de ecrã do livro de endereços

Diálogo para receber chamada.

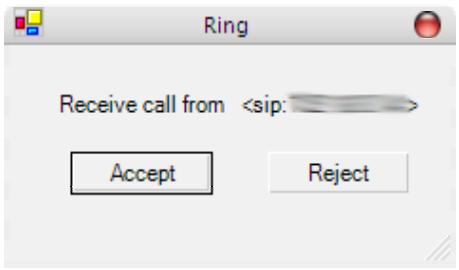


**Figura 19 –** Captura de ecrã do diálogo de receber chamada

Imagem de vídeo proveniente de outro utilizador.


**Figura 20 –** Captura de ecrã do vídeo proveniente de um utilizador

## 6.3. Pequeno manual de utilizador

Para iniciar uma chamada, pode-se introduzir um endereço SIP manualmente ou utilizar o livro de endereços.

Para utilizar o livro de endereços, basta carregar em *Address Book*, seleccionar o contacto desejado e carregar em *Call selected*. Depois, deve-se fechar o livro de endereços e carregar em *Call*.

Sempre que se desejar adicionar mais utilizadores à conversa, repete-se os passos anteriores.

Durante uma chamada, pode ser necessário pôr um utilizador em espera. Para fazê-lo, basta seleccionar o utilizador desejado e carregar em *Hold*.

Há a opção de terminar a chamada com todos os utilizadores ou apenas um. Para todos, deve-se usar o botão *EndCalls* e para apenas um, deve-se seleccionar o utilizador e carregar no botão "-".

Relativamente ao livro de endereços, pode-se adicionar novos contactos, carregando em *Add* e de seguida em *Insert*, depois de preencher os dados. Para editar um contacto, é necessário seleccioná-lo e carregar em *Edit* e de seguida em *OK*, depois de editar os dados. No caso de se pretender eliminar um contacto, deve-se, para isso, seleccionar um contacto e carregar em *Remove*.

# 7. Conclusões

Era esperado que se desenvolvesse uma aplicação que transmitisse voz e vídeo numa rede sem fios. Como os requisitos à partida eram muito pouco detalhados, foi-se em busca de requisitos para se desenvolver a aplicação. Foi definido que se utilizaria a sinalização SIP, que seria criado um livro de endereços, um registo de chamadas efectuadas e recebidas, a transferência de ficheiros.

A partir dos requisitos anteriores obteve-se uma aplicação de transmissão de voz e vídeo sobre IP com sinalização SIP. Adicionou-se à aplicação a possibilidade de utilizar contactos do Livro de Endereços do *Windows* para realizar chamadas, o registo de chamadas recebidas e efectuaras e a reprodução de ficheiros de som durante uma chamada.

Durante o desenvolvimento da aplicação, deparou-se com alguns problemas ligados às bibliotecas que foram utilizadas, principalmente com o EMIPLIB. Para resolver estes problemas, foi necessário construir aplicações de baixa complexidade, para fazer depuração e encontrar os erros na biblioteca.

Fazendo uma apreciação crítica à aplicação implementada, pode-se dizer que, relativamente à sinalização, fez-se uma escolha acertada em relação à biblioteca, visto que, pelos testes efectuados, não se conseguiu encontrar erros. A biblioteca encontra-se já em estado avançado de desenvolvimento e parece ser muito estável e rápida.

Relativamente à biblioteca que gere o áudio e o vídeo (*EMIPLIB*), verificou-se ser um pouco instável, mas que, independentemente dos erros, implementa bem o que era necessário, tendo-se revelado também uma boa escolha. Obviamente que deverão existir outra bibliotecas ou conjunto de bibliotecas que seriam mais apropriadas para este projecto.

Como possíveis melhoramentos da aplicação, seria desejável a introdução de novos *codecs*, nomeadamente os mais clássicos, como o PCMA (*G.711 A-law*) e o PCMU (*G.711 μ-law*).

Seria também desejável o suporte de STUN para se poder realizar chamadas estando por detrás de uma NAT.

Poderia ser interessante, acrescentar à aplicação uma interface gráfica, para se poder assistir a reuniões à distância, onde se mostrariam os diapositivos que estavam a ser apresentados no projector, juntamente com os vídeos dos outros utilizadores. Deveria haver uma interface especial no computador que apresenta, de forma a poder mostrar os diapositivos no projector. No ecrã do computador, não só seria possível controlar os diapositivos, mas também mostrar o vídeo proveniente dos outros utilizadores.

Converter a aplicação para *Linux* seria relativamente fácil, sendo para isso apenas necessário criar uma interface gráfica nova à aplicação.

Em conclusão, obteve-se a aplicação prevista, podendo-se adicionar sempre cada vez mais componentes. Esta aplicação poderá ficar como uma base para desenvolver novas funcionalidades ou integrá-la com outros tipos de programas.

# Referências

[1] ANACOM, http://www.anacom.pt

[2] *Speex*, http://www.speex.org/

[3] MSDN, *Introduction to DirectShow Application Programming*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/introductiontodirectshowapplicationprogramming.asp

[4] Schulzrinne, H., *RTP: A Transport Protocol for Real-Time Applications*, http://www.ietf.org/rfc/rfc3550.txt, RFC3550, Julho de 2003

[5] *EMIPLIB,* http://research.edm.luc.ac.be/emiplib/emiplib.html

[6] *JRTPLIB*, http://research.edm.luc.ac.be/jori/jrtplib/jrtplib.html

[7] *JThread*, http://research.edm.luc.ac.be/jori/jthread/jthread.html

[8] *Qt* versão 3.3.5, ftp://ftp.trolltech.com/qt/source/qt-x11-free-3.3.5.tar.gz

[9] Rosenber, J., *SIP: Session Initiation Protocol*, http://www.ietf.org/rfc/rfc3261.txt, RFC 3261, Junho de 2002

[10] *eXoSIP*, http://download.savannah.nongnu.org/releases/exosip/

[11] *oSIP*, http://ftp.gnu.org/gnu/osip/

[12] Handley, M., Jacobson, V, *SDP: Session Description Protocol*, http://www.ietf.org/rfc/rfc2327.txt, RFC 2327, Abril de 1998

[13] *Enhanced Managed .NET WAB Address Book Library*, http://www.sichbo.ca/DotNetCode/1204

# Anexo A

## Bibliografia

# Bibliografia

Wikipedia, *Audio and Data Compression*, http://en.wikipedia.org/wiki/Audio_data_compression

MSDN, *DirectShow*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/directshow.asp

MSDN, *Recording and Playing Sound with the Waveform Audio Interface*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetcomp/html/WaveInOut.asp

*SIP*, http://www.voip-info.org/wiki-SIP

Wikipedia, *Session Initiation Protocol*, http://www.voip-info.org/wiki-SIP

Packetizer, Inc*, H.323 versus SIP: A Comparison*, http://www.packetizer.com/voip/h323_vs_sip/

Cisco Systems, Inc, *Overview of the Session Initiation Protocol,* *http://www.cisco.com/univercd/cc/td/doc/product/voice/sipsols/biggulp/bgsipov.htm*

*SIP versus H.323*, http://www.iptel.org/info/trends/sip.html

*libeXoSIP2 Documentation*, http://www.antisip.com/documentation/eXosip2/index.html