UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

CRIAÇÃO DE *STREAMING* DE VÍDEO PARA TRANSMISSÃO DE SINAIS DE VÍDEO EM TEMPO REAL PELA INTERNET

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

CLODOALDO TSCHÖKE

BLUMENAU, NOVEMBRO/2001

2001/2-11

CRIAÇÃO DE *STREAMING* DE VÍDEO PARA TRANSMISSÃO DE SINAIS DE VÍDEO EM TEMPO REAL PELA INTERNET

CLODOALDO TSCHÖKE

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Francisco Adell Péricas — Orientador na FURB

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

Prof. Francisco Adell Péricas

Prof. Paulo Cesar Rodacki Gomes

Prof. Miguel Alexandre Wisintainer

SUMÁRIO

Lista de figuras ..... **v**

Lista de quadros ..... **vi**

Agradecimentos ..... **vii**

Resumo ..... **viii**

Abstract ..... **ix**

1 INTRODUÇÃO **1**

1.1 Objetivos **2**

1.2 Justificativa **2**

1.3 Organização do trabalho **3**

2 INTERNET **4**

2.1 Origem da Internet **4**

2.1.1 A Internet no Brasil 6

2.2 A arquitetura Internet **6**

2.2.1 Camada de enlace 8

2.2.2 Camada de rede 9

2.2.3 Camada de transporte 9

2.2.3.1 Endereçamento para as aplicações 10

2.2.3.2 O protocolo TCP 12

2.2.3.3 O protocolo UDP 13

2.2.4 Camada de aplicação 14

3 MULTIMEDIA NETWORKING **16**

3.1 Aplicações multimídia de rede **16**

3.1.1 Streaming de vídeo armazenado 18

3.1.2 Streaming de vídeo ao vivo 18

3.1.3 Vídeo interativo em tempo real 19

3.2 Streaming de vídeo na Internet **19**

3.3 Protocolos de tempo real **20**

3.3.1 Real-Time Streaming Protocol (RTSP) 21

3.3.2 Real-Time Protocol (RTP) 23

3.3.3 Real-Time Control Protocol (RTCP) 25

3.3.3.1 Tipos de pacotes RTCP 26
3.3.4 Resource ReServation Protocol (RSVP) 27
3.4 Enviando multimídia de um servidor de streaming para uma aplicação 28
3.5 Codificação e compressão de vídeo 30
4 TÉCNICAS E FERRAMENTAS UTILIZADAS 32
4.1 Orientação a objetos 32
4.2 Linguagem Unificada de Modelagem – UML 33
4.3 Windows sockets API 34
4.4 A biblioteca JRTPLIB 35
4.5 Programação visual (ambiente Visual C++) 35
4.6 Ferramenta CASE Rational Rose 37
4.7 Hardware para captura do sinal de TV 37
5 DESENVOLVIMENTO DO PROTÓTIPO DE SOFTWARE 39
5.1 Especificação do protótipo 40
5.1.1 Diagrama de caso de uso 40
5.1.2 Diagrama de classes 42
5.1.3 Diagramas de seqüência 43
5.2 Implementação 47
5.2.1 O software servidor 47
5.2.2 O software cliente 50
6 TRABALHOS CORRELATOS 52
6.1 Windows Media Services 53
6.1.1 Protocolos utilizados 54
6.2 Real System 56
6.2.1 Protocolos utilizados 57
6.2.2 Configurações de portas 58
7 CONCLUSÕES 59
7.1 Limitações 60
7.2 Sugestões 60
ANEXO 1 – Código fonte do protótipo 61
REFERÊNCIAS BIBLIOGRÁFICAS 71

LISTA DE FIGURAS

Figura 1 – Modelo de referência TCP/IP 7
Figura 2 – Interconexão de sub-redes em uma rede Internet 8
Figura 3 – Endereçamento para as aplicações 11
Figura 4 – Interação entre cliente e servidor usando RTSP 22
Figura 5 – Dado RTP em um pacote IP 24
Figura 6 – O RTP é parte da camada da aplicação 25
Figura 7 – Reserva de um nó da rede no caminho do fluxo de dados 27
Figura 8 – Streaming de um servidor para um reprodutor de mídia 29
Figura 9 – Exemplo de classe e objeto 32
Figura 10 – Evolução da UML 34
Figura 11 – Ferramenta Rational Rose 37
Figura 12 – Visão geral da arquitetura do protótipo 40
Figura 13 – Diagrama de caso de uso 41
Figura 14 – Diagrama de classes 42
Figura 15 – Diagrama de seqüência Inicia Sessão 43
Figura 16 – Diagrama de seqüência Termina Sessão 44
Figura 17 – Diagrama de seqüência Reproduz stream 44
Figura 18 – Diagrama de seqüência Estabelece Comunicação 45
Figura 19 – Diagrama de seqüência Termina conexão 45
Figura 20 – Diagrama de seqüência Envia Requisição 46
Figura 21 – Diagrama de Seqüência Recebe Mensagens 46
Figura 22 – Tela inicial do protótipo do servidor 47
Figura 23 – Iniciar uma sessão 48
Figura 24 – Iniciar a captura do sinal de vídeo 49
Figura 25 – Uma sessão em andamento 49
Figura 26 – Detalhes das conexões ativas 50
Figura 27 – Tela principal do reprodutor de vídeo 50
Figura 28 – Conexão do cliente ao servidor 51
Figura 29 – Cliente reproduzindo o vídeo transmitido pelo servidor 51
Figura 30 – Visão geral da arquitetura Windows Media Services 54

Figura 31 – Uso dos protocolos na arquitetura Windows Media Services.....55
Figura 32 – Visão geral dos componentes do Real System.....56

LISTA DE QUADROS

Quadro 1 – Troca de mensagens RTSP entre um cliente e o servidor 23
Quadro 2 – Descrição dos casos de uso.....41
Quadro 3 – Descrição das classes.....43
Quadro 4 – Implementação do método **EnviarRequisicao** 61
Quadro 5 – Continuação da implementação do método **EnviarRequisicao** 62
Quadro 6 – Implementação do método **GeraMensagem** na classe **CClienteRTSP**.....63
Quadro 7 – Implementação do método **ThreadCliente**.....64
Quadro 8 – Continuação (1) da implementação do método **ThreadCliente** 65
Quadro 9 – Continuação (2) da implementação do método **ThreadCliente** 66
Quadro 10 – Implementação do método **AnalisaRequisicao**.....67
Quadro 11 – Continuação (1) da implementação do método **AnalisaRequisicao**.....68
Quadro 12 – Continuação (2) da implementação do método **AnalisaRequisicao**.....69
Quadro 13 – Implementação do médoto **GeraMensagem** 70

AGRADECIMENTOS

Aos meus pais José Canísio Tschöke e Guisela Tschöke e à minha namorada, Leslie Lia Hermes, por estarem sempre ao meu lado, apoiando, incentivando e, principalmente, compreendendo minhas ausências durante todos esses anos.

A todos que de alguma forma contribuíram para a realização deste trabalho.

Ao professor Francisco Adell Péricas, pela orientação e atenção dispensada na elaboração deste trabalho.

A Deus, o grande arquiteto do Universo, que nos acompanha e ilumina em todos os momentos.

RESUMO

O presente trabalho destina-se ao estudo da transmissão e recepção de vídeo em tempo real pela Internet. Apresenta, ainda, considerações sobre os protocolos RTSP (*Real-Time Streaming Protocol*), RTP (*Real-Time Streaming Protocol*), RTCP (*Real-Time Control Protocol*) e RSVP (*Resource ReServation Protocol*), especificados para implementação de aplicações multimídia para redes TCP/IP. O estudo tem como resultado o desenvolvimento de um software para capturar e transmitir um sinal de vídeo pela Internet utilizando as tecnologias descritas.

ABSTRACT

This work devotes to study the transmission and reception of audio and video through the Internet in real time. It still presents considerations about the following protocols: RTSP (*Real-Time Streaming Protocol*), RTP (*Real-Time Streaming Protocol*), RTCP (*Real-Time Control Protocol*) and RSVP (*Resource ReServation Protocol*). These protocols are useful for multimedia applications development for TCP/IP networks. As a result of this work, it was developed a software that's able to capture and transmit a video signal through Internet, using those described technologies.

1 INTRODUÇÃO

O crescimento acelerado da Internet tem determinado um aumento do seu leque de serviços, visando atender às diversas necessidades dos usuários. A princípio existiam apenas páginas HTML bastante simplórias, os correios eletrônicos e as conseqüentes transferências de arquivos entre computadores. Porém, segundo (HUG2000), nos últimos anos, a Internet começa a obter sucesso crescente em áreas como comércio eletrônico e outras formas de comunicação eletrônica.

O *streaming media*, uma técnica para transferência de dados na qual transmite-se áudio e/ou vídeo através da Internet (ou alguma outra rede) em tempo real, destaca-se entre tais novas formas de comunicação, conhecida como *webcasting*. De acordo com (MAR2001), essa nova tecnologia pode servir aos mais variados interesses, que vão da videoconferência ao vídeo sob demanda, passando pelo Web TV até os jogos interativos.

Segundo (HUG2000), essa é uma tecnologia relativamente nova, pois, nos princípios da Internet, os arquivos de vídeo tinham de ser transferidos por inteiro para o disco rígido do usuário antes de serem vistos ou ouvidos. Assim, o usuário tinha de esperar até que o arquivo fosse transferido na sua totalidade para então poder visualizá-lo. Atualmente, as principais tecnologias de *streaming* de cunho comercial utilizadas na Internet pertencem à Real Networks (http://www.real.com) através do Real Player e à Microsoft (http://www.microsoft.com), com o Windows Media Player.

O software desenvolvido neste trabalho foi especificado e implementado baseado na arquitetura cliente/servidor, onde:

a) o servidor realiza a captura do sinal externo de vídeo e o transmite para os clientes que estão conectados;
b) o cliente conecta-se ao servidor e controla a exibição do vídeo na tela.

Para a troca de mensagens entre os programas, foi implementado o protocolo RTSP (*Real-Time Streaming Protocol*), permitindo que o cliente solicite ao servidor o início e término da transmissão de *streaming* de vídeo.

Para o envio do sinal de vídeo aos clientes conectados, foi utilizado a biblioteca de domínio livre *JRTPLIB*, que implementa os protocolos RTP (*Real-Time Protocol*) e RTCP

(*Real-Time Stream Protocol*) (LIE2000).

Para a especificação do protótipo, foi utilizada a UML (*Unified Modeling Language*), definida por (FUR1998) como uma linguagem de modelagem para aplicações orientadas à objetos.

O ambiente utilizado para a implementação do software foi o Visual C++ 6.0, da Microsoft.

1.1 OBJETIVOS

O objetivo principal deste trabalho é transformar um sinal de vídeo digital externo capturado por uma placa de vídeo, em um sinal de *streaming media* para a transmissão em tempo real pela Internet.

1.2 JUSTIFICATIVA

Conforme descrição anterior, atualmente já existem alguns softwares que podem ser facilmente adquiridos e parametrizados de modo a disponibilizar um sinal de vídeo na Internet. Porém, não importando qual seja sua procedência, um software dessa categoria deve seguir um conjunto de especificações padrão definidas por grupos de estudo como a IETF *(Internet Engineering Task Force)* para que o sinal possa ser transmitido e visualizado por qualquer computador conectado à rede mundial.

Como a Internet não foi inicialmente constituída para a transmissão desse tipo de informação, com pouca largura de banda e um crescente número de computadores domésticos conectados à rede, torna-se um desafio implementar uma aplicação para esse fim que possa oferecer ao usuário uma transmissão com qualidade aceitável. Além disso, a transmissão de vídeo envolve um grande volume de dados e a necessidade de transmissão síncrona e em tempo real requerem certos cuidados que deverão ser considerados durante o estudo.

A importância deste trabalho está em apresentar um software capaz de realizar a captura do sinal de vídeo externo, fornecido por uma placa de captura de vídeo, e gerar um sinal de *streaming* que será transmitido em tempo real pela Internet, permitindo que qualquer usuário estabeleça uma conexão para visualizar em seu computador o que antes era possível somente na televisão.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está disposto em 7 capítulos descritos a seguir:

O primeiro capítulo apresenta a introdução e os objetivos pretendidos com a elaboração do trabalho.

O segundo capítulo descreve a Arquitetura Internet no que diz respeito a conceitos, características, modelos e protocolos.

O terceiro capítulo está voltado para a transmissão de conteúdo multimídia na Internet, classificando as aplicações e protocolos envolvidos no processo.

O quarto capítulo apresenta as técnicas e ferramentas utilizadas.

O quinto capítulo destina-se ao desenvolvimento do protótipo, onde são apresentadas a especificação e implementação do mesmo.

O sexto capítulo fornece uma relação de trabalhos correlatos ao desenvolvido na pesquisa.

O sétimo capítulo finaliza o trabalho, apresentando as conclusões, limitações e sugestões para novos trabalhos.

2 INTERNET

A Internet é um conjunto de redes de computadores interligadas pelo mundo inteiro, que têm em comum um conjunto de protocolos e serviços de informação e comunicação de alcance mundial (CYC2000).

Pode-se definir a Internet como uma verdadeira comunidade virtual, formada por todos os usuários dessas redes de computadores com os mais diversos intuitos: troca de mensagens no correio eletrônico, debates ao vivo (IRC – *Internet Relay Chat*), grupos de discussão (*Usenet* e *Mailing List*), entretenimento (jogos), e até comércio. Em virtude da riqueza de possibilidades criou-se o conceito de *cyberspace*, ou espaço cibernético, um mundo eletrônico paralelo altamente democrático, onde já começa a se formar uma base de conhecimento universal (REG2000).

Para (REG2000), os padrões da Internet não são criados por órgãos internacionais de padronização, como a *International Organization for Standardization* (ISO) ou o *Institute of Electrical and Electronics Engineering* (IEEE), porém, sua arquitetura é muito aceita, sendo chamada por isso como padrão "**de facto**", ao contrário do modelo *Open System Interconnection* (OSI), considerado padrão "**de jure**".

Na verdade, diversos grupos orientam o crescimento da Internet auxiliando no estabelecimento de padrões e orientando as pessoas sobre a maneira adequada de utilizá-la. Talvez o mais importante seja a *Internet Society*, um grupo privado sem fins lucrativos que suporta o trabalho da *Internet Activities Board* (IAB), que coordena a elaboração de padrões e protocolos da Internet e designa endereços IP da rede através da *Internet Assigned Numbers Authority* (IANA). Além disso, dirige a *Internet Registry* (Central de Registros da Internet), que controla o *Domain Name System* (Sistema de nomes de Domínio) e trata da associação de nomes de referência a endereços IP (GRA1996).

2.1 ORIGEM DA INTERNET

Segundo (COM2001), as primeiras redes foram projetadas para compartilhar poder computacional em grande escala. Os primeiros computadores digitais eram extremamente caros e escassos e o governo dos Estados Unidos, grande financiador da pesquisa em ciência e engenharia, desfrutava de um orçamento insuficiente para fornecer computadores a todos os

cientistas e engenheiros. Tornou-se necessária, então, a otimização do uso dos recursos até então disponíveis, visto que os mesmos eram cruciais para os avanços da ciência e da tecnologia.

A Agência de Projetos de Pesquisa Avançada (*Advanced Research Projects Agency, ARPA*) do Departamento de Defesa dos EUA estava especialmente preocupada com essa falta de computadores, pois muitos de seus projetos necessitavam de acesso a equipamentos de última geração. Como os grupos de pesquisa precisavam de uma unidade de cada modelo novo de computador que surgia, ao final dos anos 60, tornou-se óbvio que o orçamento da ARPA não poderia acompanhar a demanda. Assim, ao invés de colocar múltiplos computadores em cada centro de pesquisa, a ARPA começou a investigar a ligação em redes de dados, colocando um computador em cada grupo e interconectando-o a uma rede de dados com um software que permitiria a um pesquisador utilizar o computador mais adequado para a execução de uma determinada tarefa.

A ARPA enfrentou muitos desafios no início do seu projeto de ligação em rede, pois ninguém sabia como construir uma rede de dados grande e eficiente, ou mesmo os programas aplicativos para utilizar tal rede. Sua pesquisa, porém, acabou sendo revolucionária. Escolhendo uma abordagem totalmente nova, conhecida como *comutação de pacotes*[1], transformou-se na base para todas as futuras redes de dados. A ARPA focalizou-se na pesquisa em redes para transformar os projetos em um sistema operante chamado ARPANET, financiando pesquisas sobre tecnologias alternativas, aplicações de rede, e uma tecnologia conhecida como ligação inter-redes (*internetworking*) que, nos anos 70, tinha se tornado o foco da pesquisa da ARPA, e mais tarde levaria ao surgimento da *Internet*. A pesquisa continuou nos anos 80, tornando-se a Internet um sucesso comercial nos anos 90 (COM2001).

Hoje as redes interligam-se através da Internet, utilizando o computador para navegar no mar de informações existente dentro do *cyberspace*, compartilhando sinais inteligentes com o resto do mundo (REG2000).

[CT1] Comentário: Verificar na bibliografia

[1] Divisão dos dados em pequenos pacotes que são enviados individualmente pela rede e reagrupados no receptor (COM2001). A idéia original da comutação de pacotes foi apresentada por Leonard Kleinrock, em sua proposta de Ph.D. no MIT em 1959.

2.1.1 A INTERNET NO BRASIL

A Internet chegou ao Brasil em 1988 por iniciativa da comunidade acadêmica de São Paulo (Fundação de Amparo à Pesquisa do Estado de São Paulo – Fapesp) e do Rio de Janeiro (Universidade Federal do Rio de Janeiro – UFRJ e Laboratório Nacional de Computação Científica – LNCC) (CYC2000).

Em 1989 foi criada, pelo Ministério de Ciência e Tecnologia, a Rede Nacional de Pesquisas (RNP), contendo um *backbone* interligando instituições educacionais à Internet. Este *backbone*, conhecido como *backbone* RNP, inicialmente interligava onze estados por meio de Pontos de Presença (*Point of Presence* – Pop) em suas capitais.

De acordo com (CYC2000), a exploração comercial da Internet foi iniciada em dezembro de 1994 com um projeto piloto da Embratel. Inicialmente o acesso à Internet efetuou-se por meio de linhas discadas, e posteriormente (abril de 1995) por acessos dedicados via Renpac.

De 1995 até os dias de hoje, o mercado de Internet no Brasil popularizou-se e amadureceu. Atualmente, este mercado é composto por provedores de *backbone*, provedores de acesso e de conteúdo de diversas origens e tamanhos; porém, tal panorama já está sofrendo modificações devido à entrada de grandes empresas no setor e de novas tecnologias de redes que permitem aos usuários acesso mais rápido, tais como ISDN, xDSL e Cable Modem.

[CT2] Comentário: Ver o que significa a sigla

[CT3] Comentário: Ver o que significa a sigla

2.2 A ARQUITETURA INTERNET

Segundo (CYC2000), a Internet é a mais bem sucedida aplicação prática do conceito de *internetworking*, que consiste em conectividade de redes de tecnologias distintas; essa conectividade foi conseguida pelo uso do conjunto de protocolos conhecido como TCP/IP, que surgiu em 1975 na rede ARPANET. Suas especificações são públicas e genéricas, o que permite a implementação por diversos fabricantes.

O TCP/IP (*Transmission Control Protocol / Internet Protocol*), ao contrário do que parece, não é apenas um protocolo, mas são dois. Enquanto o IP é responsável pelo encaminhamento de pacotes de dados entre diversas sub-redes desde a origem até o destino, o TCP tem por função o transporte fim-a-fim confiável de mensagens de dados entre dois

sistemas (REG2000).

Na pilha de protocolos TCP/IP, o endereçamento é especificado pelo IP (*Internet Protocol*), onde a cada *host*[2], é atribuído um número de 32 bits único conhecido como endereço IP. Cada pacote enviado através de uma inter-rede contém o endereço IP do remetente (origem) e do receptor pretendido (destino) (COM2001).

O TCP fornece um serviço de transporte de dados orientado à conexão, full-duplex e totalmente confiável (nenhuma duplicação ou perda de dados), que permite a dois programas aplicativos formarem uma conexão, enviarem dados em uma ou outra direção e então terminarem a conexão. Cada conexão de TCP é iniciada confiavelmente e terminada graciosamente, com todos os dados sendo entregues antes da terminação da conexão acontecer (CYC2000). Para uso em redes de alta qualidade ou em conexões onde o problema de confiabilidade não assume grande importância, foi definido o protocolo UDP (*User Datagram Protocol*) que opera no modo sem conexão e possui funcionalidades bem mais simples que o TCP.

Segundo (PER2001), o modelo de referência TCP/IP, especificado pela IETF[3], é baseado em um conjunto de quatro camadas:

FIGURA 1 – MODELO DE REFERÊNCIA TCP/IP

Fonte: (PER2001)

[2] Um computador de usuário final conectado a uma rede. Em uma inter-rede, cada computador é classificado como um host ou um roteador

[3] *The Internet Engineering Task Force*: grupos de pesquisadores e técnicos responsáveis pelo desenvolvimento de padrões para o funcionamento da Internet.

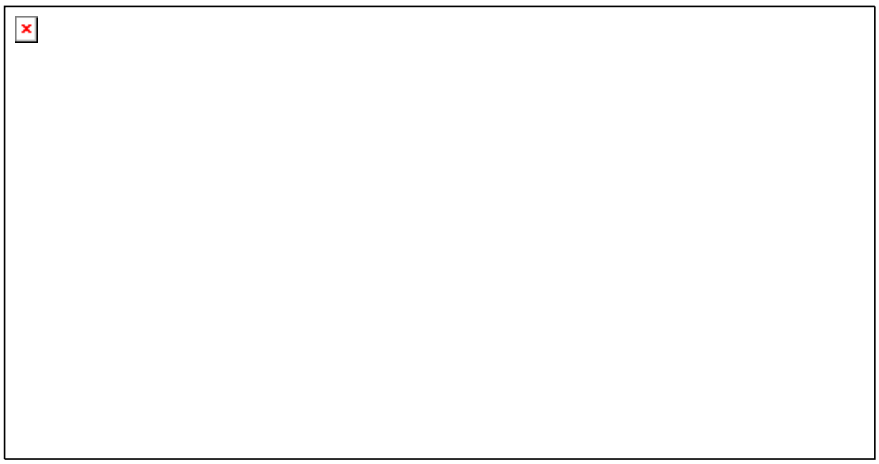
2.2.1 CAMADA DE ENLACE

Consiste em rotinas de acesso à rede física. A camada de enlace interage com o hardware, permitindo que as demais camadas sejam independentes do hardware utilizado. A arquitetura Internet não define um padrão para a camada física de acesso ao meio, permitindo, portanto, empregar qualquer tecnologia para interconectar um sistema computacional a uma rede Internet, bastando para isso que sejam desenvolvidas as respectivas interfaces de comunicação entre o IP e cada rede.

Dessa forma, pode-se encontrar uma grande variedade de tecnologias diferentes de redes, cada uma, possuindo seus próprios protocolos, esquemas de endereçamento, taxas de transmissão e meios físicos. Todavia, tais tecnologias precisam ser vistas pela Internet como se fossem iguais. A compatibilização de tais redes é realizada pelos *gateways*, que permitem que conjuntos de redes heterogêneas interoperem, enxergando de um lado as redes aos quais está interconectado e, de outro lado, enxergando uma rede IP, que utiliza um esquema padronizado de endereçamento. Deste modo, as diferenças existentes entre as redes interconectadas tornam-se transparentes ao usuário do sistema (TAN1996).

Dentre as tecnologias de rede existentes, podemos citar: Ethernet, Token Ring, FDDI (*Fiber Distributed Data Interface*), X.25, Frame Relay, ATM, etc.

FIGURA 2 – INTERCONEXÃO DE SUB-REDES EM UMA REDE INTERNET



Fonte: (TAN1996)

2.2.2 CAMADA DE REDE

A camada de rede, também chamada de Internet, é responsável pelo endereçamento dos equipamentos e pelo roteamento dos dados na rede.

O endereçamento de equipamentos na Internet é baseado em um identificador que independe da tecnologia de rede envolvida, conhecido como endereço IP. Este endereço, em modo normal, está associado a um único equipamento, identificando a rede e o equipamento nessa rede. Existe também um outro modo de endereçamento conhecido como endereçamento *multicast*, em que um endereço IP está associado a um grupo de computadores (CYC2000).

O roteamento na Internet é a tarefa executada pelo protocolo IP (*Internet Protocol*), que é o responsável pela entrega das informações geradas pelas aplicações aos seus destinatários. De acordo com (TAN1996), o IP é o principal protocolo desta camada e opera no modo *datagrama*, isto é, sem conexão. Isso significa que não são pré-estabelecidos circuitos virtuais entre a origem e o destino para o transporte dos pacotes de dados. Segundo (REG2000), a operação no modo *datagrama* é uma comunicação não confiável, não sendo usado nenhum reconhecimento fim-a-fim ou entre nós intermediários, nem qualquer tipo de controle de fluxo. Nenhum mecanismo de controle de erro de dados é utilizado, apenas um controle de verificação do cabeçalho, para garantir que os *gateways* encaminhem as mensagens corretamente.

2.2.3 CAMADA DE TRANSPORTE

A camada de transporte tem como principal objetivo prover a transferência confiável de dados entre processos de diferentes estações pertencentes ou não à mesma rede, garantindo que os dados sejam entregues livres de erro e em seqüência, sem perdas ou duplicação, liberando, portanto, as camadas superiores dessas arquiteturas do encargo de gerenciar a infra-estrutura de comunicação usada pelos mais variados tipos de aplicação (CAR1994).

Para a camada de transporte, a arquitetura Internet especifica dois tipos de protocolos:

a) **TCP (Transmission Control Protocol)**: fornece um serviço orientado à conexão para a camada de aplicação, garantindo a entrega dos dados na ordem correta;

b) **UDP (User Datagram Protocol)**: fornece um serviço sem conexão para a camada de aplicação e não garante a entrega dos dados, mas apresenta menos sobrecarga que o TCP.

A utilização de um ou de outro protocolo depende das necessidades da aplicação considerada (qualidade de serviço, tipo de dados, etc.). Dentro da arquitetura Internet, aplicações tais como a de gerenciamento, utilizam o UDP, e outras, como a de transferência de arquivos, utilizam o TCP.

2.2.3.1 ENDEREÇAMENTO PARA AS APLICAÇÕES

O endereçamento dos programas aplicativos é realizado pela camada de Transporte, e é baseado em um identificador conhecido como *port*. Esse identificador é um número de 16 bits que pode assumir valores entre 1 e 65535.

Para obter endereço em uma Internet, um programa abre um canal de comunicação com a camada de Transporte, escolhendo um de seus protocolos e um valor para esse identificador; a partir de então, passa a ter a possibilidade de se comunicar com outros programas que rodam nos demais equipamentos dessa Internet. Embora esse processo de abertura de canal varie conforme o Sistema Operacional dos equipamentos, ele é sempre identificado por esse número.

A associação entre os valores numéricos dos *ports* e os programas de aplicação não é unívoca, isto é, mais de um programa pode estar usando um mesmo número de *port*, bem como um programa pode abrir mais de um canal com a camada de Transporte, e, portanto, ter mais de um identificador associado.

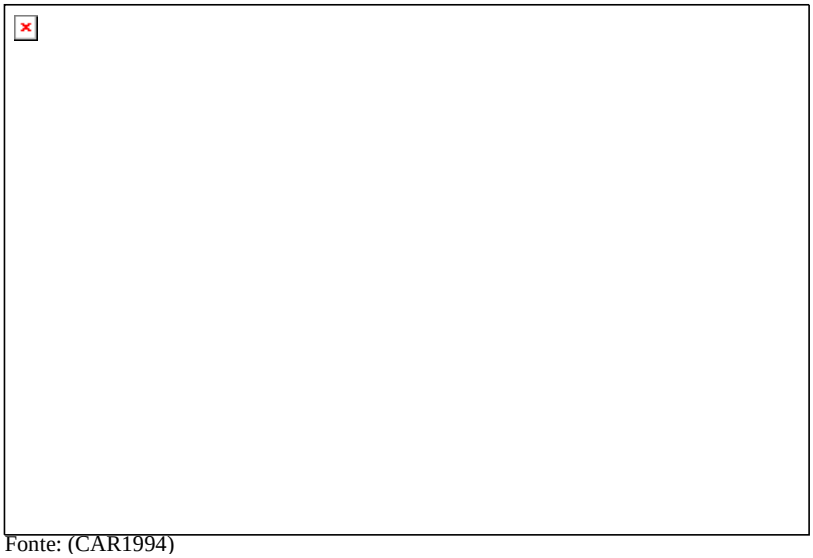
Assim, a comunicação entre programas de aplicação em uma rede TCP/IP se dá baseada em três parâmetros: o endereço IP dos equipamentos em que os programas rodam, o protocolo da camada de Transporte escolhido para a comunicação (TCP ou UDP), e os valores dos *ports* escolhidos pelos programas. Essa tríade de parâmetros em cada programa constitui um *end-point*, ou *socket*.

A alocação de um *port* em um equipamento pode ser feita de forma livre (alocação do primeiro *port* livre, por exemplo) ou específica (alocação de um *port* predeterminado), sendo o uso de uma das duas formas adotado conforme o tipo de aplicação envolvido.

Em aplicações baseadas no modelo cliente/servidor, os valores dos *ports* usados pelo programa servidor são predeterminados, enquanto o programa cliente pode usar valores quaisquer para esse identificador. Por exemplo, na aplicação Telnet, que usa o protocolo de transporte TCP, o programa servidor utiliza o *port* 23.

A Figura 3 ilustra uma rede TCP/IP com várias aplicações em execução; todas elas envolvendo os programas servidores que rodam em um mesmo equipamento:

FIGURA 3 – ENDEREÇAMENTO PARA AS APLICAÇÕES



Fonte: (CAR1994)

Do ponto de vista do servidor, tem-se a seguinte tabela que relaciona os fluxos de dados entre os programas:

Aplicação	Endereço Local	Endereço Remoto
WWW	200.246.93.1, 80 TCP	142.1.1.7, 1171 TCP
FTP	200.246.93.1, 21 TCP	192.1.1.9, 1207 TCP
DNS	200.246.93.1, 53 UDP	103.1.1.1, 1741 UDP

Para as aplicações padrão existentes na Internet são reservados alguns valores de *port* combinados a um dos protocolos de transporte:

Aplicação ou serviço	Número do port	Protocolo
FTP	21	TCP
Telnet	23	TCP
SMTP (e-mail)	25	TCP
DNS	53	UDP
HTTP (WWW)	80	TCP
POP	110	TCP

2.2.3.2 O PROTOCOLO TCP

O TCP é um protocolo da camada de transporte orientado à conexão que fornece um serviço confiável de transferência de dados fim-a-fim, implementando mecanismos de recuperação de dados perdidos, danificados ou recebidos fora de sequência e minimizando o atraso de trânsito para transmissão dos dados. O TCP assume que pode obter um serviço de datagrama simples e, potencialmente, não-confiável dos protocolos das camadas inferiores (CAR1994).

Conforme (CAR1994), a descrição do TCP especifica as seguintes características do protocolo:

a) o formato das mensagens que as duas estações trocam entre si;
b) os procedimentos que as estações utilizam para assegurar que os dados cheguem corretamente, recuperando erros de perda ou duplicação de mensagens;
c) o modo pelo qual o protocolo faz a distinção entre múltiplas destinações em uma determinada estação;
d) o modo pelo qual as duas estações devem iniciar uma transferência e, também, entrar em acordo sobre quando a comunicação está completa.

A unidade de transferência de dados trocada entre estações usando o protocolo TCP é chamada de segmento. Esse segmento é formado por dados oriundos da camada de aplicação, aos quais são adicionadas as informações necessárias para execução do protocolo (CHI1999). Dessa forma, um segmento TCP trafega por uma Internet TCP/IP encapsulado em datagramas IP.

O TCP é usado nas seguintes situações:

a) quando as aplicações necessitam dos mecanismos que controlam o tráfego de dados entre os *sockets*;
b) quando os dados de uma aplicação são essenciais, necessitando verificar se os dados chegaram ao destino corretamente;
c) quando a qualidade e a eficiência da rede não são elevadas, podendo ocasionar erros na transmissão.

2.2.3.3 O PROTOCOLO UDP

O UDP é um protocolo de transporte pertencente à família de protocolos TPC/IP que fornece um serviço mínimo de transporte em redes que usam o protocolo IP, permitindo que as aplicações tenham acesso direto aos serviços da camada de rede. O tipo de serviço fornecido pelo UDP tem as seguintes características:

a) não orientado à conexão;
b) transmissão não confiável.

O fato de ser não orientado à conexão significa que um datagrama pode ser enviado a qualquer momento, sem qualquer tipo de aviso, negociação ou preparação com o *host* destino. É esperado que o *host* de destino esteja preparado para receber e processar os datagramas.

O fato do protocolo não ser confiável significa que:

a) não existe nenhuma garantia de que os datagramas sejam entregues no destino;
b) não existe registro dos datagramas enviados;
c) os datagramas podem chegar fora de ordem;
d) os datagramas podem chegar duplicados;
e) não existe controle de fluxo;
f) não existe controle de congestionamento na rede.

Deste modo, compete às aplicações que usam o UDP, implementar mecanismos de detecção e correção de erros, de modo a garantir que os datagramas cheguem corretamente ao destino.

Embora as características do serviço prestado pelo UDP sejam as mesmas que as da

camada IP, este vem acrescentar dois serviços que a camada IP não disponibiliza:

a) a capacidade de distinguir um dentre os vários processos que estejam a usar os serviços da camada de rede num mesmo *host* (multiplexação/demultiplexação);
b) a capacidade de verificação da exatidão dos dados recebidos.

O UDP é usado nas seguintes situações:

a) quando as aplicações não necessitam do nível de serviço que o TCP fornece; o uso do UDP diminui a complexidade no processamento dos datagramas;
b) quando os dados de uma aplicação não são essenciais, como por exemplo um *query* DNS. Se a resposta não chegou volta-se a perguntar;
c) quando a qualidade e a eficiência da rede é elevada.

2.2.4 CAMADA DE APLICAÇÃO

A camada de aplicação consiste nas aplicações e processos que utilizam a rede. Qualquer usuário pode criar suas aplicações, pois o TCP/IP é uma arquitetura aberta, ou seja, não existe um padrão que defina como deve ser estruturada uma aplicação (CYC2000).

Segundo (REG2000), o nível mais alto desta arquitetura tem por função efetuar a interface com as aplicações que desejam utilizar os meios de comunicação de dados. A arquitetura provê aplicações padronizadas para as principais tarefas do segmento de comunicação de dados.

Exemplos de protocolos de aplicação:

a) **Telnet**: permite ao usuário de um sistema acessar remotamente outro sistema através de uma sessão de terminal;
b) **File Transfer Protocol (FTP)**: provê serviços de transferência, renomeação e remoção de arquivos e diretórios remotamente;
c) **Simple Mail Transfer Protocol (SMTP)**: provê uma interface com o usuário para enviar e receber mensagens;
d) **HTTP (HyperText Transfer Protocol)**: implementa o serviço World Wide Web que provê uma interface (normalmente um Web Browser) para visualização de páginas Web, em formato HTML;

e) **SNMP (Simple Network Management Protocol)**: desenvolvido para utilização em um modelo cliente/servidor de aplicações de gerência, cabendo ao programa gerente o papel de cliente; e aos agentes, o de servidor, caracterizando-se pela implementação das ações de gerenciamento baseada em consultas e atualizações de variáveis (objetos) das entidades gerenciadas;

f) **POP (Post Office Protocol)**: utilizado para receber mensagens de correio eletrônico, permitindo ao usuário acessar a caixa postal onde suas mensagens foram entregues, copiando-as para sua máquina.

3 MULTIMEDIA NETWORKING

As redes de computadores foram desenvolvidas para conectar computadores em diferentes locais com o intuito de estabelecer comunicação e compartilhar dados. Inicialmente, a maioria dos dados que trafegavam nas redes era textual; hoje, com o avanço da multimídia e das tecnologias de rede, transmitir informação multimídia vem tornando-se um aspecto indispensável na Internet (LIU2000).

Segundo (ROS2001), os últimos anos testemunharam um explosivo avanço no desenvolvimento de aplicações de rede que transmitem e recebem conteúdo multimídia pela Internet. Novas aplicações, também conhecidas como aplicações de mídia contínua, como entretenimento, telefonia IP, rádio pela Internet, sites multimídia, teleconferência, jogos interativos, mundos virtuais, ensino à distância e outros, são anunciadas diariamente.

Conforme (LIU2000), para os desenvolvedores, *multimedia networking* significa desenvolver uma infra-estrutura de hardware e software para suportar o transporte de informação multimídia na rede, podendo esta encontrar-se no contexto de uma rede local ou na Internet.

3.1 APLICAÇÕES MULTIMÍDIA DE REDE

Os esforços exigidos para aplicações desse tipo diferem significantemente das tradicionais aplicações orientadas à dado como Web (texto/imagem), e-mail, FTP, etc. Em particular, aplicações multimídia possuem as seguintes características que as diferem das demais aplicações de rede:

a) **considerações com o tempo**: aplicações multimídia são altamente sensíveis a atrasos (*delay sensitive*) na transmissão e às variações que podem ocorrer nesses atrasos (*interarrival jitter*);
b) **tolerância à perda**: ocasionais perdas podem apenas causar ocasionais falhas na exibição do vídeo e tais perdas podem ser parcial ou totalmente camufladas;

Estas diferenças sugerem que uma arquitetura de rede projetada inicialmente para comunicação confiável de dados possa não ser adequada para suportar aplicações multimídia. Assim, inúmeros esforços têm sido feitos para permitir que a arquitetura Internet possa oferecer suporte aos serviços exigidos por esse novo tipo de aplicação (ROS2001).

Segundo (TEC2001), prover serviços de vídeo consiste em um sistema especializado de distribuição multimídia cujo propósito é a coleção, armazenamento, distribuição e apresentação de imagens em movimento. Existem hoje duas técnicas utilizadas para fornecer serviços de vídeo:

a) **download and play**: esta técnica requer que o arquivo de vídeo seja completamente transferido para o cliente antes de ser usado ou visualizado. Vídeos podem ser transferidos como arquivos binários ou através de mensagens de e-mail. O tempo necessário para a transferência de grandes arquivos pode ser o principal aspecto a ser considerado quando o usuário está esperando para iniciar a visualização, porém, o armazenamento local do mesmo permite ao usuário visualizar o vídeo sempre que desejar. Os principais formados de arquivo de vídeo usados são AVI ou ActiveMovie para Windows e QuickTime para Mac ou Windows;
b) **streaming**: nesta técnica o sinal de vídeo é transmitido ao cliente e sua apresentação inicia-se após uma momentânea espera para armazenamento dos dados em um *buffer*[4]. Nesta forma de transmitir vídeo não é preciso fazer o download prévio do arquivo, o micro vai recebendo as informações continuamente enquanto mostra ao usuário. Esta técnica reduz o tempo de início da exibição e também elimina a necessidade de armazenamento local do arquivo. Transmissões eficazes desses sinais de vídeo através de redes com baixa largura de banda requerem uma alta taxa de compressão de dados para garantir a qualidade visual da apresentação. A técnica de compressão mais comum atualmente é conhecida por MPEG (*Motion Pictures Experts Group*).

A Internet possui hoje uma grande variedade de aplicações multimídia das quais, três estão voltadas à transmissão de vídeo em tempo real na rede.

[4] Um pequeno armazenamento prévio do vídeo que será mostrado em seguida. Ocorre no início do *streaming* ou quando a transmissão é interrompida devido algum congestionamento na rede.

3.1.1 STREAMING DE VÍDEO ARMAZENADO

Neste tipo de aplicação, clientes requisitam arquivos de vídeo que estão armazenados em servidores. Nesse caso, o conteúdo multimídia foi pré-gravado e armazenado em um servidor. Como resultado, o usuário pode controlar o vídeo mostrado à distância com funções similares às disponíveis em um videocassete.

Para esse tipo de aplicação, a transmissão de conteúdo multimídia só acontecerá sob a demanda do cliente, podendo existir vários clientes conectados ao servidor simultaneamente; cada um visualizando um conteúdo diferente.

3.1.2 STREAMING DE VÍDEO AO VIVO

Este tipo de aplicação é similar à tradicional transmissão de rádio e televisão, conhecida como *broadcast*, onde o cliente assume uma posição passiva e não controla quando o *stream*[5] começa ou termina. A única diferença está no fato dessa transmissão ser feita através da Internet. Tais aplicações permitem ao usuário receber um sinal de rádio ou televisão ao vivo que foi emitido de qualquer parte do mundo. Neste caso, como o *streaming* de vídeo não é armazenado em um servidor, o cliente não pode controlar a exibição da mídia.

Neste tipo de transmissão podem existir muitos clientes recebendo o mesmo conteúdo simultaneamente com a sua distribuição ocorrendo de duas formas:

a) **unicast**: é uma conexão ponto-a-ponto entre o cliente e o servidor, onde cada cliente recebe seu próprio *stream* do servidor. Dessa forma, cada usuário conectado ao *stream* tem sua própria conexão e os dados vêm diretamente do servidor;
b) **multicast**: ocorre quando o conteúdo é transmitido sobre uma rede com suporte à *multicast*, onde todos os clientes na rede compartilham o mesmo *stream*. Assim, preserva-se largura de banda, podendo ser extremamente útil para redes locais com baixa largura de banda.

[5] fluxo contínuo que permite ao usuário visualizar o conteúdo multimídia imediatamente, antes que o arquivo seja completamente transferido.

3.1.3 VÍDEO INTERATIVO EM TEMPO REAL

Esse tipo de aplicação permite às pessoas utilizar áudio e vídeo para comunicar-se em tempo real. Como exemplos de aplicações interativas em tempo real temos softwares de telefonia e vídeo conferência na Internet, onde dois ou mais usuários podem se comunicar oral e visualmente.

Segundo (FLU1995), aplicações desse tipo envolvem muitos indivíduos ou grupos de indivíduos em uma espécie de diálogo. O objetivo é não manter uma simples conversa bilateral, mas suportar uma reunião entre dois ou mais participantes remotamente.

3.2 STREAMING DE VÍDEO NA INTERNET

De acordo com (LIU2000), a Internet não é naturalmente adequada à transmissão de informação em tempo real. Para executar multimídia sobre a Internet, muitas questões precisam ser respondidas:

a) multimídia indica intenso tráfego de dados. O hardware atual não oferece largura de banda suficiente;
b) aplicações multimídia estão geralmente relacionadas com *multicast*, ou seja, o mesmo fluxo de dados, não múltiplas cópias, é enviado a um grupo de receptores. Por exemplo, uma transmissão de vídeo ao vivo pode ser enviada a milhares de clientes. Os protocolos desenvolvidos para aplicações multimídia precisam considerar o *multicast* para reduzir o tráfego;
c) o preço para agregar recursos de rede aos atualmente existentes torna-se impraticável. Aplicações em tempo real requerem largura de banda, então, precisa haver alguns mecanismos para que essas aplicações reservem os recursos necessários ao longo da rota de transmissão;
d) a Internet é uma rede de transmissão de pacotes que são encaminhados independentemente através de redes compartilhadas. As tecnologias atuais não podem garantir que dados de tempo real não irão encontrar seu destino sem serem desordenados. Alguns novos protocolos de transporte precisam ser usados para garantir que os dados de áudio e vídeo sejam mostrados continuamente, na ordem correta e em sincronismo;
e) é necessário que existam algumas operações padrão para as aplicações gerenciarem o transporte e apresentação de dados multimídia.

Segundo (ROS2001), existem hoje inúmeras discussões sobre como a Internet poderia atuar para melhor acomodar o tráfego multimídia com todas as suas implicações. Por um lado, alguns pesquisadores argumentam que não é necessário fazer modificações nos serviços e protocolos atuais, mas sim, adicionar mais largura de banda às conexões. Opostos a esse ponto de vista, outros defendem a idéia que a adição de largura de banda pode ser dispendioso e, em breve surgiriam novas aplicações multimídia que consumiriam essa banda extra.

Enfocando outra abordagem, alguns pesquisadores aconselham mudanças fundamentais na Internet, de forma que as aplicações possam reservar explicitamente largura de banda em uma conexão. Tais estudiosos acreditam que, se as aplicações indicarem o tráfego que pretendem realizar durante a comunicação, ao mesmo tempo em que a rede efetuar o policiamento dessas conexões a fim de verificar seu consumo real, o aproveitamento de largura da banda existente será maior. Tais mecanismos, quando combinados, requerem novos e complexos softwares nos *hosts* e roteadores, assim como novos tipos de serviços.

3.3 PROTOCOLOS DE TEMPO REAL

Em seu trabalho, (LIU2000) cita que um grupo de pesquisa da IETF conhecido por *Integrated Services Working Group* desenvolveu um avançado modelo de serviço para a Internet chamado *Integrated Services* de tempo real que pode ser visto através do RFC 1633. Esses serviços irão habilitar redes IP a fornecer serviços de qualidade para aplicações multimídia.

Formado por protocolos como o RSVP (*Resource ReServation Protocol* – Protocolo de Reserva de Recursos), juntamente com o RTP (*Real-Time Transport Protocol* – Protocolo de Transporte em Tempo Real), RTCP (*Real-Time Control Protocol* – Protocolo de Controle de Tempo Real) e o RTSP (*Real-Time Streaming Protocol* – Protocolo de Fluxo Contínuo em Tempo Real), o RFC 1633 fornece fundamentos para serviços de tempo real. Sua proposta trata de permitir a configuração e o gerenciamento das aplicações tradicionais e multimídia em uma única infra-estrutura.

3.3.1 REAL-TIME STREAMING PROTOCOL (RTSP)

De acordo com (SCH1998), o RTSP é um protocolo do nível da aplicação para controle dos dados transmitidos na rede com propriedades de tempo real. Ele provê uma arquitetura para habilitar o controle da transmissão de conteúdo multimídia, como por exemplo, vídeo sob demanda.

A descrição do RTSP, definida no RFC 2326 (disponível em <http://www.ietf.org>), especifica as seguintes características para o protocolo:

a) estabelece e controla fluxos de mídia contínua como áudio e vídeo, atuando como um controle remoto de rede para servidores multimídia, permitindo ao cliente realizar operações similares àquelas disponíveis em videocassetes, como: pausa, avanço, retorno e interrupção de uma apresentação;
b) suporta as seguintes operações:
   - recuperação de mídia de um servidor: um cliente pode requisitar uma apresentação a um servidor via HTTP ou outros métodos;
   - convite de um servidor a uma conferência: um servidor de mídia pode ser "convidado" a participar de uma conferência existente, tanto para apresentar quanto para gravar o conteúdo multimídia;
   - adição de uma mídia a uma apresentação existente: particularmente para mídia ao vivo. Interessante quando o servidor deseja informar ao cliente a chegada de um novo conteúdo;
c) sua sintaxe e operação são intencionalmente similares ao HTTP/1.1 assim, a infra-estrutura existente pode ser reutilizada. Porém, introduz alguns métodos novos e seu próprio identificador de protocolo;
d) novos métodos e parâmetros podem ser facilmente acrescentados;
e) mensagens de RTSP podem ser transportadas tanto via UDP quanto TCP;
f) reutiliza mecanismos de segurança da WEB (como mecanismos de autenticação HTTP) ou mecanismos da camada de transporte e de rede;
g) o cliente pode negociar o método de transporte (TCP ou UDP) que precisar para processar um *stream* de mídia contínua;
h) se algumas características básicas estiverem desabilitadas, o cliente pode determinar o que não está implementado e apresentar ao usuário a interface mais

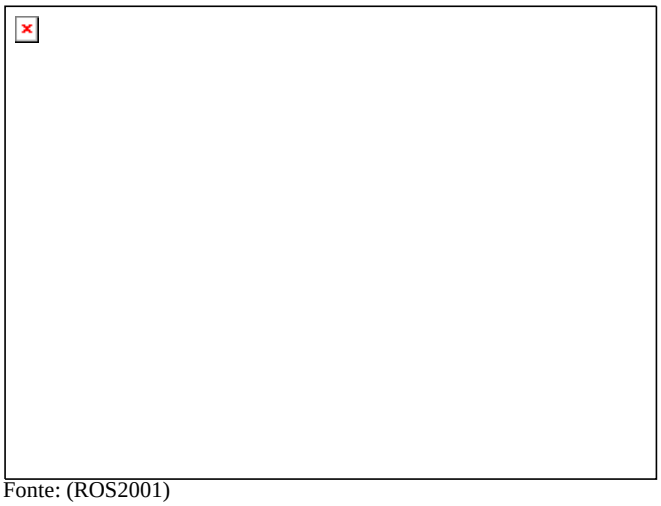
apropriada, desabilitando os mecanismos não disponíveis.

(ROS2001) apresenta em seu livro as seguintes tarefas não realizadas pelo RTSP:

a) não define métodos de compressão de áudio e vídeo;
b) não define como o sinal é encapsulado em pacotes para transmissão em uma rede. O encapsulamento para *streaming media* pode ser realizado pelo protocolo RTP ou um protocolo proprietário;
c) não restringe como a mídia será transportada. Pode ser via UDP ou TCP;
d) não restringe como um reprodutor de mídia armazena o sinal de vídeo em um *buffer*. O vídeo pode ser mostrado assim que chegar ao cliente ou com o atraso de alguns segundos ou ainda pode ser transferido totalmente antes de ser exibido.

A Figura 4 demonstra um browser requisitando uma apresentação a um servidor Web:

FIGURA 4 – INTERAÇÃO ENTRE CLIENTE E SERVIDOR USANDO RTSP



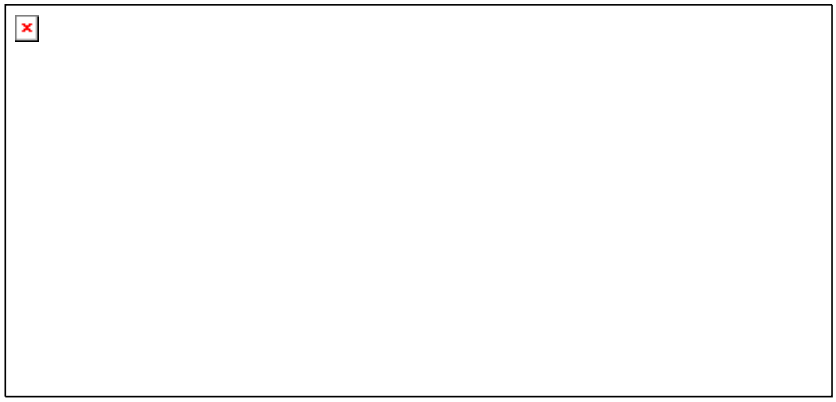
Fonte: (ROS2001)

Neste exemplo, o servidor Web encapsula a descrição do arquivo de apresentação em uma resposta HTTP e envia a mensagem ao browser solicitante. Quando o browser recebe a resposta, invoca o reprodutor de mídia (comumente conhecido por *media player*) baseado no tipo do conteúdo retornado pelo servidor Web. Em seguida, o reprodutor e servidor de mídia trocam uma série de mensagens RTSP entre si. Requisições como PLAY e PAUSE enviadas pelo reprodutor permitem ao usuário controlar a exibição da mídia.

Cada sessão iniciada com uma requisição RTSP SETUP possui um identificador fornecido pelo servidor em uma resposta RTSP SETUP. O cliente repete o identificador de sessão para cada requisição até fechar a sessão com uma requisição TEARDOWN.

Abaixo pode-se ver um exemplo simplificado de uma sessão RTSP entre o cliente (**C:**) e o servidor (**S:**).

QUADRO 1 – TROCA DE MENSAGENS RTSP ENTRE UM CLIENTE E O SERVIDOR



3.3.2 REAL-TIME PROTOCOL (RTP)

De acordo com (SCH1996), o RTP é um protocolo de transporte de rede fim-a-fim que oferece funções para aplicações transmitirem dados em tempo real, como áudio e vídeo, através de serviços de rede *unicast* ou *multicast*. O RTP não realiza reserva de recursos nem fornece garantia de qualidade (*QoS – Quality of Service*) para serviços de tempo real. O transporte de dados é acrescido por um protocolo de controle, o RTCP, que permite o monitoramento da entrega dos dados até em grandes redes *multicast* além de fornecer um serviço mínimo de identificação.

Segundo (ROS2001), o RTP pode ser utilizado para o transporte de formatos comuns como PCM ou GSM para som e MPEG1 e MPEG2 (descrito na seção 3.5) para vídeo, além de poder ser utilizado para transportar outros formatos proprietários. O RTP também é descrito na arquitetura de protocolos H.323 para uso em aplicações de vídeo interativo em tempo real como áudio e vídeo conferência.

Como já foi citado na seção 3.2, a Internet é uma rede de transmissão de pacotes que

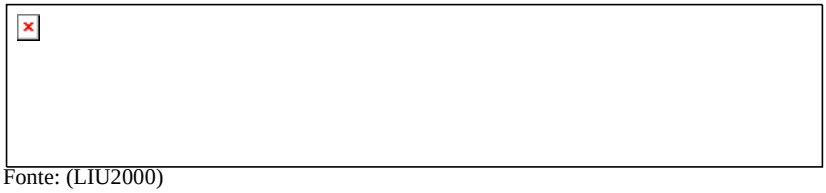
são encaminhados independentemente através de redes compartilhadas, não garantindo a chegada ao destino na mesma ordem que foram enviados. Porém, aplicações multimídia requerem apropriada temporização na transmissão e exibição dos dados. O RTP fornece registro do tempo (*timestamping*), numeração seqüencial e outros mecanismos para tratar dessas dificuldades.

Para (LIU2000), *timestamping* é a informação mais importante para aplicações de tempo real. O transmissor define o seu valor de acordo com o instante em que o primeiro *byte* da primeira amostra do pacote é gerada. Este valor é acrescido do tempo percorrido por um pacote, de acordo com o número de amostras que o formam. Depois de receber os pacotes, o receptor utiliza o *timestamping* para reconstruir a ordem de tempo original e exibir a informação na ordem correta.

O UDP não entrega os pacotes em ordem de tempo. Assim, números seqüenciais são usados para colocá-los na ordem correta, além de serem usados para detecção da perda dos mesmos.

Basicamente, o RTP é um protocolo que funciona sobre o UDP, onde partes da informação multimídia gerada por codificadores são encapsuladas em pacotes RTP. No lado da aplicação responsável pelo envio da informação, cada pacote RTP é encapsulado em segmentos UDP para então ser encaminhado às camadas inferiores da arquitetura Internet. Da parte do receptor, pacotes RTP chegam à aplicação através de *sockets* UDP. É de responsabilidade da aplicação extrair a informação multimídia dos pacotes (ROS2001).

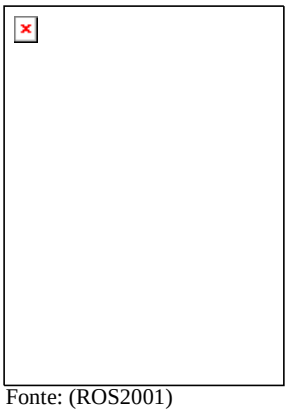
FIGURA 5 – DADO RTP EM UM PACOTE IP



Fonte: (LIU2000)

Da perspectiva do desenvolvedor, o RTP faz parte da camada de aplicação, ou seja, o encapsulamento e extração de pacotes RTP em *sockets* UDP são de responsabilidade do desenvolvedor na implementação da aplicação.

FIGURA 6 – O RTP É PARTE DA CAMADA DA APLICAÇÃO



Fonte: (ROS2001)

Em um pacote, o dado RTP é conhecido por *payload* (carga útil) e sua interpretação pela aplicação é determinada pelo campo *payload type*, que pode assumir um dos tipos padrão, definidos no RFC 1890, ou, então, utilizar um formato proprietário. A tabela abaixo descreve alguns dos formatos padrão para áudio e vídeo:

Número do payload	Formato de áudio	Formato de vídeo
3	GSM	–
9	G.722	–
14	Áudio MPEG	–
15	G.728	–
26	–	Motion JPEG
31	–	H.261
32	–	Vídeo MPEG 1
33	–	Vídeo MPEG 2

3.3.3 REAL-TIME CONTROL PROTOCOL (RTCP)

O RFC 1889 também especifica o RTCP como sendo um protocolo que aplicações multimídia de rede possam usar em conjunto com o RTP.

Segundo (ROS2001), em uma sessão RTP, pacotes RTCP são transmitidos entre os participantes daquela sessão, contendo relatórios estatísticos que podem ser úteis para a aplicação. Tais estatísticas incluem: número de pacotes enviados, número de pacotes perdidos e variações nos atrasos durante a transmissão. A especificação do RTP não indica o que a aplicação deve fazer com tais informações; esta tarefa fica a encargo do desenvolvedor.

[CT4] Comentário: não sei o que eh essa tranqueira e ela aparece um monte de vezes. Preciso de uma traducao

Para determinar os participantes de uma sessão, o RTCP pode usufruir de

mecanismos de distribuição como IP Multicast, ou, os receptores dos pacotes RTP podem manter uma tabela dos participantes, anotando as origens dos *streams*, disponíveis no campo SSRC (*synchronization source)* e as origens do *payload*, encontradas no campo CSRC (*contributing source*) contidos nos pacotes RTP recebidos.

3.3.3.1 TIPOS DE PACOTES RTCP

Para cada *stream* que um receptor recebe em uma sessão RTP, é gerado um relatório de recepção agregado a pacotes RTCP. O pacote é então enviado a todos os participantes daquela sessão (ROS2001).

Relatórios de recepção incluem diversos campos, dos quais os mais importantes são listados abaixo:

a) a origem do *stream* de RTP para o qual o relatório está sendo gerado;
b) a fração de pacotes perdidos dentro do *stream*. Cada receptor calcula o número de pacotes RTP perdidos dividido pelo número de pacotes RTP enviados como parte de um *stream*;
c) o último número seqüencial (*sequence number*) recebido no *stream* de pacotes;
d) a variação nos atrasos, que é calculada como sendo a média de tempo de chegada entre sucessivos pacotes no *stream* de RTP.

Para cada *stream* de RTP que o remetente está transmitindo, são criados e transmitidos relatórios RTCP. Esses pacotes incluem informação sobre o *stream* de RTP, incluindo:

a) a origem do *stream* de RTP;
b) o *timestamp* e tempo real do pacote RTP mais recentemente gerado no *stream*;
c) o número de pacotes enviados no *stream*;
d) o número de bytes enviados no *stream*.

Relatórios do remetente podem ser utilizados para sincronizar diferentes *streams* de mídia dentro de uma sessão RTP. Por exemplo, uma aplicação de videoconferência pode gerar dois *streams* separados, um para vídeo e outro para áudio. Os valores do *timestamp* nesses pacotes estão ligados com o momento da amostra de áudio e vídeo e não com o tempo real em que os sinais foram gerados. Como cada relatório do remetente contém o *timestamp* do pacote RTP e o tempo real em que o pacote foi criado, o receptor pode usar essa associação para

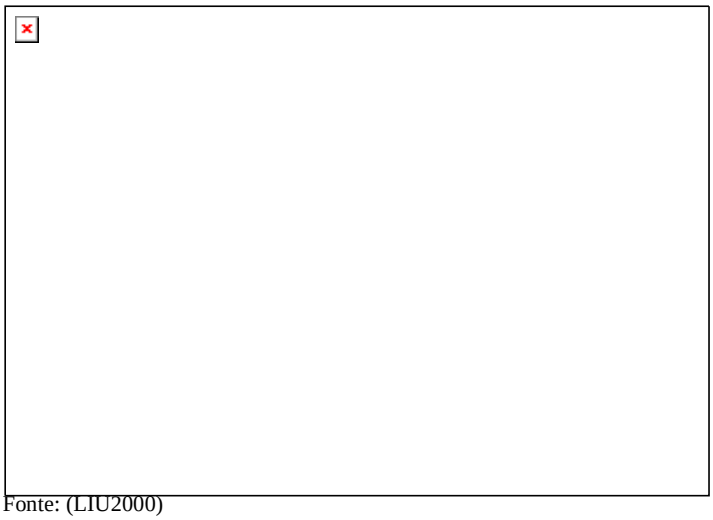
sincronizar a exibição do áudio e vídeo (ROS2001).

Para cada *stream* RTP que o remetente está transmitindo, também são criados pacotes com a descrição da origem dos mesmos. Estes pacotes contêm informações como endereço de e-mail, nome e a aplicação geradora do *stream*. Ele também inclui a origem do RTP associado (ROS2001).

3.3.4 RESOURCE RESERVATION PROTOCOL (RSVP)

RSVP é um protocolo de controle que permite aos receptores requisitar uma determinada qualidade de serviço para seu fluxo de dados. Aplicações de tempo real utilizam o RSVP para reservar a quantidade necessária de recursos e roteadores ao longo do caminho de transmissão, a fim de obter disponibilidade da largura de banda necessária para a transmissão (LIU2000). A Figura 7 ilustra o esquema base desse protocolo.

FIGURA 7 – RESERVA DE UM NÓ DA REDE NO CAMINHO DO FLUXO DE DADOS



Fonte: (LIU2000)

A seguir, são descritos os elementos que compõem o esquema da Figura 7:

a) **Policy Control**: trata das permissões da entidade que efetuou o pedido. Nomeadamente administração, autenticidade, controle de acesso, pagamento de serviços, etc.;
b) **Admission Control**: verifica os recursos físicos da rede, decidindo se o pedido pode ou não ser atendido;

c) **RSVP Daemon**: analisa os resultados dos blocos anteriores. Caso um deles falhe, envia uma notificação de erro à entidade que efetuou o pedido. Caso contrário, estabelece os parâmetros dos blocos *packet classifie*r e *packet scheduler* de forma a atender ao pedido;
d) **Packet scheduler**: ordena os vários pacotes a serem transmitidos;
e) **Packet classifier**: atribui aos vários pacotes de informação diferentes categorias de QoS correspondente.

(LIU2000) descreve as seguintes características para o RSVP:

a) fluxos RSVP são *simplex*: RSVP distingue remetentes de receptores. Mesmo que, um *host* possa atuar tanto como remetente quanto como receptor, uma reserva RSVP somente aloca recursos para *streams* em uma direção;
b) suporta *unicast* e *multicast*: desde que a reserva seja iniciada pelo receptor e seu estado seja leve, RSVP pode facilmente manipular mudanças em membros e rotas. Um *host* pode enviar mensagens IGMP (*Internet Group Management Protocol*) para inscrever-se em um grupo *multicast*;
c) o RSVP é orientado ao receptor e manipula receptores heterogêneos: em grupos heterogêneos de *multicast*, receptores têm diferentes capacidades e níveis de QoS. Os remetentes podem dividir o tráfego em muitos fluxos, cada um é um fluxo RSVP com diferente nível de QoS.
d) compatibilidade: esforços têm sido feitos para executar o RSVP tanto sobre o IPv4 (protocolo IP versão 4, atualmente em uso) quanto sobre o IPv6 (futura versão desenvolvida para o protocolo IP).

3.4 ENVIANDO MULTIMÍDIA DE UM SERVIDOR DE STREAMING PARA UMA APLICAÇÃO

Uma das formas de fornecer vídeo pela Internet se dá através dos servidores de *streaming*, que tratam de enviar sinais de vídeo para reprodutores de mídia. Este servidor pode ser proprietário, como aqueles comercializados pela RealNetworks (Real Media Server) e Microsoft (Media Server) ou um servidor de *streaming* de domínio público (ROS2001). Com um servidor de *streaming*, áudio e vídeo podem ser enviados sobre UDP (preferencialmente, em relação ao TCP) utilizando protocolos da camada de aplicação, mais adequados do que *streaming* via servidor HTTP.

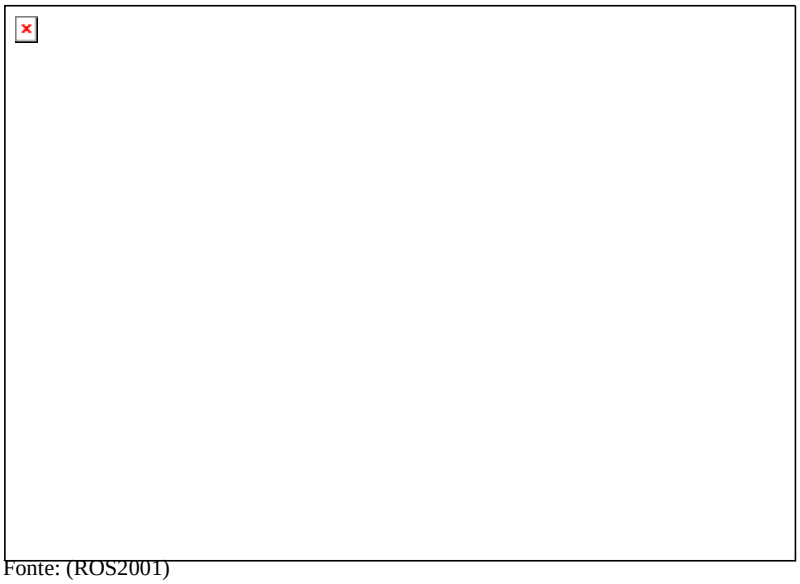
Esta arquitetura requer dois servidores, como mostra a Figura 8. Um servidor de HTTP para servir às páginas Web; e um servidor de *streaming*, para servir aos arquivos de vídeo. Os dois servidores podem executar em um mesmo sistema ou em dois sistemas distintos.

Uma simples arquitetura para servidor de *streaming* descreve os seguintes passos:

a) um navegador Web faz uma requisição HTTP para obter um arquivo de descrição da apresentação, conhecido também por arquivo *metafile*.
b) o servidor Web retorna o arquivo utilizado ao navegador para identificar o reprodutor de mídia adequado para exibir o vídeo;
c) o arquivo é repassado para o reprodutor contendo informações referentes à localização do vídeo no servidor de *streaming*;
d) o reprodutor faz a requisição do arquivo diretamente ao servidor de *streaming*. A partir deste momento, servidor e reprodutor interagem diretamente utilizando seus próprios protocolos e podem permitir uma interação maior do usuário com o *stream* de vídeo.

A arquitetura demonstrada na Figura 8 possui várias opções para entregar o vídeo do servidor para os clientes (*media players*).

FIGURA 8 – STREAMING DE UM SERVIDOR PARA UM REPRODUTOR DE MÍDIA



Fonte: (ROS2001)

Algumas delas são descritas a seguir:

a) o vídeo é transportado sobre o UDP a uma taxa constante, igual àquela com que o receptor apresenta o conteúdo (esta taxa é equivalente à taxa de codificação do vídeo);
b) similar à anterior, nesta opção o receptor atrasa a exibição por 2-5 segundos para eliminar a variação nos atrasos em pacotes do mesmo *stream*, que podem ocorrer em função da rede. Esta tarefa é realizada pelo cliente, armazenando o vídeo recebido em um *buffer*. Uma vez que o cliente guardou alguns segundos da mídia, ele começa a "descarregar" o *buffer*. Novamente, a taxa de preenchimento do *buffer* é igual à taxa com que é descarregado, exceto na ocorrência da perda de pacotes;
c) processo semelhante ao descrito, porém, neste caso, a mídia é transportada sobre o TCP. Como o TCP retransmite pacotes perdidos, desta maneira pode-se oferecer uma qualidade melhor do que com o UDP. Por outro lado, a taxa de preenchimento do *buffer* varia conforme a perda de pacotes, o que pode ocasionar um esvaziamento do *buffer*, resultando indesejáveis pausas na exibição do vídeo no cliente.

3.5 CODIFICAÇÃO E COMPRESSÃO DE VÍDEO

Segundo (ROS2001), antes da transmissão do vídeo através de um computador na rede, ele precisa ser digitalizado e comprimido. A necessidade por digitalização é óbvia: os computadores transmitem bits na rede, assim, toda informação precisa ser representada como uma seqüência de bits. A compressão é importante porque o vídeo não comprimido consome uma quantidade muito grande de armazenamento e largura de banda.

Um vídeo é uma seqüência de imagens, normalmente exibidas à uma taxa constante, por exemplo a 24-30 imagens por segundo. Uma imagem digital não comprimida consiste em uma matriz de pontos, sendo cada ponto codificado em um número de bits que representam luminosidade e cor (ROS2001).

Existem dois tipos de redundância em vídeo, os quais podem ser explorados para compressão:

a) **espacial**: existente na imagem fornecida. Por exemplo, uma imagem que

consiste de muitos espaços em branco pode ser eficientemente comprimida;

b) **temporal**: consiste na repetição da imagem numa imagem subseqüente. Se por exemplo, uma imagem e a subseqüente imagem forem exatamente iguais, não há razão para re-codificar a imagem, é mais eficiente simplesmente indicar durante a codificação que a imagem subseqüente é igual a anterior.

O padrão de compressão conhecido por MPEG (*Motion Pictures Experts Group*) é, sem dúvida, a técnica de compressão mais popular. Este inclui o MPEG 1 para vídeo com qualidade de CD-ROM (1.5 Mbps), MPEG 2 para vídeo com qualidade de DVD (3-6 Mbps), e MPEG 4 para compressão de vídeo orientada a objetos. O padrão de compressão H.261 que faz parte da arquitetura de protocolos H.323 também é muito popular na Internet. Existem ainda inúmeros padrões de compressão proprietários.

4 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a realização do protótipo de software apresentado neste trabalho, foi necessário o estudo de algumas técnicas e ferramentas que serviram de apoio à especificação e implementação, visando facilitar seu desenvolvimento e compreensão. A seguir são apresentadas algumas considerações sobre as técnicas e ferramentas utilizadas.

4.1 ORIENTAÇÃO A OBJETOS

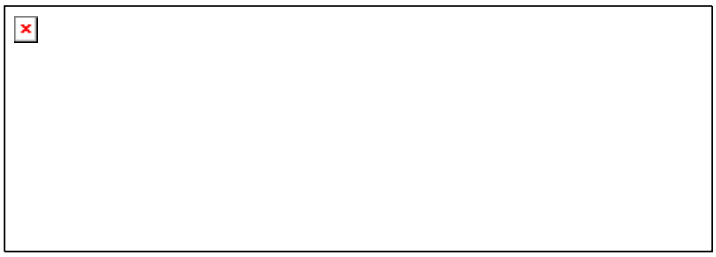
(FUR1998) destaca em seu livro que Rumbaugh define orientação à objetos como "uma nova maneira de pensar os problemas utilizando modelos organizados a partir de conceitos do mundo real. O componente fundamental é o objeto que combina estrutura e comportamento em uma única entidade".

Essa tecnologia oferece modularidade de seus elementos, sendo que uma aplicação no universo dos objetos consiste de um conjunto de blocos de construção autocontidos e predefinidos que podem ser localizados, reparados ou substituídos.

Quando se utiliza a orientação à objetos na especificação e implementação de um software deve-se tomar conhecimento de alguns conceitos, dos quais, destacam-se o do próprio objeto e o das classes de objetos.

Segundo (FUR1998), um objeto é uma ocorrência específica (instância) de uma classe e esta, por sua vez, consiste de uma estrutura que define atributos e operações que controlam o estado e o comportamento dos objetos.

FIGURA 9 – EXEMPLO DE CLASSE E OBJETO



Fonte: adaptação de (FUR1998)

Outros conceitos relacionados a orientação à objetos são descritos a seguir:

a) **atributos**: de acordo com (COAD1992), um atributo consiste em dados

(informações de estado) através dos quais cada objeto em uma classe tem seu próprio valor, ou seja, representa a característica do objeto;

b) **serviços**: especifica a maneira pela qual os dados de um objeto são manipulados. São os procedimentos, operações que um objeto deve possuir para realizar sua finalidade;
c) **mensagens**: sinal enviado de um objeto a outro requisitando um serviço através da execução de uma operação;
d) **herança**: permite que uma nova classe (subclasse) seja descrita a partir de outra classe (superclasse) já existente, herdando as características e o comportamento da superclasse e podendo adicionar novas características e comportamentos aos herdados;
e) **encapsulamento**: combinação de atributos e serviços em uma classe (FUR1998);
f) **polimorfismo**: segundo (FUR1998), consiste da habilidade para usar a mesma mensagem para invocar comportamentos diferentes do objeto.

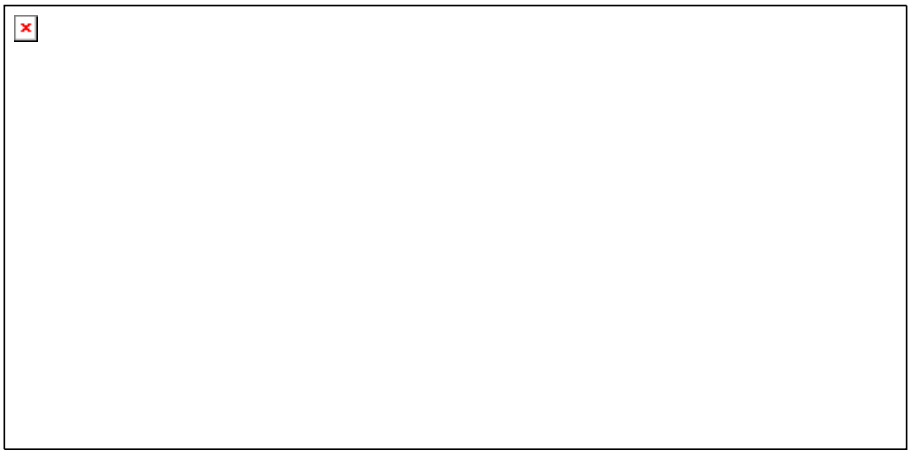
4.2 LINGUAGEM UNIFICADA DE MODELAGEM – UML

Segundo (FUR1998), a UML (*Unified Modeling Language*) é uma linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação.

A UML, chamada a princípio de Método Unificado, foi apresentada ao público pela primeira vez em outubro de 1995, como resultado da combinação da notação de outros três métodos: Booch, OMT (*Object Modeling Thecnique*) e OOSE (*Object-Oriented Software Engineering*). Nesse instante, seus autores estavam motivados a criar uma linguagem de modelagem unificada que se tornasse poderosa o suficiente para modelar qualquer tipo de aplicação de tempo real, cliente/servidor ou outros tipos de softwares padrões (FUR1998).

Novos métodos foram agregados e parceiros importantes juntaram-se à esse esforço produzindo, em setembro de 1997, a UML versão 1.1, cujo aprovação, aconteceu em novembro do mesmo ano pela OMG (*Object Management Group*) como sendo uma linguagem de modelagem não proprietária bem definida, expressiva, poderosa e geralmente aplicável.

FIGURA 10 – EVOLUÇÃO DA UML



Fonte: (FUR1998)

Segundo (FUR1998), a UML pode ser usada para:

a) mostrar as fronteiras de um sistema e suas funções principais utilizando atores e casos de uso;
b) ilustrar a realização de casos de uso com diagramas de interação;
c) representar uma estrutura estática de um sistema utilizando diagramas de classe;
d) modelar o comportamento de objetos com diagramas de transição de estado;
e) revelar a arquitetura de implementação física com diagramas de componente e de implantação;
f) estender sua funcionalidade através de estereótipos.

4.3 WINDOWS SOCKETS API

Para comunicação com os protocolos de rede foi preciso utilizar uma interface de *socket* conhecida por Windows Socket API (*Aplication Program Interface*), fornecida pelo sistema operacional Windows. Segundo (COM1997), essa interface fornece funções gerais que provêm a comunicação de rede, usando os protocolos disponíveis. As funções de *socket* referem-se a todos os protocolos componentes da arquitetura TCP/IP como uma família única, permitindo ao programador especificar o tipo de serviço requerido bem como o nome de um protocolo específico.

A Windows Sockets API (comumente conhecida por WINSOCK) é encontrada na biblioteca *winsock.dll* e consiste em uma adaptação das primitivas *socket* do UNIX BSD para o Microsoft Windows.

(REG2000) especifica as seguintes características para a biblioteca *Winsock*:

a) suporte a vários protocolos, viabilizando inclusive comunicação multimídia;
b) serviços para endereçamento em vários domínios: DNS, SAP, X.500, etc.;
c) possibilidade de sobreposição das funções *send* e *receive*;
d) configuração da qualidade do serviço, permitindo variação da largura de banda, tempos de latência e prioridade;
e) permite aos usuários utilizarem capacidades multiponto e multicamada específicos do protocolo de transporte disponível.

4.4 A BIBLIOTECA JRTPLIB

Segundo (LIE2000), *JRTPLib* é uma biblioteca orientada à objetos escrita em C++ que oferece suporte para o protocolo RTP. Seu uso permite o envio e recepção de pacotes RTP enquanto que, as funções do RTCP são realizadas internamente pela biblioteca.

Conforme descreve (LIE2000), a biblioteca suporta, entre outras, as plataformas GNU/Linux e MS-Windows 95/98 e NT.

Durante a definição das técnicas e ferramentas que seriam utilizadas no desenvolvimento do protótipo, analisou-se também outra biblioteca que implementa as especificações do protocolo RTP, conhecida por *RTPLib*.

A referida biblioteca foi desenvolvida em C++ pela *Lucent Technologies* e está disponível em: <www-out.bell-labs.com\project\RTPlib\default.html>.

A biblioteca *JRTPLib* foi escolhida principalmente pelo fato de sua implementação ter como base a orientação a objetos.

4.5 PROGRAMAÇÃO VISUAL (AMBIENTE VISUAL C++)

Segundo (HOL1999), o Visual C++ é um ambiente de desenvolvimento extremamente poderoso, contido em um pacote formado por inúmeras ferramentas para

auxiliar na programação Windows. Também conhecido por Microsoft Visual Studio, integra ferramentas para criação, edição, compilação e teste de software em uma única interface.

Para o desenvolvimento dos programas utilizou-se a linguagem C++, originalmente criada para desenvolver programas grandes, tornando o processo mais gerenciável, ou seja, facilitando o tratamento da programação Windows. Além disso, com a migração da linguagem C para a linguangem C++, a metodologia de programação procedural foi substituída pela orientação a objetos.

O ambiente dispõe da biblioteca Microsoft Foundation Class (MFC), um pacote de código pré-escrito e pronto para uso (HOL1999). Essa biblioteca fornece uma coleção de classes escritas em C++ que realizam a maioria do trabalho penoso no desenvolvimento de aplicações Windows.

O Visual C++ também oferece muitas ferramentas de programação, tais como o editor de menu para a criação de menus, o editor de diálogo para a criação de caixas de diálogo e os Assistentes do Visual C++, que escrevem boa parte do programa apenas com base em algumas informações fornecidas nas etapas de execução da ferramenta (HOL1999).

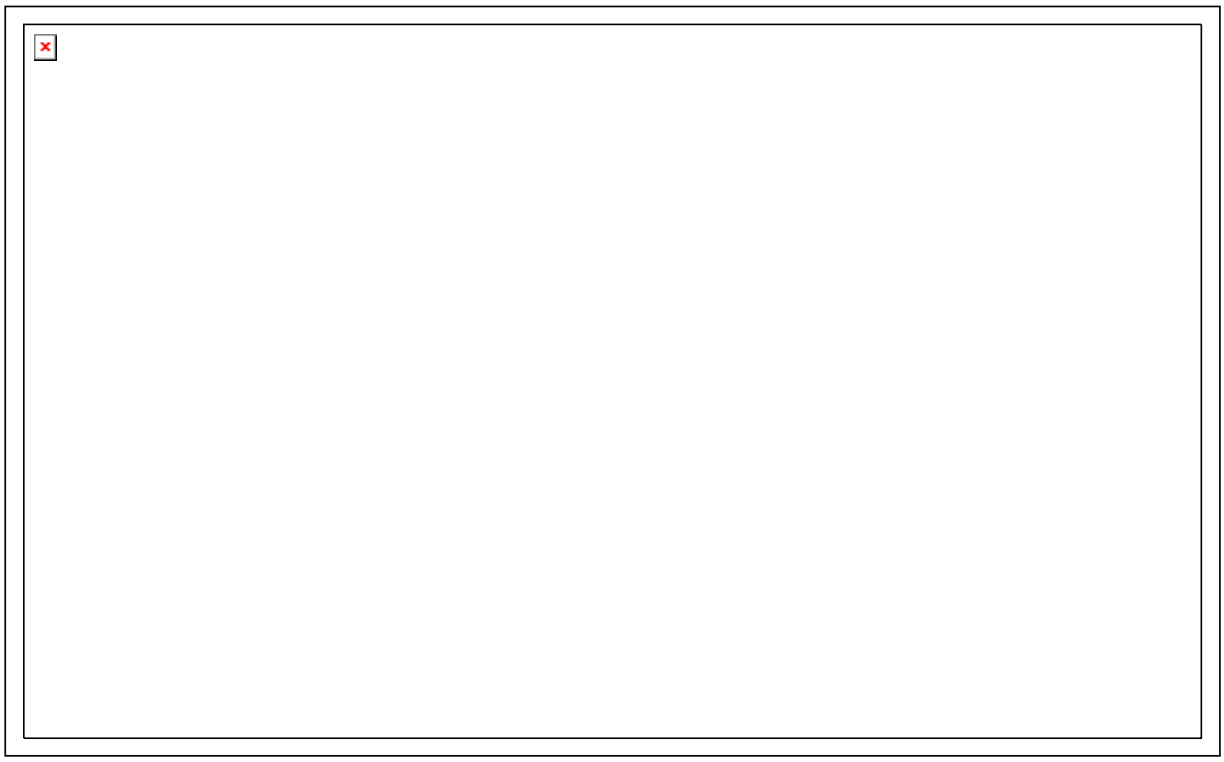
Os projetos em desenvolvimento são colocados em espaços de trabalho (*workspaces*), sendo que um espaço de trabalho pode conter vários projetos. A grande quantidade de arquivos que um programa Windows utiliza nos projetos são organizados automaticamente pelo ambiente.

O trabalho foi desenvolvido utilizando a versão 6.0 do ambiente Microsoft Visual Studio e versão 4.2 da biblioteca MFC.

4.6 FERRAMENTA CASE RATIONAL ROSE

O *Rational Rose*, utilizado para especificar o protótipo, é uma ferramenta CASE (*Computer Aided Systems Engineering*) que permite especificar, visualizar, documentar e construir artefatos de um sistema utilizando os diagramas que fazem parte da UML (RAT2001). A Figura 11 mostra a ferramenta sendo utilizada durante a especificação do protótipo.

FIGURA 11 – FERRAMENTA RATIONAL ROSE



4.7 HARDWARE PARA CAPTURA DO SINAL DE TV

Para realizar a captura do sinal externo de TV, foi necessário utilizar uma placa específica para esse fim, capaz de captar e converter um sinal convencional de TV, geralmente analógico, para o formato digital.

A placa utilizada é comercializada sob o nome de *"Pixel View – Play TV pro"*, e apresenta basicamente as seguintes características:

a) oferece suporte para captura de sinais de vídeo, TV e rádio;
b) realiza a captura de vídeo à uma taxa máxima de 30 amostras/segundo, possibilitando ao usuário utilizar taxas menores. O padrão é de apenas 15 amostras/segundo;
c) trabalha no padrão de cores RBG fornecendo opções para os formatos de 16, 24 e 32 *bits* de cores;
d) permite ao usuário configurar o tamanho da janela de exibição, podendo esta variar, em largura e altura, de 160x120 pontos até o tamanho da tela.

Para captar o sinal fornecido pela placa de vídeo, foram utilizadas APIs do Windows que fornecem acesso ao seu *driver*, possibilitando assim, transmitir o sinal de TV pela rede utilizando a técnica de *streaming media*.

O uso de APIs permite que o software funcione igualmente bem com outras marcas e/ou modelos de placas para captura de vídeo, desde que, permita ao usuário trabalhar no padrão de cores RGB (16 *bits*) com uma janela de visualização de 160x120 pontos. Essa configuração se faz necessária para o correto funcionamento do software desenvolvido.

5 DESENVOLVIMENTO DO PROTÓTIPO DE SOFTWARE

Para o desenvolvimento e implementação do protótipo de software, observou-se a necessidade de se criar um sistema que possibilitasse a captura e transmissão de sinais de vídeo pela Internet. A transmissão de sinais de áudio não será contemplado pelo protótipo de software proposto, porém, tal característica pode ser agregada ao protótipo em trabalhos futuros.

A arquitetura cliente/servidor foi escolhida como sendo a solução mais apropriada. De um lado, um servidor aguarda conexões e transmite os sinais de vídeo a um ou mais clientes. De outro, clientes estabelecem conexão e requisitam início e término da recepção.

Os sinais de vídeo capturados são encapsulados em pacotes RTP e transmitidos utilizando-se a biblioteca *JRTPLib*, descrita na seção 4.4.

O estabelecimento das conexões, bem como início e término da exibição de vídeo no cliente é realizado pela troca de mensagens RTSP entre o cliente e o servidor. Um modelo de servidor RTSP foi proposto incluindo as seguintes características:

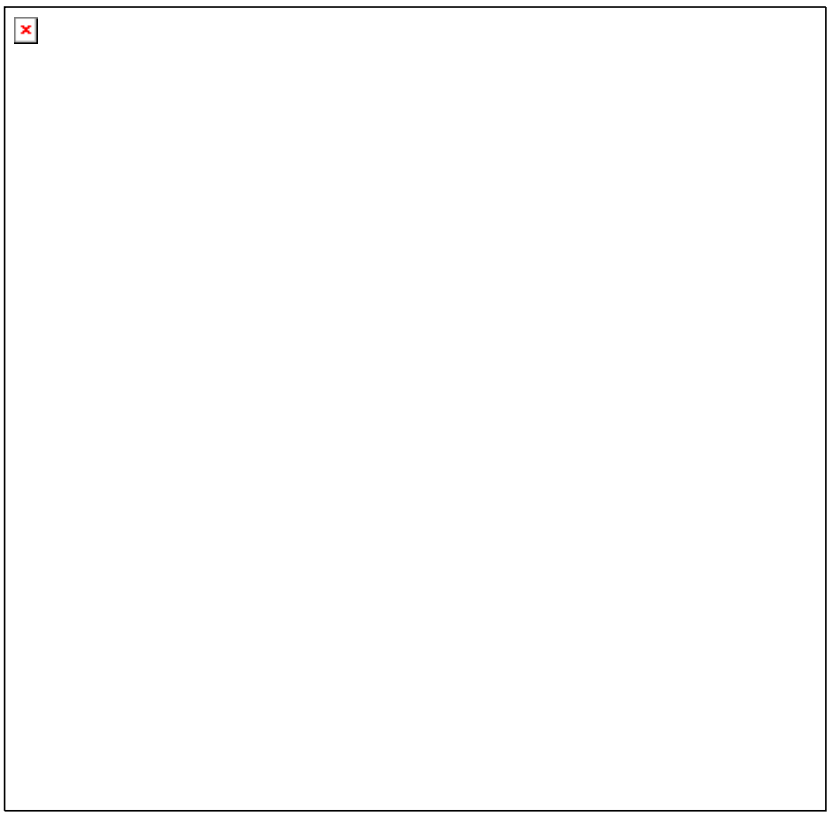
a) implementação mínima do protocolo necessária para viabilizar a comunicação com o cliente, realizando o tratamento de requisições SETUP, PLAY, PAUSE e TEARDOWN;
b) verificação da sintaxe das requisições com envio de respostas ao cliente indicando sua aceitação ou não;
c) transmissão do sinal de vídeo a um ou mais clientes via *unicast*;
d) exibição de algumas informações referentes às conexões e trocas de mensagens RTSP e RTP entre o servidor e os clientes.

O software cliente permite ao usuário estabelecer conexão com um servidor e controlar a exibição do vídeo, dispondo de comandos para iniciar, dar pausa e parar a reprodução.

5.1 ESPECIFICAÇÃO DO PROTÓTIPO

Para demonstrar o funcionamento do protótipo, o esquema apresentado na Figura 12 dá uma visão geral de sua arquitetura e como são realizadas as trocas de mensagens entre cliente e servidor durante a conexão.

FIGURA 12 – VISÃO GERAL DA ARQUITETURA DO PROTÓTIPO

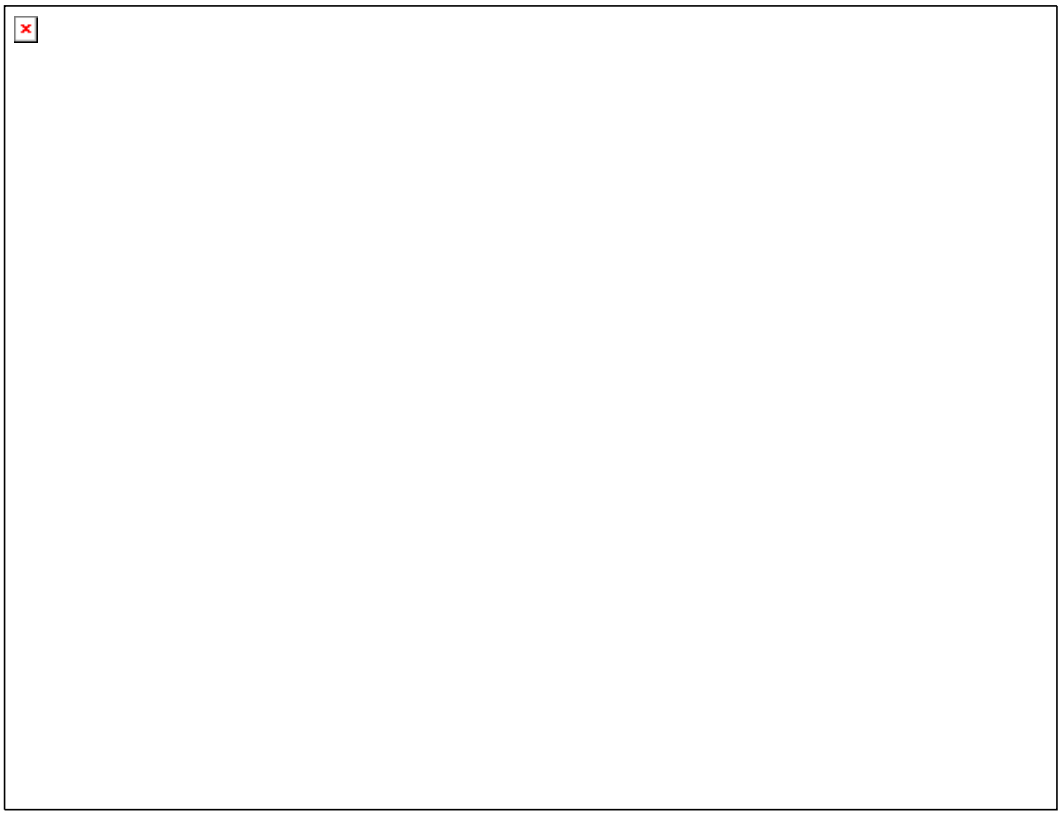


A especificação do protótipo foi realizada utilizando o diagrama de caso de uso, o diagrama de classes e o diagrama de seqüência.

5.1.1 DIAGRAMA DE CASO DE USO

A Figura 13 demonstra o diagrama de caso de uso do protótipo, onde pode-se visualizar o papel do servidor, do *driver* de captura e do cliente. Os casos de usos descrevem a interação que ocorre entre os atores.

FIGURA 13 – DIAGRAMA DE CASO DE USO



O Quadro 2 detalha os casos de uso, trazendo informações como seu número, nome, ator que inicia a ação e descrição.

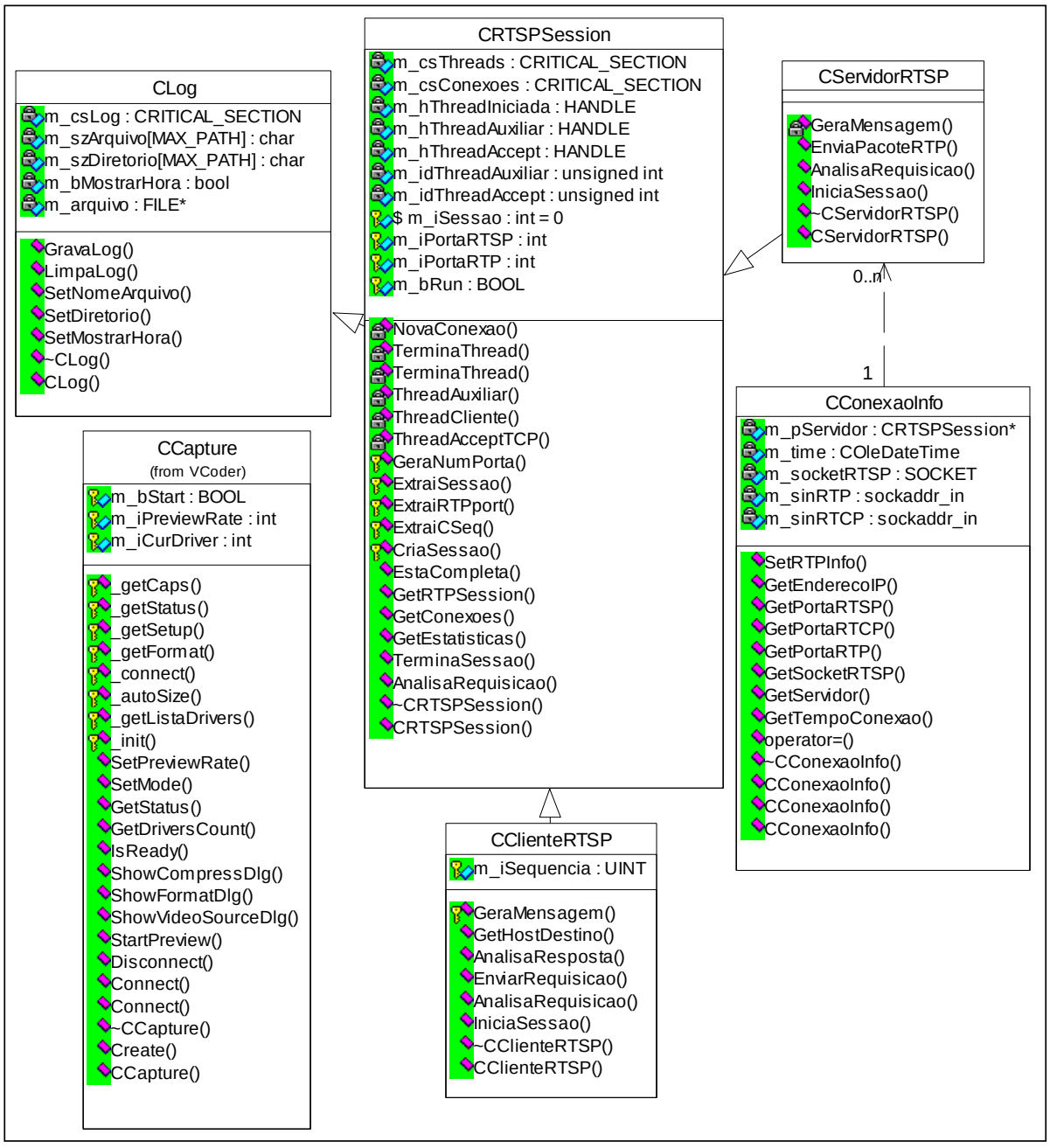
QUADRO 2 – DESCRIÇÃO DOS CASOS DE USO

No	***Caso de uso***	***Quem inicia a ação***	***Descrição do caso de uso***
1	Inicia sessão	Usuário	É inicializada uma sessão RTSP na porta 554. Informa ao driver de captura para iniciar a captura
2	Termina sessão	Usuário	Termina as sessões RTSP e RTP. Envia mensagens aos clientes indicando fim da conexão
3	Reproduz *streams*	CCapture	Reproduz em tela as amostras de vídeo recebidas
4	Estabelece conexão	Usuário	Inicia uma sessão RTSP em uma porta aleatória e avisa ao servidor a existência de nova conexão
5	Termina conexão	Usuário	Envia mensagem ao servidor requisitando desconexão
6	Envia requisição	Usuário	Envia uma mensagens RTSP requisitando início, pausa ou término da transmissão do *stream*
7	Recebe mensagens	Cliente	Recebe as mensagens do cliente, analisa, processa e envia uma resposta.

5.1.2 DIAGRAMA DE CLASSES

A Figura 14 mostra o diagrama de classes desenvolvido para especificar o protótipo.

FIGURA 14 – DIAGRAMA DE CLASSES



O Quadro 3 apresenta uma breve descrição das classes modeladas para o protótipo.

QUADRO 3 – DESCRIÇÃO DAS CLASSES

Classe	***Descrição da classe***
CLog	Classe base utilizada pelas subclasses para gravar mensagens em um arquivo de *log*. Essas mensagens podem ser utilizadas pelo desenvolvedor para acompanhamento do funcionamento do protótipo.
CRTSPSession	Esta classe implementa as funcionalidades do RTSP comuns ao cliente e ao servidor. É utilizada como base para as classes CClienteRTSP e CServidorRTSP.
CClienteRTSP	Implementa as funcionalidades do RTSP para o módulo cliente.
CServidorRTSP	Implementa as funcionalidades do RTSP para o módulo servidor.
CConexãoInfo	Utilizada pela classe CServidorRTSP para guardar informações dos clientes conectados, como: horário em que a conexão foi estabelecida, portas RTSP, RTP e RTCP.
CCapture	A função desta classe é realizar a captura do sinal externo de TV, através da interação com o *driver* da placa de captura de vídeo; bem como o envio do *stream* à classe CRTSPSession que realiza a sua transmissão aos clientes conectados via RTP.

5.1.3 DIAGRAMAS DE SEQÜÊNCIA

As figuras a seguir demonstram os diagramas de seqüência do protótipo os quais apresentam componentes do sistema colaborando entre si.

FIGURA 15 – DIAGRAMA DE SEQÜÊNCIA INICIA SESSÃO

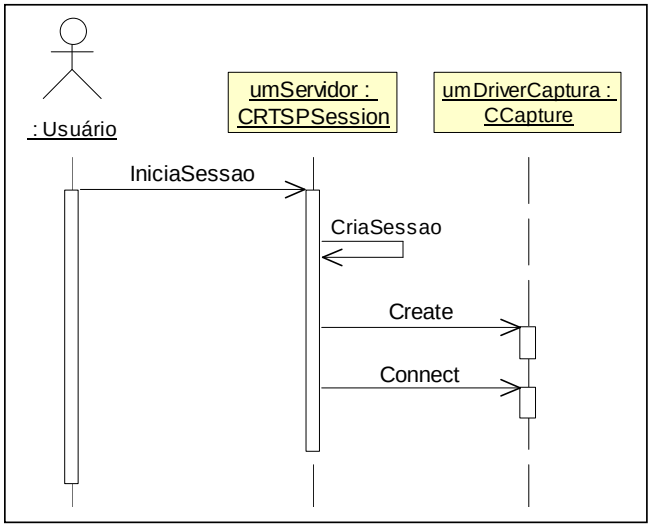


FIGURA 16 – DIAGRAMA DE SEQÜÊNCIA TERMINA SESSÃO

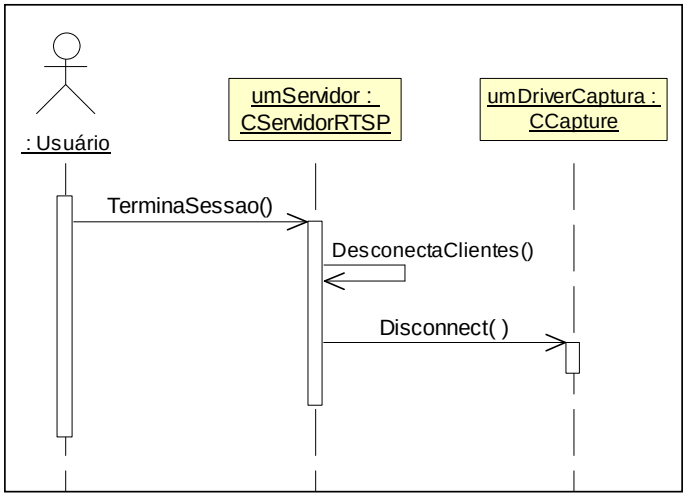


FIGURA 17 – DIAGRAMA DE SEQÜÊNCIA REPRODUZ STREAM

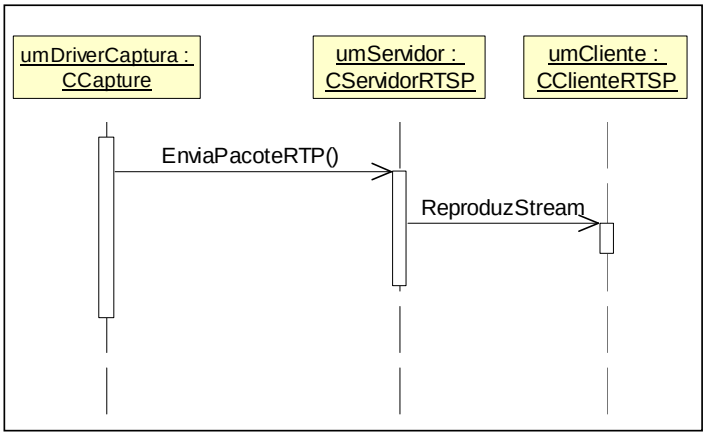


FIGURA 18 – DIAGRAMA DE SEQÜÊNCIA ESTABELECE COMUNICAÇÃO

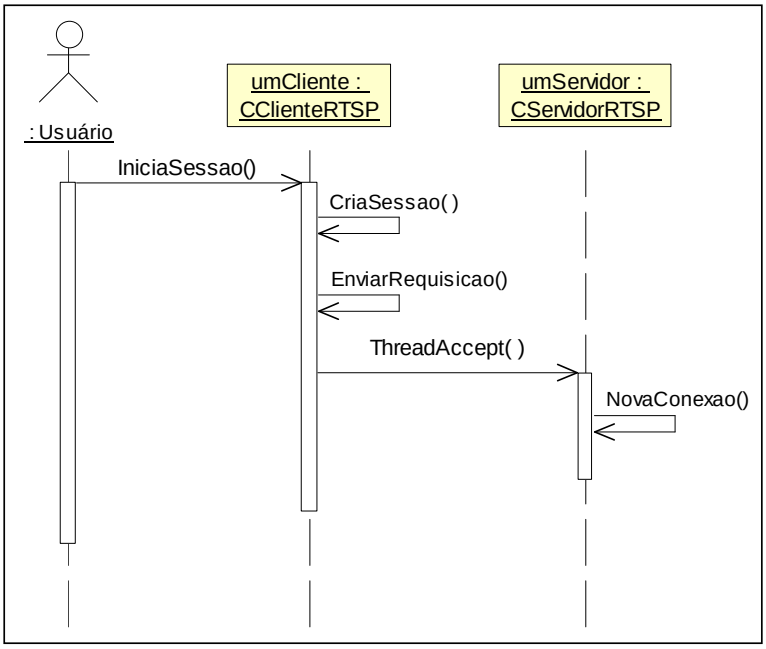


FIGURA 19 – DIAGRAMA DE SEQÜÊNCIA TERMINA CONEXÃO

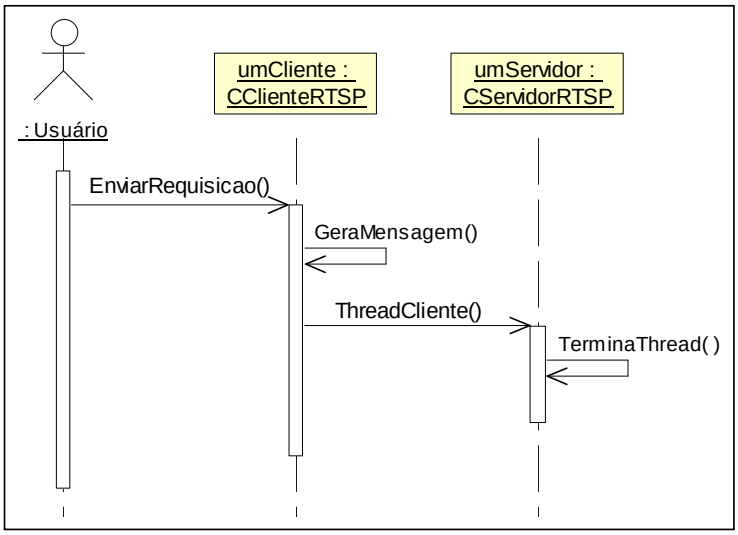
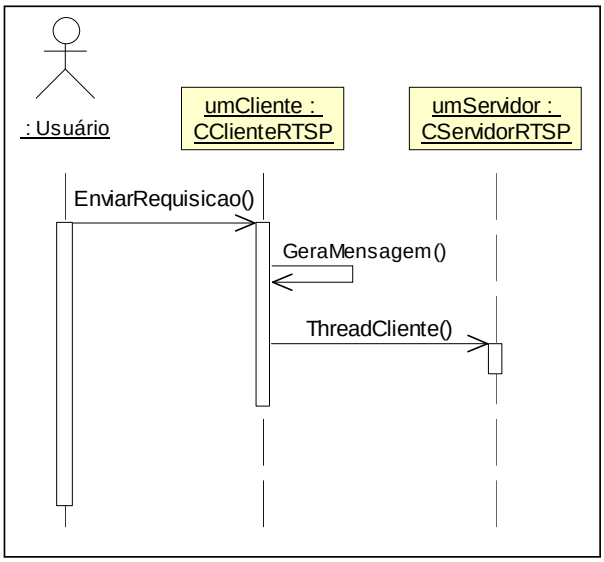
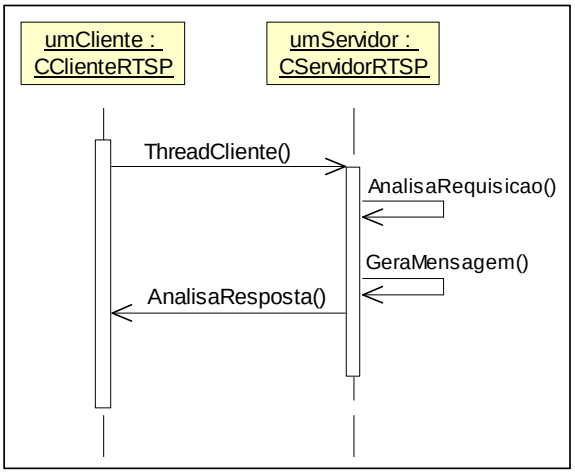


FIGURA 20 – DIAGRAMA DE SEQÜÊNCIA ENVIA REQUISIÇÃO



O código fonte que implementa o diagrama de seqüência **Envia Requisição** pode ser visualizado nos Quadros 4, 5 e 6, localizados no ANEXO 1.

FIGURA 21 – DIAGRAMA DE SEQÜÊNCIA RECEBE MENSAGENS



A implementação do diagrama de seqüência **Recebe Mensagens** está demonstrada nos Quadros 7 à 13, localizados no ANEXO 1.

5.2 IMPLEMENTAÇÃO

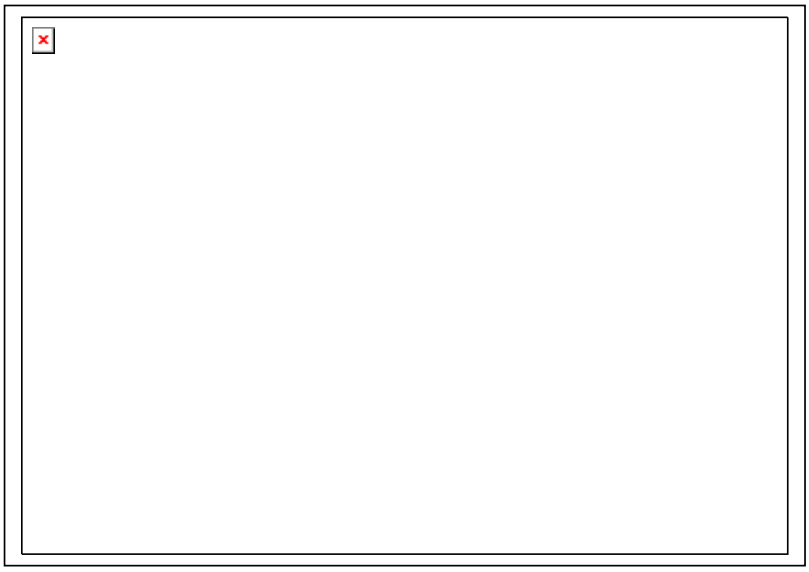
A seguir, será apresentado o funcionamento da implementação do protótipo de software, onde, serão mostradas as interfaces do sistema, bem como suas características.

O protótipo de software possui dois módulos, um servidor e um cliente. O servidor pode ser executado em qualquer computador conectado a uma LAN ou na Internet com endereço IP disponível para aceitar conexões dos clientes. O computador utilizado para executar o servidor precisa possuir uma placa de captura de vídeo passível de receber sinais externos de uma antena de TV. Por outro lado, o cliente precisa ser executado em um computador que consiga endereçar o servidor através de uma rede local ou na Internet.

5.2.1 O SOFTWARE SERVIDOR

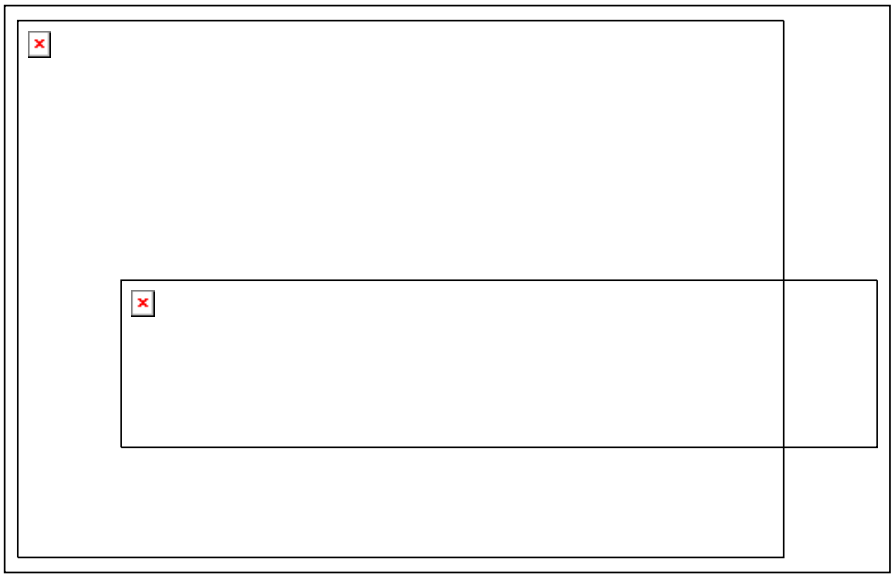
Inicialmente será apresentado o funcionamento do servidor, chamado VCODER. A Figura 22 mostra a tela inicial do software. A interface do software consitui-se de uma barra de menus que fornece ao usuário opções para trabalhar com arquivos de sessão e alterar configurações de captura. Possui também um conjunto de informações indicando tempo de captura, mensagens e pacotes RTP e RTSP.

FIGURA 22 – TELA INICIAL DO PROTÓTIPO DO SERVIDOR



Ao entrar no menu **Arquivo**, o usuário pode iniciar uma nova sessão de captura, conforme mostra a Figura 23. Uma sessão consiste em uma identificação que será usada pelo cliente para conectar-se ao servidor.

FIGURA 23 – INICIAR UMA SESSÃO

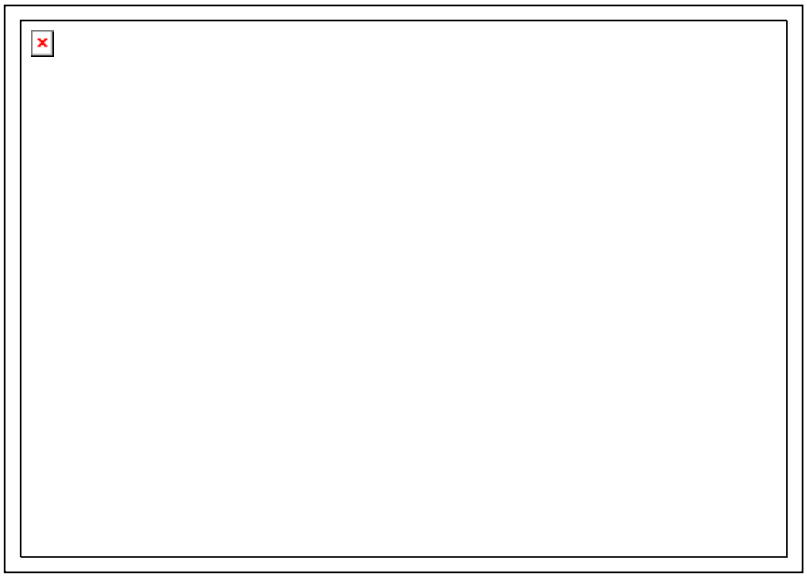


O usuário pode identificar se existe uma sessão definida ou não consultando a barra de títulos. As outras opções do menu **Arquivo** permitem ao usuário salvar as informações da sessão em disco e recuperar um arquivo de sessão previamente gravado.

Definida uma sessão, o usuário pode iniciar a captura através da opção **Iniciar** do menu **Capturar**, conforme mostra a Figura 24. Uma sessão em andamento pode ser visualizada na Figura 25.

FIGURA 24 – INICIAR A CAPTURA DO SINAL DE VÍDEO

FIGURA 25 – UMA SESSÃO EM ANDAMENTO



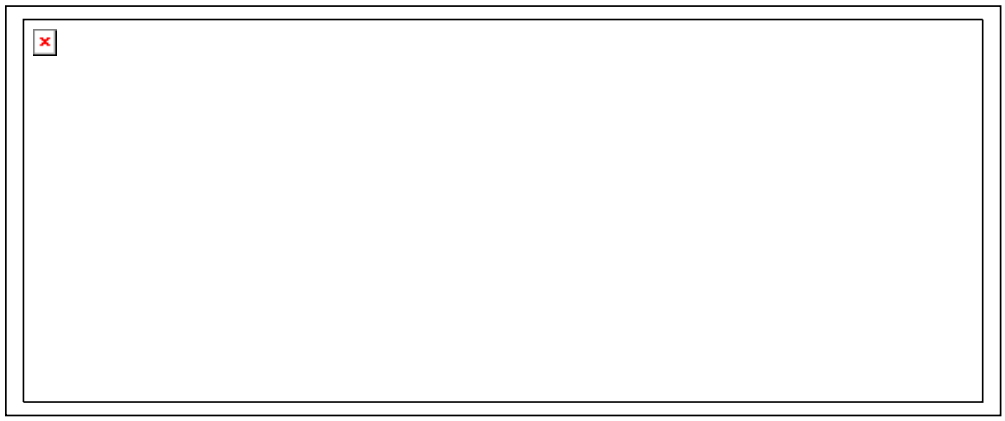
A captura pode ser terminada usando a opção **Interromper** encontrada no mesmo menu. Ao realizar essa operação, uma mensagem TEARDOWN é enviada a todos os clientes indicando que a sessão foi terminada.

As outras opções disponíveis no menu **Capturar** permitem ao usuário acessar

configurações fornecidas pelo próprio driver de captura de vídeo.

O botão **Detalhes** da tela principal exibe uma janela com informações referente às conexões ativas naquele momento, conforme mostra a Figura 26.

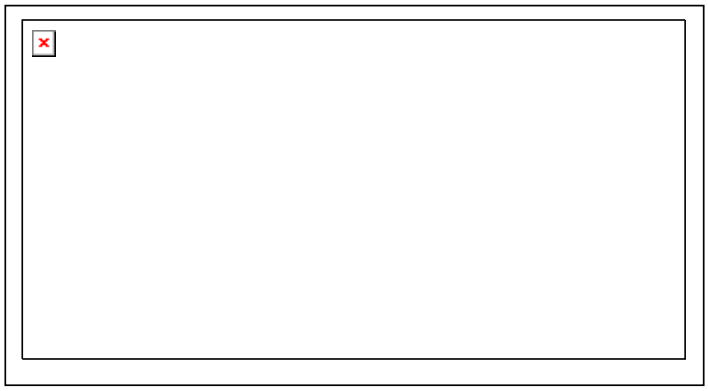
FIGURA 26 – DETALHES DAS CONEXÕES ATIVAS



5.2.2 O SOFTWARE CLIENTE

A seguir, será descrito o funcionamento do protótipo do cliente, chamado VPLAYER. A Figura 27 mostra a tela principal do protótipo, que tem por função reproduzir o vídeo fornecido pelo servidor ao qual está conectado.

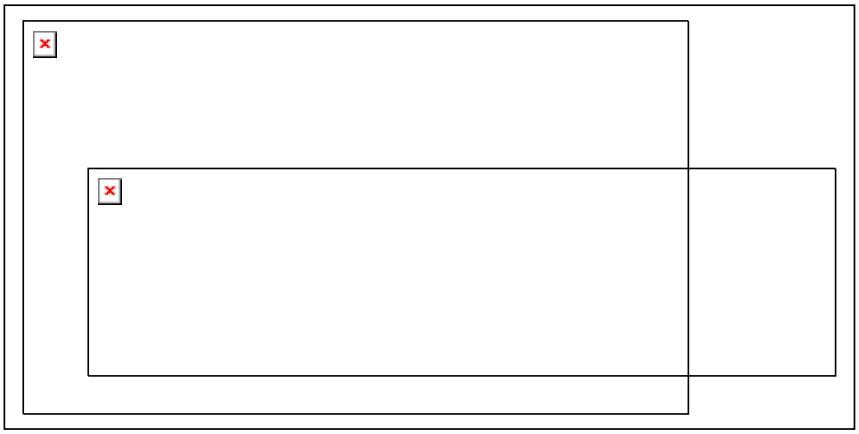
FIGURA 27 – TELA PRINCIPAL DO REPRODUTOR DE VÍDEO



A interface do cliente fornece ao usuário uma barra de menus com opções que permitem ao usuário conectar-se a uma sessão ativa e controlar a reprodução do vídeo na tela. A área central da janela está reservada para exibição do vídeo.

Para conectar-se a um servidor, o usuário deve utilizar a opção **Conectar** do menu **Arquivo** e informar o endereço e nome de uma sessão ativa, conforme mostra a Figura 28.

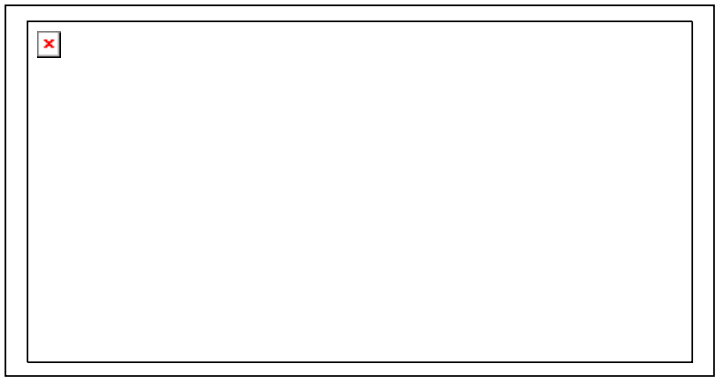
FIGURA 28 – CONEXÃO DO CLIENTE AO SERVIDOR



Para o controle da exibição do vídeo, o software oferece as operações **Executar** e **Pausar**; encontradas tanto no menu **Reproduzir** quanto na barra de ferramentas localizada na parte inferior da janela.

A Figura 29 exemplifica um cliente reproduzindo um vídeo através de uma conexão à uma sessão de nome “sessão_demonstração” e endereço 192.168.130.77.

FIGURA 29 – CLIENTE REPRODUZINDO O VÍDEO TRANSMITIDO PELO SERVIDOR



6 TRABALHOS CORRELATOS

Apesar da transmissão de vídeo em tempo real ser uma tecnologia relativamente nova, a atenção a ela dispensada vem aumentando consideravelmente, sendo possível hoje encontrar trabalhos tanto de cunho comercial quanto científico relacionados ao tema.

Através de buscas na Internet encontram-se iniciativas tratando de especificar e implementar sistemas de transmissão de vídeo em tempo real pela rede. Alguns desses trabalhos são resultantes de estudos realizados por profissionais no sentido de implementar as especificações propostas pela IETF com o objetivo de efetivar o transporte de conteúdo multimídia utilizando a tecnologia de rede existente.

Dentre os trabalhos encontrados, destacam-se:

a) **Danubio Streaming Architecture**: provê um sistema de código aberto para transmissão de *streaming media* sobre redes como Internet ou intranets. Implementa os protocolos padrão RTSP e RTP e pode ser utilizado tanto em plataformas Linux quanto Windows. Disponível em <http://www.danubio.org/>;

b) **Komssys - KOM(S) Streaming System**: fornece um servidor e cliente para *streaming* de áudio/vídeo com suporte para plataformas Linux e AIX. Utiliza o protocolo RTSP para controlar a comunicação com os servidores de *streaming* e o RTP para transportar o vídeo. Tem como principal formato de codificação de vídeo, o padrão MPEG-1. Disponível em <http://www.kom.e-technik.tu-darmstadt.de/kom-player/>;

c) **LIVE.COM Streaming Media**: um conjunto de bibliotecas de código aberto desenvolvidas em C++ que podem ser utilizadas para implementação de aplicações de *streaming media*. Pode ser compilada em Unix, Linux e Windows e fornece suporte para os protocolos RTP, RTCP e RTSP na construção de clientes e servidores de *streaming*. Disponível em: <http://live.sourceforge.net/>.

Em âmbito comercial, as principais tecnologias de *streaming* utilizadas na Internet são propostas pela Real Networks (http://www.real.com) através do Real Player e pela Microsoft (http://www.microsoft.com), com o Windows Media Player, cujo detalhamento dar-se-á nas próximas seções.

6.1 WINDOWS MEDIA SERVICES

A solução para transmissão de conteúdo multimídia através de redes oferecida pela Microsoft é conhecida por Windows Media Services. Segundo (MIC2000), é composta por uma série de serviços e ferramentas para produzir, administrar, transmitir e receber conteúdo multimídia através da Internet e redes corporativas.

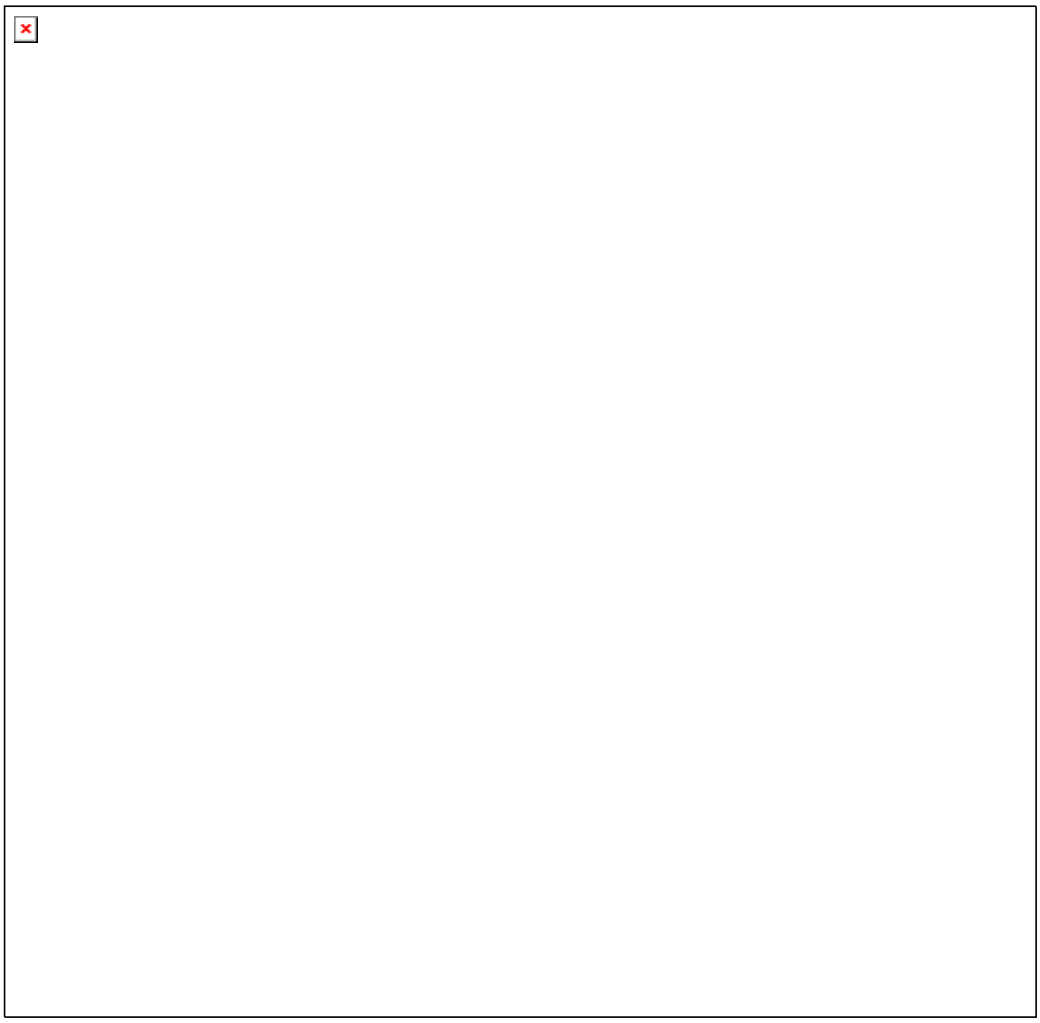
De acordo com (MIC2000), a arquitetura pode ser agrupada em três grupos, cada um sendo composto por uma ou mais ferramentas:

a) **Windows Media Tools**: uma série de ferramentas usadas para produzir e modificar o conteúdo que é transmitido pelo Windows Media Server. A principal ferramenta para converter um sinal de vídeo em um *stream* é conhecida por Windows Media Encoder;
b) **Windows Media Server**: é um conjunto de serviços executados em uma estação servidora Windows que transmite o sinal de vídeo aos clientes via *unicast* ou *multicast*;
c) **Windows Media Clients**: conhecido por Windows Media Player, recebe e renderiza *streams* de um Windows Media Server. É usado para renderizar *streams* ASF[6] que podem incluir áudio e vídeo.

O diagrama mostrado na Figura 30 ilustra, de modo geral, como um conteúdo ASF é oferecido aos usuários.

[6] **Advanced Stream Format**: um formato de dados para transmitir áudio e vídeo, imagens e comandos de *scripts* em pacotes sobre uma rede. O conteúdo pode ser um arquivo .asf ou um *stream* ao vivo gerado pelo Windows Media Encoder. ASF é um formato de arquivo proprietário desenvolvido pela Microsoft.

FIGURA 30 – VISÃO GERAL DA ARQUITETURA WINDOWS MEDIA SERVICES



6.1.1 PROTOCOLOS UTILIZADOS

Para comunicar entre os diversos componentes de um sistema Windows Media Services, (MIC2000) descreve os seguintes protocolos implementados:

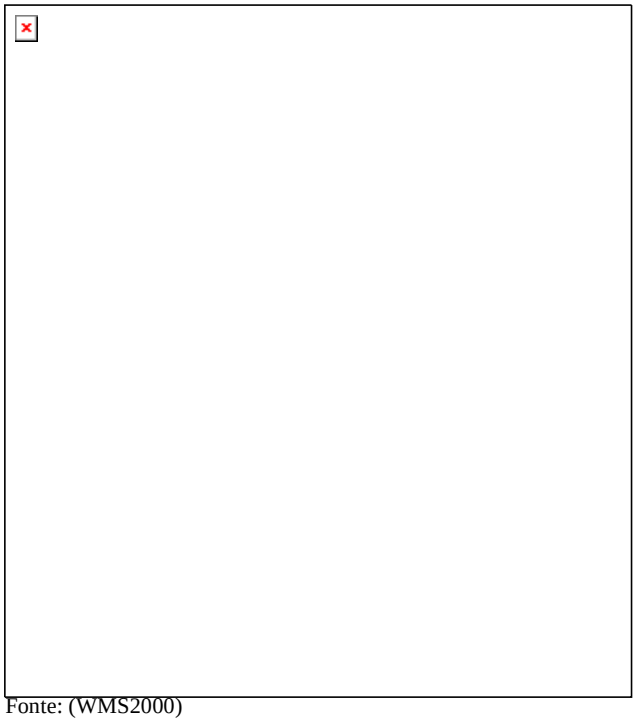
a) **MMS (Microsoft Media Server protocol)**: utilizado para acessar conteúdo *unicast* em um servidor de conteúdo multimídia. Pode funcionar tanto sobre UDP quando TCP. Ao iniciar uma conexão com o servidor, o protocolo MMS tenta comunicar-se com o servidor via UDP (MMSU); se não for possível estabelecer a conexão, o protocolo tenta uma conexão via TCP (MMST);

b) **MSBD (Media Stream Broadcast Distribution protocol)**: é usado para distribuir *streams* entre o Windows Media Encoder e o Windows Media Server. É um protocolo orientado à conexão (TCP) otimizado para o uso de *streaming media*;

c) **HTTP (Hipertext Transfer Protocol)**: o Windows Media Server também pode ser configurado para usar o protocolo HTTP para transmitir conteúdo multímida pela Internet. Entre as vantagens de seu emprego está a inexistência das restrições impostas pela utilização de *firewalls* em uma rede, visto que a maioria deles não impõe restrições na transmissão de informação via HTTP.

O diagrama mostrado na Figura 31 ilustra como os protocolos são usados para comunicar entre os componentes de um sistema Windows Media Services.

FIGURA 31 – USO DOS PROTOCOLOS NA ARQUITETURA WINDOWS MEDIA SERVICES



Fonte: (WMS2000)

6.2 REAL SYSTEM

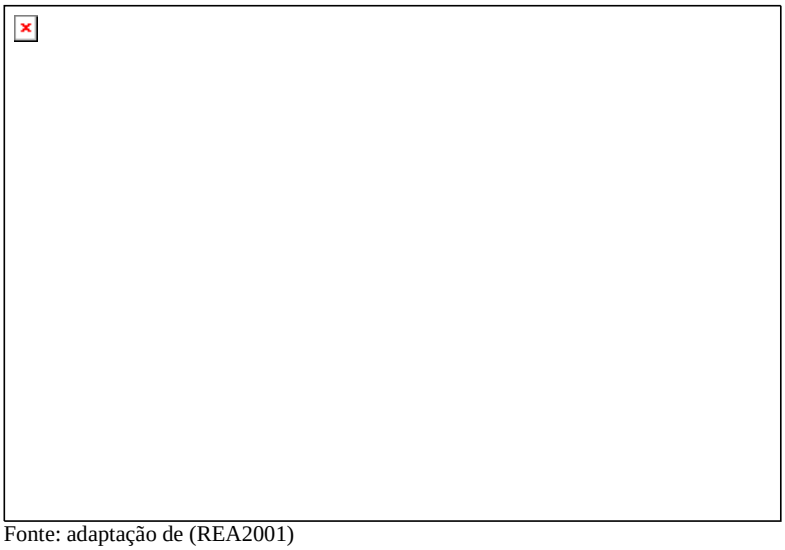
Pesquisas indicam que a Real Networks foi uma das pioneiras no desenvolvimento de aplicações multimídia para rede ao apresentar em meados da década de 90 o seu primeiro produto, o Real Audio. Com ele era possível carregar arquivos de som pela Internet e ouví-los em seu próprio computador. Desde o lançamento, o produto passou por diversas inovações e avanços e hoje é comercializado como um conjunto de ferramentas para transmissão de conteúdo multimídia chamado de Real System.

De forma análoga ao produto da Microsoft, o sistema é formado basicamente por três aplicações (REA2001):

a) **Real Producer**: é uma ferramenta que converte um sinal de diversas origens em *streaming media*, que pode gravar o conteúdo em arquivos ou transmitir *streams* ao vivo pela Internet;
b) **Real Server**: é a ferramenta responsável pela transmissão do sinal codificado pelo Real Producer aos clientes conectados através de uma intranet ou Internet. Transmite conteúdo ao vivo e sob demanda, através de redes *unicast* ou *multicast*;
c) **Real Player**: recebe e renderiza *streams* fornecidos pelo Real Server.

O diagrama mostrado na Figura 32 ilustra como os componentes do Real System trabalham em conjunto:

FIGURA 32 – VISÃO GERAL DOS COMPONENTES DO REAL SYSTEM



Fonte: adaptação de (REA2001)

1. o Real Producer converte o sinal capturado e envia continuamente um stream ao Real Server;
2. um visitante navega em uma página Web e clica em um link para uma apresentação de *streaming media* fornecida pelo Real Server;
3. o Real Server cria um pequeno arquivo *metafile* e envia ao browser solicitante;
4. o browser recebe o *metafile* e envia ao Real Player. O *metafile*, chamado RAM, contém o endereço da apresentação mencionada no link;
5. o Real Player lê o link no *metafile* e requisita a apresentação diretamente do Real Server;
6. o Real Server envia continuamente o arquivo de apresentação ao Real Player, que mostra a apresentação.

6.2.1 PROTOCOLOS UTILIZADOS

Segundo (REA2001), para iniciar a comunicação entre o Real Player e o Real Server, o Real Player abre uma conexão TCP. Essa conexão é usada para enviar mensagens entre os pontos nos dois sentidos. Uma vez aprovada a requisição, o Real Server envia a apresentação no sentido Servidor → Cliente usando um canal UDP.

O Real Server dispõe dos seguintes protocolos para comunicar com os clientes:

a) **RTSP (Real Time Streaming Protocol)**: protocolo cliente/servidor, especificado especialmente para servir apresentações multimídia pela rede. Descrito no RFC 2326, é um padrão aberto muito útil para transmissão de grande quantidade de informação;

b) **PNA (Progressive Networks Audio)**: protocolo cliente/servidor proprietário especificado e utilizado pela RealNetworks no RealSystem na versão 5.0 e anteriores;

c) **HTTP (Hypertext Transfer Protocol)**: protocolo usado por servidores Web para comunicar com os navegadores (*browsers*).

6.2.2 CONFIGURAÇÕES DE PORTAS

As configurações das portas indicam ao Real Server onde “ouvir” requisições RTSP, PNA e HTTP.

Requisição	Protocolo	Porta
rtsp://	RTSP	554, UDP
pna://	PNA	7070, UDP
http://	HTTP	80 ou 8080, TCP

7 CONCLUSÕES

Como se pôde constatar, a transmissão de conteúdo multimídia na Internet constitui-se um tema relativamente novo e possuidor de alguns problemas a serem solucionados, considerando-se o período de existência da rede e o fato de sua estrutura não atender aos propósitos da transmissão de informação em tempo real.

Embora muitos estudos estejam sendo realizados no sentido de adequar a Internet à transmissão deste tipo de informação, as tecnologias utilizadas no processo carecem de especificações mais detalhadas, de modo a incrementar sua utilização prática.

Propostas têm sido elaboradas com o intuito de oferecer a solução mais adequada para os problemas impostos. A exemplo, pode-se citar o trabalho realizado por grupos de estudo na IETF, a fim de definir um conjunto de protocolos padrão dispostos a viabilizar a transmissão de informação em tempo real na Internet utilizando a estrutura atual.

O estudo realizado procurou abordar as tecnologias e protocolos envolvidos na transmissão de vídeo pela Internet, visando o desenvolvimento de um software que viesse de encontro ao objetivo do trabalho

O protótipo de software desenvolvido atingiu seus objetivos, permitindo transformar um sinal de vídeo digital externo capturado por uma placa de vídeo, em um sinal de *streaming media* para a transmissão em tempo real pela Internet.

A implementação do protótipo não se preocupou com o aspecto de compressão de vídeo, por estar aquém dos objetivos do trabalho. Porém, tal aspecto constitui em relevante tema para o desenvolvimento de novos estudos nesta área.

Em relação às tecnologias e ferramentas utilizadas, pôde-se constatar que a linguagem de programação Visual C++ mostrou-se bastante adequada, pois dispunha dos recursos imprescindíveis para a realização do protótipo, como funções para acesso ao driver de captura de vídeo e à biblioteca Winsock. Quanto a ferramenta Rational Rose e a UML, ambos serviram para a modelagem do protótipo, porém, foram encontradas dificuldades para representar os diagramas de seqüência dos casos de uso que envolviam a comunicação entre o cliente e o servidor. O uso da ferramenta restringiu-se à especificação e modelagem dos

objetos, não sendo portanto, utilizada para geração de código. Com relação a biblioteca *JRTPLib*, esta apresentou boa legibilidade, o que facilitou seu entendimento apesar de possuir documentação insuficiente. O principal fator considerado na escolha desta biblioteca foi o uso da orientação à objetos em sua implementação.

7.1 LIMITAÇÕES

Por não implementar algoritmos de compressão de vídeo, o software desenvolvido consome largura de banda superior àquela considerada adequada para obter um resultado satisfatório na transmissão de vídeo pela Internet.

7.2 SUGESTÕES

Buscando aprimorar os resultados obtidos com o protótipo, sugere-se:

a) utilizar algoritmos de compressão de vídeo para reduzir o consumo dos recursos utilizados na transmissão do vídeo;
b) aprimorar o protocolo RTSP inicialmente implementado, de forma a melhor atender as especificações contidas no RFC 2326;
c) implementar um software semelhante para *streaming* de áudio.

ANEXO 1 – CÓDIGO FONTE DO PROTÓTIPO

No presente anexo são apresentadas algumas partes do código fonte do protótipo de software desenvolvido.

A seguir, os quadros 4, 5 e 6 demonstram a implementação do diagrama de seqüência **Envia Requisição**. O método **EnviarRequisicao**, apresentado no Quadro 4, é responsável pela formatação e envio da mensagem de requisição RTSP ao servidor. Para cada mensagens enviada, o cliente aguarda uma resposta do servidor, indicando a aceitação ou não da requisição.

QUADRO 4 – IMPLEMENTAÇÃO DO MÉTODO **ENVIARREQUISICAO**

```
int CClienteRTSP::EnviarRequisicao(SOCKET s, int iMensagem)
{
   if (s == INVALID_SOCKET)
   {
      WSASetLastError(WSAENOTSOCK);
      return SOCKET_ERROR;
   }

   int result;
   string szRequest, szResponse;
   WSABUF Buffer;
   DWORD dwNumeroBytesEnviados = 0;
   DWORD dwBytesEnviados = 0;

   result = GeraMensagem(iMensagem, m_iSessao, ++m_iSequencia, szRequest);
   if (result == SOCKET_ERROR)
      return result;

   //envia a mensagem
   do
   {
      Buffer.len = (szRequest.size() - dwBytesEnviados) >= SENDBLOCK ? SENDBLOCK :
                              szRequest.size() - dwBytesEnviados;
      Buffer.buf = (char*)((DWORD)szRequest.c_str() + dwBytesEnviados);

      result = WSASend(s, &Buffer, 1, &dwNumeroBytesEnviados, 0, 0, NULL);
      if(result != SOCKET_ERROR)
         dwBytesEnviados += dwNumeroBytesEnviados;
   }
   while((dwBytesEnviados < (UINT)szRequest.size()) && SOCKET_ERROR != result);

   //
   // INFORMACOES ESTATISTICAS
   //
   m_estatisticas.nRTSPByteEnviado += (double)dwBytesEnviados;
   m_estatisticas.nRTSPEnviado++;

   //espera resposta do servidor
   WSAEVENT EventArray[2];
   WSANETWORKEVENTS NetworkEvents;
   WSAEVENT Event, ShutdownEvent;

   Event = WSACreateEvent();
   if (Event == WSA_INVALID_EVENT)
      return -1;
```

QUADRO 5 – CONTINUAÇÃO DA IMPLEMENTAÇÃO DO MÉTODO **ENVIARREQUISICAO**

```
   ShutdownEvent = WSACreateEvent();
   if (ShutdownEvent == WSA_INVALID_EVENT)
      return -1;

   result = WSAEventSelect(s, Event, FD_READ|FD_WRITE|FD_CLOSE);
   if (result == SOCKET_ERROR)
      return result;

   EventArray[0] = Event;
   EventArray[1] = ShutdownEvent;

   for(;;)
   {
      DWORD EventCaused = WSAWaitForMultipleEvents(2, EventArray, FALSE, WSA_INFINITE, FALSE);
      if (WSA_WAIT_FAILED == EventCaused || EventCaused == 1 || EventCaused == WSA_WAIT_TIMEOUT)
         return -1;

      //
      // Determina qual evento de rede ocorreu
      //
      result = WSAEnumNetworkEvents(s, Event, &NetworkEvents);
      if (result == SOCKET_ERROR || !NetworkEvents.lNetworkEvents)
         continue;

      //
      // Processa os eventos de rede
      //
      if (NetworkEvents.lNetworkEvents & FD_READ)
      {
         DWORD NumberOfBytesRecvd;
         WSABUF Buffers;
         DWORD dwBufferCount = 1;
         char szBuffer[MAX_BUFFER];
         DWORD Flags = 0;
         Buffers.buf = szBuffer;
         Buffers.len = MAX_BUFFER;

         result = WSARecv(s, &Buffers, dwBufferCount, &NumberOfBytesRecvd, &Flags, NULL, NULL);
         if(result != SOCKET_ERROR)
         {
            //
            // INFORMACOES ESTATISTICAS
            //
            m_estatisticas.nRTSPByteRecebido += (double)dwBytesEnviados;
            m_estatisticas.nRTSPRecebido++;

            //
            // verifica se a resposta esta completa
            //
            szResponse += string(szBuffer, NumberOfBytesRecvd);
            if(!EstaCompleta(szRequest))
               continue;

            if (!AnalisaResposta(s, szResponse))
               return SOCKET_ERROR;

            break;
         }
         else
         {
            char buf[1024];
            sprintf(buf, "WSARecv(...) falhou: %d", WSAGetLastError());
            GravaLog(buf);
         }
      }
   }

   return 0;
}
```

O método **GeraMensagem**, utilizado pelo cliente para formatar as mensagens RTSP que serão transmitidas ao servidor, é demonstrado no Quadro 6.

QUADRO 6 – IMPLEMENTAÇÃO DO MÉTODO **GERAMENSAGEM** NA CLASSE **CCLIENTERTSP**

```
//mensagens enviadas pelo cliente
const char setup[]      = "SETUP %s/%s RTSP/1.0\nCSeq: %d\nTransport: RTP %d\r\n\r\n";
const char play[]       = "PLAY %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";
const char pause[]      = "PAUSE %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";
const char teardown[]   = "TEARDOWN %s/%s RTSP/1.0\nCSeq: %d\nSession: %d\r\n\r\n";

int CClienteRTSP::GeraMensagem(int iMensagem, int iSessao, int iCSeq, string &szMensagem)
{
   char pMens[2048];
   char szDT[128];
   struct tm *newtime;
   long ltime;

   //
   // Gera string de data GMT atual
   //
   time(&ltime);
   newtime = gmtime(&ltime);
   strftime(szDT, 128, "%a, %d %b %Y %H:%M:%S GMT", newtime);

   switch (iMensagem)
   {
      case RTSP_SETUP:
         sprintf(pMens, setup, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, m_iPortaRTP);
         break;

      case RTSP_PLAY:
         sprintf(pMens, play, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, iSessao);
         break;

      case RTSP_PAUSE:
         sprintf(pMens, pause, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, iSessao);
         break;

      case RTSP_TEARDOWN:
         sprintf(pMens, teardown, m_szDestino.c_str(), m_szNomeSessao.c_str(), iCSeq, m_iSessao);
         break;

      case RTSP_RESPONSE_OK:
      case RTSP_RESPONSE_OKS:
      case RTSP_RESPONSE_400:
      case RTSP_RESPONSE_454:
      case RTSP_RESPONSE_501:
      case RTSP_RESPONSE_551:
      default:
         ASSERT(FALSE);
         WSASetLastError(WSAENOBUFS);
         return SOCKET_ERROR;
   }

   szMensagem = pMens;
   return 0;
}
```

O diagrama de seqüência **Recebe Mensagens** definido para o servidor, está implementado nos métodos **ThreadCliente**, **AnalisaRequisição** e **GeraMensagem**; apresentados nos Quadros 7 à 13.

QUADRO 7 – IMPLEMENTAÇÃO DO MÉTODO **THREADCLIENTE**

```
unsigned __stdcall CRTSPSession::ThreadCliente(LPVOID pParam)
{
   CConexaoInfo *pConexao = (CConexaoInfo*)pParam;
   CRTSPSession *pSession = pConexao->GetServidor();
   SOCKET s = pConexao->GetSocketRTSP();

   int result;
   WSAEVENT EventArray[2];
   WSANETWORKEVENTS NetworkEvents;
   WSABUF Buffer;
   DWORD NumberOfBytesSent;
   DWORD dwBytesSent;
   BOOL  bKeepAlive = FALSE;
   string szRequest;
   string szResponse;

   WSAEVENT Event = WSACreateEvent();
   if(Event == WSA_INVALID_EVENT)
   {
      pSession->GravaLog("WSACreateEvent(...) falhou", "ThreadCliente", WSAGetLastError());
      pSession->TerminaThread(NULL, s, pConexao, GetCurrentThreadId());
      return THREADEXIT_SUCCESS;
   }

   result = WSAEventSelect(s, Event, FD_READ|FD_WRITE|FD_CLOSE);
   if(result == SOCKET_ERROR)
   {
      pSession->GravaLog("WSAEventSelect(...) falhou", "ThreadCliente", WSAGetLastError());
      pSession->TerminaThread(Event, s, pConexao, GetCurrentThreadId());
      return THREADEXIT_SUCCESS;
   }

   EventArray[0] = Event;
   EventArray[1] = pSession->m_eShutdown;

   for(;;)
   {
      DWORD EventCaused = WSAWaitForMultipleEvents(2, EventArray, FALSE, WSA_INFINITE, FALSE);
      if(WSA_WAIT_FAILED == EventCaused)
      {
         pSession->GravaLog("WSAWaitForMultipleEvents falhou", "ThreadCliente", WSAGetLastError());
         pSession->TerminaThread(Event, s, pConexao, GetCurrentThreadId());
         return THREADEXIT_SUCCESS;
      }

      if(EventCaused == 1 || EventCaused == WSA_WAIT_TIMEOUT)
      {
         pSession->TerminaThread(Event, s, pConexao, GetCurrentThreadId());
         return THREADEXIT_SUCCESS;
      }

      //
      // Determina qual evento de rede ocorreu
      //
      result = WSAEnumNetworkEvents(s, Event, &NetworkEvents);
      if(result == SOCKET_ERROR)
      {
         pSession->GravaLog("WSAEnumNetworkEvents falhou", "ThreadCliente", WSAGetLastError());
         continue;
      }
```

QUADRO 8 – CONTINUAÇÃO (1) DA IMPLEMENTAÇÃO DO MÉTODO **THREADCLIENTE**

```
      if(!NetworkEvents.lNetworkEvents)
         continue;

      //
      // Processa eventos da rede
      //
      if(NetworkEvents.lNetworkEvents & FD_READ)
      {
         DWORD NumberOfBytesRecvd;
         WSABUF Buffers;
         DWORD dwBufferCount = 1;
         char szBuffer[MAX_BUFFER];
         DWORD Flags = 0;
         UINT iRequisicao = 0;
         Buffers.buf = szBuffer;
         Buffers.len = MAX_BUFFER;

         result = WSARecv(s, &Buffers, dwBufferCount, &NumberOfBytesRecvd, &Flags, NULL, NULL);
         if(result != SOCKET_ERROR)
         {
            pSession->m_estatisticas.nRTSPRecebido++;
            pSession->m_estatisticas.nRTSPByteRecebido += (double)NumberOfBytesRecvd;

            //
            // Verifica se a requisicao esta completa
            //
            szRequest += string(szBuffer, NumberOfBytesRecvd);
            if (!pSession->EstaCompleta(szRequest))
               continue;

            result = pSession->AnalisaRequisicao(s, szRequest, szResponse, iRequisicao);
            if (iRequisicao == RTSP_SETUP && result != SOCKET_ERROR)
            {
               struct sockaddr_in addr;
               int len = sizeof(struct sockaddr_in);

               result = getpeername(s,(struct sockaddr*)&addr, &len);
               if (result == SOCKET_ERROR)
               {
                  pSession->GravaLog("WSAEnumNetworkEvents(...) falhou", "ThreadCliente",
                                     WSAGetLastError());
                  continue;
               }

               //
               // PEGA A PORTA RTP PARA GUARDAR NA CConexaoInfo do cliente
               //
               int n = szRequest.find("RTP", 0);
               if (n != string::npos)
               {
                  n = szRequest.find(" ", n);
                  if (n != string::npos)
                  {
                      string szPorta = szRequest.substr(n+1, szRequest.find("\n", n)-n);
                      addr.sin_port = htons(atoi(szPorta.c_str()));
                  }
               }

               pConexao->SetRTPInfo(addr);
            }
```

QUADRO 9 – CONTINUAÇÃO (2) DA IMPLEMENTAÇÃO DO MÉTODO **THREADCLIENTE**

```
            //
            // Envia resposta ao cliente
            //
            NumberOfBytesSent = 0;
            dwBytesSent = 0;
            do
            {
               Buffer.len = (szResponse.size() - dwBytesSent) >= SENDBLOCK ? SENDBLOCK:
                                 szResponse.size() - dwBytesSent;
               Buffer.buf = (char*)((DWORD)szResponse.c_str() + dwBytesSent);

               result = WSASend(s, &Buffer, 1, &NumberOfBytesSent, 0, 0, NULL);
               if(SOCKET_ERROR != result)
                  dwBytesSent += NumberOfBytesSent;
            }
            while((dwBytesSent < szResponse.size()) && SOCKET_ERROR != result);

            if(WSAGetLastError() == WSAEWOULDBLOCK)
            {
               bResend = TRUE;
               continue;
            }

            if(result != SOCKET_ERROR)
            {
               pSession->m_estatisticas.nRTSPByteEnviado += (double)dwBytesSent;
               pSession->m_estatisticas.nRTSPEnviado++;
            }
            else
               pSession->GravaLog("WSASend(...) falhou", "ThreadCliente, primeiro envio",
                           WSAGetLastError());

            szRequest.erase(0, string::npos);
         }
         else
            pSession->GravaLog("WSARecv(...) falhou", "ThreadCliente", WSAGetLastError());
      }

      if(NetworkEvents.lNetworkEvents & FD_CLOSE)
      {
         pSession->TerminaThread(Event, s, pConexao, GetCurrentThreadId());
         return THREADEXIT_SUCCESS;
      }
   }

   return THREADEXIT_SUCCESS;
}
```

O método **ThreadCliente** é implementado seguindo o conceito de processos concorrentes, ou *threads*; sendo criada uma nova *thread* para cada conexão estabelecida com um cliente. Dessa forma, o método é responsável pela recepção das mensagens RTSP enviadas pelo cliente ao qual está associado o *socket* '**s**', utilizado no comandos: *WSARecv*, para recepção das mensagens; e *WSASend*, para envio das respostas às requisições do cliente.

O método **AnalisaRequisicao**, utilizado pelo servidor para processar as requisições RTSP enviadas pelo cliente, é demonstrado no Quadro 10.

QUADRO 10 – IMPLEMENTAÇÃO DO MÉTODO **ANALISAREQUISICAO**

```
int CServidorRTSP::AnalisaRequisicao(SOCKET s,string szRequisicao,string &szResposta,UINT iRequisicao)
{
   string szOperacao;
   string szCSeq;
   string szSessao;
   int n, result;

   GravaLog(szRequisicao.c_str());

   //
   // Verifica Sessao
   //
   int iSessao = ExtraiSessao(szRequisicao);

   //
   // Verifica CSeq
   //
   int iSeq = ExtraiCSeq(szRequisicao);
   if (iSeq > 0)
   {
      //
      // Verifica operacao
      //
      n = szRequisicao.find(" ", 0);
      if (n != string::npos)
      {
         szOperacao = szRequisicao.substr(0, n);
         if(szOperacao == "SETUP")
         {
            n = szRequisicao.find("/", 0);
            if (n != string::npos)
            {
               int n2 = szRequisicao.find(" ", n);
               szSessao = szRequisicao.substr(n+1, n2-n-1);

               if (szSessao != m_szNomeSessao)
               {
                  GeraMensagem(RTSP_RESPONSE_454, iSessao, iSeq, szResposta);
                  return SOCKET_ERROR;
               }
            }

            iRequisicao = RTSP_SETUP;
            m_iSessao += 10;

            struct sockaddr_in addr;
            int len = sizeof(struct sockaddr_in);

            result = getpeername(s,(struct sockaddr*)&addr, &len);
            if (result == SOCKET_ERROR)
            {
               GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
               return result;
            }

            addr.sin_port = htons(ExtraiRTPport(szRequisicao));
            ASSERT(addr.sin_port > 0);

            m_clientes.SetAt(m_iSessao, addr);
            TRACE("cliente rtp:%s (%d)\n", inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));

            GeraMensagem(RTSP_RESPONSE_OKS, m_iSessao, iSeq, szResposta);
         }
```

QUADRO 11 – CONTINUAÇÃO (1) DA IMPLEMENTAÇÃO DO MÉTODO **ANALISAREQUISICAO**

```
else if (iSessao > 0)
{
   if (szOperacao == "PLAY" || szOperacao == "PAUSE" || szOperacao == "TEARDOWN")
   {
      //
      // Pega o endereco relacionado a sessao
      //
      struct sockaddr_in addr;
      if (!m_clientes.Lookup(iSessao, addr))
      {
         GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
         return SOCKET_ERROR;
      }

      if (szOperacao == "PLAY")
      {
         iRequisicao = RTSP_PLAY;

         result = m_rtp.AddDestination(ntohl(addr.sin_addr.S_un.S_addr),
                           ntohs(addr.sin_port));
         if (result < 0)
         {
            GravaLog(RTPGetErrorString(result));
            GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
         }
         else
            GeraMensagem(RTSP_RESPONSE_OK, iSessao, iSeq, szResposta);
      }

      if (szOperacao == "PAUSE")
      {
         iRequisicao = RTSP_PAUSE;

         result = m_rtp.DeleteDestination(ntohl(addr.sin_addr.S_un.S_addr),
                           ntohs(addr.sin_port));
         if (result < 0)
         {
            GravaLog(RTPGetErrorString(result));
            GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
         }
         else
            GeraMensagem(RTSP_RESPONSE_OK, iSessao, iSeq, szResposta);
      }

      if (szOperacao == "TEARDOWN")
      {
         iRequisicao = RTSP_TEARDOWN;

         result = m_rtp.DeleteDestination(ntohl(addr.sin_addr.S_un.S_addr),
                           ntohs(addr.sin_port));
         if (result < 0)
         {
            GravaLog(RTPGetErrorString(result));
            GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
         }
         else
            GeraMensagem(RTSP_RESPONSE_OK, iSessao, iSeq, szResposta);
      }
   }
```

QUADRO 12 – CONTINUAÇÃO (2) DA IMPLEMENTAÇÃO DO MÉTODO **ANALISAREQUISICAO**

```
            else
            {
               GravaLog("Not implemented");
               GeraMensagem(RTSP_RESPONSE_501, iSessao, iSeq, szResposta);
               GravaLog(szResposta.c_str());
               return FALSE;
            }
         }
         else
         {
            GravaLog("Session not found");
            GeraMensagem(RTSP_RESPONSE_454, iSessao, iSeq, szResposta);
            GravaLog(szResposta.c_str());
            return FALSE;
         }
      }
      else
      {
         GravaLog("Bad Request");
         GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
         GravaLog(szResposta.c_str());
         return FALSE;
      }
   }
   else
   {
      GravaLog("Bad Request");
      GeraMensagem(RTSP_RESPONSE_400, iSessao, iSeq, szResposta);
      GravaLog(szResposta.c_str());
      return FALSE;
   }

   GravaLog(szResposta.c_str());
   return TRUE;
}
```

O método **AnalisaRequisição** implementa tratamentos para requisições SETUP, PLAY, PAUSE e TEARDOWN e gera mensagens de resposta devolvidas aos clientes indicando se a requisição foi aceita ou não.

O método **GeraMensagem**, utilizado pelo servidor para formatar as mensagens de resposta enviadas aos clientes, é demonstrado no Quadro 13.

QUADRO 13 – IMPLEMENTAÇÃO DO MÉDOTO **GERAMENSAGEM**

```
//mensagens enviadas pelo servidor
const char response_ok[]    = "RTSP/1.0 200 OK\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_setup[] = "RTSP/1.0 200 OK\nCSeq: %d\nSession: %d\nTransport: RTP %d\nDate:
                                    %s\r\n\r\n";
const char response_400[]   = "RTSP/1.0 400 Bad Request\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_454[]   = "RTSP/1.0 454 Session not found\nCSeq: %d\nDate: %s\r\n\r\n";
const char response_501[]   = "RTSP/1.0 501 Not Implemented\nCSeq: %d\nSession: %d\nDate: %s\r\n\r\n";
const char response_551[]   = "RTSP/1.0 551 Option not supported\nCSeq: %d\nSession: %d\nDate:
                                    %s\r\n\r\n";

int CServidorRTSP::GeraMensagem(int iMensagem, int iSessao, int iCSeq, string &szMensagem)
{
   char pMens[2048];
   char szDT[128];
   struct tm *newtime;
   long ltime;

   //
   // Gera string de data GMT atual
   //
   time(&ltime);
   newtime = gmtime(&ltime);
   strftime(szDT, 128, "%a, %d %b %Y %H:%M:%S GMT", newtime);

   switch (iMensagem)
   {
      case RTSP_RESPONSE_OK:
         sprintf(pMens, response_ok, iCSeq, iSessao, szDT);
         break;

      case RTSP_RESPONSE_OKS:
         sprintf(pMens, response_setup, iCSeq, iSessao, m_iPortaRTP, szDT);
         break;

      case RTSP_RESPONSE_400:
         sprintf(pMens, response_400, iCSeq, iSessao, szDT);
         break;

      case RTSP_RESPONSE_454:
         sprintf(pMens, response_454, iCSeq, szDT);
         break;

      case RTSP_RESPONSE_501:
         sprintf(pMens, response_501, iCSeq, iSessao, szDT);
         break;

      case RTSP_RESPONSE_551:
         sprintf(pMens, response_551, iCSeq, iSessao, szDT);
         break;

      case RTSP_SETUP:
      case RTSP_PLAY:
      case RTSP_PAUSE:
      case RTSP_TEARDOWN:
      default:
         ASSERT(FALSE);
         WSASetLastError(WSAENOBUFS);
         return SOCKET_ERROR;
   }

   szMensagem = pMens;
   return 0;
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

(CAR1994) CARVALHO, Teresa C. Melo de Brito. **Arquitetura de redes de computadores OSI e TCP/IP**. São Paulo: Makron Books, 1994.

(CHI1999) CHIOZZOTTO, Mauro e SILVA, Luis Antonio P. **TCP/IP tecnologia e implementação**. São Paulo: Érica, 1999.

(COA1992) COAD, Peter e YOURDON, Edward. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1992.

(COM1997) COMMER, Douglas E., STEVENS, David L. **Internetworking with TCP/IP:** client-server programming and applications. New Jersey: Prentice-Hall, 1997.

(COM2001) COMMER, Douglas E. **Redes de computadores e Internet**. 2. ed. Porto Alegre: Bookman, 2001.

(CYC2000) CYCLADES BRASIL. **Guia Internet de conectividade.** 6. ed. São Paulo: Editora SENAC, 2000.

(FLU1995) FLUCKIGER, François. **Understanding networked multimedia:** applications and technology. Londres: Prentice Hall, 1995.

(FUR1998) FURLAN, José Davi. **Modelagem de objetos através da UML.** São Paulo: Makron Books, 1998.

(GRA1996) GRALLA, Preston. **Como funciona a Internet**. São Paulo: Quark Editora, 1996.

(HOL1999) HOLZNER, Steven. **Programando Visual C++ em tempo recorde.** Tradução: Jeremias René D. P. dos Santos. São Paulo: Makron Books, 1999.

(HUG2000) HUGO, Vitor. **Video digital**. Disponível em: <http://www.geocities.com/hollywood/studio/1630/> . Acesso em: 10 mai. 2001.

(LIE2000) LIESENBORGS, Jori. **JRTPLIB**. Bélgica, 2000. Disponível em: <http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html>. Acesso em: 12 nov. 2001.

(LIU2000) LIU, Chunlei. **Multimedia Over IP: RSVP, RTP, RTCP, RTSP**. Ohio, 2000. Disponível em: <http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multimedia/index.htm>. Acesso em: 19 out. 2001.

(MAR2001) MARQUES, Hugo; CRISTOVÃO, João. Berkeley, **Vídeo na Internet**. [2001?]. Disponível em <http://amalia.img.lx.it.pt/~fp/st/ano2000_2001/trabalhos2000_2001/trabalho2>. Acesso em 10 mai. 2001.

(MIC2000) MICROSOFT, Corporation. **Windows Media Services SDK**. [S.l]. 18 jul. 2000. Disponível em <http://msdn.microsoft.com/code/default.asp?url=/code/sample.asp?url=/msdn-files/027/000/961/msdncompositedoc.xml>. Acesso em: 12 nov. 2001.

(PER2001) PÉRICAS, Francisco A.. **Redes de computadores:** conceitos e arquiteturas. Blumenau, [2001?]. Disponível em: <http://www.inf.furb.br/~pericas>. Acesso em: 16 set. 2001.

(RAT2001) RATIONAL, Software Corporation. **Unified modeling language v.1.3**, [S.l], jan. 2000. Disponível em: <http://www.rational.com/uml/resources/documentation>. Acesso em: 06 nov. 2001.

(REA2001) REAL, Netwoks. **Resources & support**, [S.l]. [2001?]. Disponível em <http://www.realnetworks.com/resources/index.html>. Acesso em: 13 nov. 2001.

(REG2000) REGIS, Eduardo J. **Protótipo de software para distribuição de arquivos recebidos por e-mail via Internet**. Blumenau, 2000. 86 f. Trabalho de Graduação (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau.

(ROB1995) ROBERTS, Dave. **Developing for the Internet with Winsock**. Scottsdale: Coriolis Group Books, 1995.

(ROS2001) ROSS, Keith W. e KUROSE, James F. **Computer Neworking:** a top-down approach featuring the Internet. Boston: Addison Wesley, Inc., 2001.

(SCH1996) SCHULZRINNE, H et al. **RTP: a transport protocol for real-time applications**. [S.l], jan. 1996. Disponível em <http://www.ietf.org>. Acesso em: 14 out. 2001.

(SCH1998) SCHULZRINNE, H; RAO, A e LANPHIER, R. **Real Time Streaming Protocol (RTSP)**. [S.l], abr. 1998. Disponível em <http://www.ietf.org>. Acesso em: 10 out. 2001.

(TAN1996) TANENBAUM, Andrew S. **Redes de computadores**. Rio de Janeiro: Campus, 1996.

(TEC2001) TECNOLOGY GUIDES: White papers for IT professionals. **Real Time video on the Internet**. [S.l], 2001. Disponível em <http://www.techguide.com/titles/rtvideo.shtml>. Acesso em: 22 out. 2001.