

ANTONIO BENEDITO COIMBRA SAMPAIO JUNIOR

**ESPECIFICAÇÃO DE UM PROTOCOLO DE COMUNICAÇÃO HIPERMÍDIA COM
QUALIDADE DE SERVIÇO E MODULARIDADE FUNCIONAL**

DISSERTAÇÃO DE MESTRADO

Departamento de Informática

Rio de Janeiro, 12 de Junho de 2001.

ANTONIO BENEDITO COIMBRA SAMPAIO JUNIOR

**ESPECIFICAÇÃO DE UM PROTOCOLO DE COMUNICAÇÃO HIPERMÍDIA COM
QUALIDADE DE SERVIÇO E MODULARIDADE FUNCIONAL**

Dissertação de Mestrado apresentada ao Departamento de Informática da PUC-Rio, como parte dos requisitos para obtenção do título de Mestre em Informática: Ciência da Computação.

Orientador: Luiz Fernando Gomes Soares.

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 12 de Junho de 2001

Este trabalho é dedicado

aos meus amados pais, Antonio Benedito Coimbra Sampaio e Marly Maklouf dos Santos Sampaio, pelo incentivo, carinho e dedicação em todos os momentos, sempre sem medir esforços para me ajudar.

Agradecimentos

- Ao orientador e mestre, professor Luiz Fernando Gomes Soares, pelo apoio e pelas importantes sugestões apresentadas que nos direcionaram para a realização deste trabalho.
- Ao co-orientador professor Sérgio Colcher que colaborou para a realização deste trabalho.
- Aos meus pais e irmãos, Antonio, Marly, Andrea e Victor, que, apesar da distância, se fizeram fortemente presentes durante todo o desenvolvimento deste trabalho.
- À Daniella, pelo apoio, carinho e conselhos sempre úteis.
- Ao professor Bruno Feijó pela ajuda dispensada.
- À equipe do Laboratório Telemídia, pela amizade e pelo companheirismo, que muito contribuíram para a realização deste trabalho. Em especial, aos amigos Antonio Tadeu, Déborah Muchaluat e Rogério Rodrigues, pela presença constante, apoio e orientação indispensáveis à efetivação deste trabalho.
- Aos meus amigos, Luciana, Milene e Thyago, pelo apoio e amizade.
- À Pontifícia Universidade Católica do Rio de Janeiro, na figura de seus professores, pelo ensino ministrado no decorrer do curso.
- À Universidade da Amazônia e CNPq pelo apoio financeiro fornecido durante o curso.
- A Deus por ter me iluminado e me dado força e inspiração para conseguir terminar este trabalho.

RESUMO

Nos últimos anos, tem-se observado uma crescente demanda pela transmissão de informações multimídia com garantias de Qualidade de Serviço (QoS – *Quality of Service*) na Internet. Esta demanda tem sido motivada, principalmente, pelo fato de que a rede mundial de computadores, desde a sua concepção, oferece apenas o serviço de melhor-esforço, sem garantias de entrega dos pacotes de dados transmitidos pelas aplicações. Em situações de congestionamento, os referidos pacotes podem ser perdidos ou sofrer atrasos indeterminados, o que não é desejável para as aplicações multimídia. Motivado pelo fato de que o protocolo de comunicação hipermídia utilizado na WWW, o HTTP, não oferece o suporte adequado à transmissão multimídia com garantias de QoS, este trabalho foi desenvolvido tendo como objetivo apresentar um novo protocolo de comunicação hipermídia que oferece tratamento de qualidade de serviço (QoS), modularidade funcional e independência de protocolos de transporte e reserva de recursos. Para desenvolver este protocolo, foi realizada uma análise dos diversos protocolos de comunicação hipermídia e multimídia desenvolvidos pela IETF, ITU e WAPForum, e foram levantadas as principais características e limitações existentes nestes protocolos. Com base neste estudo, foi proposto o novo protocolo de comunicação hipermídia que possui as principais características desejáveis encontradas nos demais protocolos, sem contudo, herdar as suas limitações. Diversos cenários de uso do novo protocolo de comunicação desenvolvido foram apresentados e testados, tendo sido dado destaque ao cenário solicitação de documentos com garantias de qualidade, sendo esta a contribuição central do trabalho.

Palavras-chave: Qualidade de Serviço, Protocolos de Comunicação Hipermídia e Multimídia, Internet e WWW.

ABSTRACT

In recent years an increasing demand for the transmission of multimedia information with guaranteed Quality of Service (QoS) over the Internet has arisen. This has been driven mainly by the fact that since its conception, the world wide network offers only the *best-effort* service without guarantees of delivery of the data packages transmitted through internet applications. On situations of congestion, the referred packages can be lost or suffer undetermined delays in their transmission what is not desirable in multimedia applications. Driven by the fact that the hypermedia protocol currently used on the WWW, the HTTP, does not offer adequate support for multimedia transmission with guaranteed QoS, this research was developed with the objective of presenting the complete specification of a new hypermedia communication protocol which offers Quality of Service (QoS) treatment, functional modularity and independence between the transport and resource reservation protocols. To develop the protocol here proposed, an analysis of the various hypermedia and multimedia protocols developed by IETF, ITU and WAPForum, was carried out in order to single out their core characteristics and limitations. Based on the findings of this analysis, a new hypermedia communication protocol which possesses the desirable characteristics of the other protocols but does not inherit their limitations was proposed. Various scenarios of use for the proposed protocol were presented, within these, particular emphasis was given to the scenario of document request with Quality of Service; this being the area where the main contribution of this research lies.

Key words: Quality of Service, Hypermedia and Multimedia Communication Protocols, Internet and WWW.

SUMÁRIO

1. INTRODUÇÃO	1
1.1. Necessidade de Novos Protocolos de Comunicação	1
1.2. Histórico	2
1.3. Motivação	2
1.4. Objetivos	4
1.5. Organização da Dissertação	5
2. TRABALHOS RELACIONADOS.....	7
2.1 – Protocolos de Comunicação.....	7
2.2 – Protocolos de Comunicação HiperMídia	7
2.2.1 – HTTP (<i>HyperText Transfer Protocol</i>)	7
2.2.2 - WEBDAV (<i>WWW Distributed Authoring and Versioning - DAV</i>)	11
2.2.3 - WSP (<i>Wireless Session Protocol</i>)	13
2.3 – Protocolos de Comunicação Multimídia	17
2.3.1 – RTSP (<i>Real Time Streaming Protocol</i>)	17
2.3.2 – SIP (<i>Session Initiation Protocol</i>)	19
2.3.3 – H.323.....	23
2.4 – Interfaces de Comunicação com QoS.....	28
2.4.1 - API com QoS do <i>Middleware</i> Da CaPo++	28
2.4.2 - API com QoS para Aplicações de Trabalho Cooperativo (CSCW).....	31
3. ESPECIFICAÇÃO DO PROTOCOLO.....	35
3.1- Modelo de Referência do Protocolo de Comunicação HiperMídia	35
3.2- Especificação do Protocolo de Comunicação	38
3.2.1- Ambiente de Oferecimento de Serviços (AOS).....	38
3.2.2- Especificação do Serviço.....	42
3.2.3- Especificação do Serviço de Nível Inferior Utilizado.....	47
3.2.4- Vocabulário do Protocolo.....	47
3.2.5- Formato das Mensagens.....	49
3.2.6- Regras do Protocolo.....	50
3.3- QoS no Protocolo HiperMídia	63
3.3.1- API com QoS.....	65
3.3.2. Protocolo com QoS.....	67
3.3.2.1. Instânciação do <i>Framework</i> para <i>Parametrização de Serviços</i>	68
3.3.2.2. Instânciação dos <i>Frameworks</i> para <i>Compartilhamento e Orquestração de Recursos</i> ..	73
3.3.2.2.1. Instânciação dos <i>Frameworks</i> para <i>Compartilhamento de Recursos</i>	73
3.3.2.2.2. Instânciação dos <i>Frameworks</i> para <i>Orquestração de Recursos</i>	76
4. CENÁRIOS DE USO DO PROTOCOLO DE COMUNICAÇÃO HIPERMÍDIA	78
4.1- Funcionamento do Protocolo de Comunicação HiperMídia	78
4.2. Cenário de Iniciação do Provedor de Serviços.....	78
4.3. Cenário de Requisição de Serviços com QoS	81
4.3.1. Criando a Sessão de Aplicação	81
4.3.2. Solicitando o Transporte de um Documento com Garantias de Qualidade.....	89
4.4. Cenário de Estabelecimento de Contratos de Serviço	101
4.5. Cenário de Manutenção (Controle e Gerência) do Contrato de Serviço	114
4.6. Considerações Finais.....	115

5. CONCLUSÕES & TRABALHOS FUTUROS.....	116
5.1- Comparação com Trabalhos Relacionados	116
5.1.1- Comparação das Características dos Protocolos de Comunicação.....	116
5.1.2- API com QoS.....	121
5.2- Contribuições da Dissertação.....	123
5.3- Trabalhos Futuros.....	124
Apêndice A - API do Protocolo de Comunicação Hiperímia.....	126
Apêndice B - Primitivas da Interface da Camada de Adaptação	154
Apêndice C - Mensagens do Protocolo de Comunicação Hiperímia	160
Apêndice D (HypProtocol.dtd) - A DTD do Protocolo Hiperímia	165
Referências Bibliográficas.....	168

LISTA DE FIGURAS

Figura 2.2.1 – Funcionamento Básico do HTTP	8
Figura 2.2.2 - Exemplo de Mensagem PROPFIND.....	13
Figura 2.2.3.A - Pilha de Protocolos WAP	14
Figura 2.2.3.B - Formato da Mensagem WSP.....	16
Figura 2.3.1 – Descrição de uma Sessão RTSP utilizando a linguagem RTSL.....	19
Figura 2.3.2.A – Relação Sessão, Fluxo de Dados e Protocolo de Transporte.....	20
Figura 2.3.2.B – Exemplo de um arquivo SDP	20
Figura 2.3.2.C – Exemplo de uma Sessão SIP	21
Figura 2.3.3.A – Arquitetura H.323.....	24
Figura 2.3.3.B – Modelo de Referência H.323.....	25
Figura 2.4.1.A – Arquitetura Da CaPo++	29
Figura 2.4.1.B - Exemplo de um Arquivo de Configuração.....	30
Figura 2.4.2.A - Abstrações de Sessão.....	32
Figura 2.4.2.B - Arquitetura Definida em [43].....	33
Figura 2.4.2.C – Mapeamento de Parâmetros de QoS [43].....	33
Figura 3.1 – Modelo de Referência do Protocolo de Comunicação HiperMídia	35
Figura 3.2.1.A – Exemplo de Ambiente de Oferecimento de Serviços.....	39
Figura 3.2.1.B – AOS com dois MediaPipes: para áudio e texto.....	40
Figura 3.2.1.C – Sessão de Aplicação com dois MediaPipes	41
Figura 3.2.2.A – Pattern Composite adaptado para o Protocolo HiperMídia.....	45
Figura 3.2.2.B - Troca de Mensagens para Abrir uma Sessão de Aplicação.....	46
Figura 3.2.5 – Exemplo da mensagem OpenSession.xml	49
Figura 3.2.6.A – Máquina de Estados da Sessão de Aplicação (Cliente).....	51
Figura 3.2.6.B – Máquina de Estados da Sessão de Aplicação (Servidor).....	55
Figura 3.2.6.C – Máquina de Estados do MediaPipe (Cliente)	58
Figura 3.2.6.D – Máquina de Estados do MediaPipe (Servidor)	61
Figura 3.3.1.A – Primitivas de QoS.....	65
Figura 3.3.1.B – AOS com Sessão de Aplicação e MediaPipes	66
Figura 3.3.2 – Processo de Desenvolvimento de Frameworks	68
Figura 3.3.2.1 – Instância do Framework para Parametrização de Serviços	71
(Módulo de software ServiceCategory)	71
Figura 3.3.2.2 – Instância do Framework para Compartilhamento e Orquestração de Recursos.....	75
(Módulo de software Quality)	75
Figura 4.2.A – Estações Cliente e Servidor utilizadas no Laboratório Telemídia da Puc-Rio [56]	79
Figura 4.2.B – Tela do Cliente e do Servidor iniciados no protótipo de comunicação	80
Figura 4.3.1.A – Janela do Protótipo para solicitar a abertura de uma Sessão de Aplicação.....	82
Figura 4.3.1.B – Mensagem OpenSession.xml gerada pelo protocolo hiperMídia	83
Figura 4.3.1.C – Diagrama de Sequência realizado no Cliente para solicitar a criação de uma Sessão de Aplicação com a unidade funcional QoS.....	83
Figura 4.3.1.D – Diagrama de Sequência realizado pelo Servidor para criar uma Sessão de Aplicação	86
Figura 4.3.1.E – Diagrama de Sequência realizado pelo cliente para criar uma Sessão de Aplicação.....	87
Figura 4.3.1.F – Sessão de Aplicação criada entre o cliente (imperio) e o servidor (tradicao).....	88
Figura 4.3.1.G – Janela do Protótipo para visualizar as Sessões e MediaPipes.....	88
(Controle e Dados) criados pelo Cliente	88
Figura 4.3.2.A – Janela do Protótipo para solicitar o recebimento de um Documento	90
Figura 4.3.2.B – Mensagem GetQoS.xml gerada pelo protocolo hiperMídia.....	91
Figura 4.3.2.C – Diagrama de Sequência realizado pelo Cliente para solicitar o transporte de um documento com garantias de qualidade	92
Figura 4.3.2.D – Regra de Mapeamento para Áudio codificação PCM.....	94
Figura 4.3.2.E – Regra de Mapeamento dos Parâmetros de Transporte para Parâmetros RSVP.....	96
Figura 4.3.2.F – Sessão RSVP Servidor enviando mensagens PATH	97
Figura 4.3.2.G – Mensagem GetQoS.xml gerada pelo protocolo hiperMídia	98
Figura 4.3.2.H – Diagrama de Sequência realizado pelo Servidor para iniciar o anúncio do tráfego que será gerado para transportar o documento solicitado com garantias de qualidade.....	99

Figura 4.3.2.I – Sessão RSVP Cliente recebendo mensagem PATH.....	100
Figura 4.4.A – Algoritmo para calcular o parâmetro R do protocolo RSVP.....	106
Figura 4.4.B – Sessão RSVP Cliente enviando a mensagem RESV calculada	107
Figura 4.4.C – Mensagem Reserve.xml gerada pelo protocolo hiperâmia.....	108
Figura 4.4.D – Diagrama de Sequência realizado pelo Cliente para criar um MediaPipe.....	109
Figura 4.4.E – Sessão RSVP Servidor recebendo mensagem RESV.....	110
Figura 4.4.F – Diagrama de Sequência realizado pelo Servidor para criar um MediaPipe	112
Figura 4.4.G – AOS com MediaPipe Controle e MediaPipe Dados	113
Figura 4.4.H – Player RTP para receber a transmissão do áudio (sound.wav) em tempo-real.....	113

LISTA DE TABELAS

Tabela 2.2.1.A - Métodos HTTP/1.1	10
Tabela 2.2.1.B - Códigos e Frases-Explicativas HTTP/1.1.....	10
Tabela 2.2.2 - Novos Métodos Definidos no WEBDAV	12
Tabela 2.2.3.A - Primitivas do Serviço Orientado à Conexão.....	15
Tabela 2.2.3.B - Primitivas do Serviço Não-Orientado à Conexão	15
Tabela 2.2.3.C - Mensagens WSP	17
Tabela 2.3.1 - Métodos Definidos no RTSP	19
Tabela 2.3.2.A – Métodos SIP	22
Tabela 2.3.2.B - Códigos e Frases-Explicativas HTTP/1.1.....	22
Tabela 2.3.3.A – Mensagens H.225 RAS	26
Tabela 2.3.3.B – Mensagens H.225.....	26
Tabela 2.3.3.C – Mensagens H.245.....	27
Tabela 2.3.3.D – Padrões H.323.....	27
Tabela 2.4.1 - Primitivas da API	30
Tabela 2.4.2 - Primitivas da API	32
Tabela 3.2.2 – Descrição das Primitivas de Serviço das Unidades Funcionais	43

1

INTRODUÇÃO

Esta dissertação especifica um protocolo de comunicação hipermídia com qualidade de serviço e modularidade funcional, destacando-se as principais motivações para o desenvolvimento do protocolo proposto e quais os objetivos a serem alcançados após o desenvolvimento deste trabalho.

1.1. Necessidade de Novos Protocolos de Comunicação

Nos últimos anos, tem-se observado uma verdadeira revolução na área de comunicação de dados: infovias, como a Internet, tem sido aperfeiçoadas para dar suporte a uma vasta gama de novos serviços; a rede de telefonia celular tem se expandido a uma velocidade nunca antes imaginada; os consumidores estão mais exigentes, solicitando serviços de comunicação com garantias de qualidade e baixa tarifação, tirando proveito de um mercado cada vez mais competitivo e globalizado. No Brasil, em particular, tal revolução tem proporcionado investimentos elevados para a expansão da infra-estrutura de comunicação existente e o oferecimento de novos serviços.

A área de especificação de protocolos de comunicação tem sido diretamente influenciada pelas mudanças acima destacadas. Para especificar novos serviços de comunicação, com as mais variadas características, é necessário utilizar protocolos que ofereçam as funcionalidades desejadas. Por exemplo, para oferecer um serviço de teleconferência, uma prestadora de telecomunicações deve utilizar um conjunto de protocolos que controlem a transmissão das informações, devendo garantir que o áudio e o vídeo transportados estejam sincronizados no tempo. Ocorre que muitas vezes o protocolo de comunicação desejado não existe. Nessa situação, torna-se necessário especificar novos protocolos de comunicação que atendam as necessidades desejadas e que sejam eficientes, robustos e consistentes.

O presente trabalho descreve todas as etapas realizadas para especificar um novo protocolo de comunicação hipermídia que oferece serviços de comunicação com garantias de qualidade.

1.2. Histórico

A palavra protocolo não está associada somente à comunicação de dados entre computadores, pois a idéia de especificar protocolos de comunicação é muito anterior ao surgimento dos primeiros computadores. A primeira grande invenção que passou a utilizar um conjunto bem definido de sinais para a troca de informações foi o telégrafo. Os primeiros sistemas telegráficos, como o *semáforo de Chappe*, o *semáforo com seis refletores de George Murray* e o *semáforo com dez refletores de Edelcrantz*, definiam: a codificação utilizada para controlar uma sessão (início e fim); o controle de retransmissão; o controle na taxa de transmissão; e mensagens de reconhecimento positivo e negativo [2].

O termo protocolo só foi utilizado pela primeira vez na área de comunicação de dados envolvendo computadores em 1967. O documento *A protocol for use in the NPL data communication network*, escrito por *R.A.Scantlebury* e *K.A. Bartlett*, descrevia um conjunto de regras que coordenavam a comunicação entre estações de trabalho no Laboratório Nacional de Física da Inglaterra [2]. Desde então, novos protocolos de comunicação tem sido desenvolvidos, influenciados, principalmente, pelas diversas mudanças tecnológicas que vêm ocorrendo na área de comunicação de dados.

1.3. Motivação

Um número crescente de aplicações contendo texto, imagem, áudio e vídeo, tais como: vídeo sob-demanda, videoconferência, educação à distância, correio eletrônico multimídia, tele-medicina, entre outras, estão sendo utilizadas na Internet, exigindo que as características temporais e de sincronização dessas mídias sejam mantidas durante o processo de comunicação e reprodução, sob pena de não fazer sentido para o receptor a informação recebida. Isso requer dos protocolos de comunicação uma garantia de qualidade em seus serviços.

Por outro lado, diversas aplicações multimídia estão sendo integradas na WWW (*World Wide Web*), tirando vantagem da popularidade e facilidade de acesso às informações distribuídas na Internet. Sendo um sistema hipermídia distribuído, a WWW deveria dar suporte à recepção, transmissão e apresentação das diversas mídias que podem compor um hiper-documento. Porém, a WWW não oferece o suporte adequado à transmissão multimídia, pois: 1- a Internet ainda não fornece garantias em relação a parâmetros de qualidade (serviço de melhor-esforço); e 2 – utiliza o protocolo de comunicação HTTP (*HyperText Transfer Protocol*), não fazendo distinção no tratamento das mídias contínuas e não contínuas no tempo, tratando da mesma forma, por exemplo, a transmissão de um vídeo e a transmissão de um documento texto.

O HTTP foi construído sobre a pilha TCP/IP (*Transmission Control Protocol / Internet Protocol*) [9, 37], embora mencione a possibilidade de sua utilização com outros sistemas de transporte. O protocolo TCP pode assegurar que cada pacote chegue corretamente ao seu destino e em ordem, porém, sob o ponto de vista de um protocolo responsável por transportar mídias contínuas, isso não é uma vantagem [4, 19, 20], pois: 1- utiliza o mecanismo *three-way handshake*, incorrendo em um tempo desnecessário para o recebimento e reconhecimento dos pacotes de sinalização; 2- a retransmissão de cada pacote perdido é inadequado para aplicações com requisitos de tempo real; e 3- o controle de fluxo utiliza o algoritmo *slow-start*, influenciando a interatividade de uma transmissão.

Para ser possível a utilização de aplicações multimídia na Internet, torna-se necessário o oferecimento de serviços com garantias de qualidade. Garantir qualidade significa reservar todos os recursos necessários para que uma aplicação funcione com um desempenho aceitável (QoS - *Quality of Service*). Nos últimos anos, várias propostas têm surgido objetivando o oferecimento de serviços com garantias de qualidade na Internet, destacando-se: 1- criação de um novo modelo de serviços para a Internet (serviços integrados e diferenciados); e 2- desenvolvimento de novos protocolos de comunicação com QoS.

1.4. Objetivos

Este trabalho tem como objetivo principal a especificação de um protocolo de comunicação para transferência de documentos hipermídia com garantias de qualidade na Internet. As principais funcionalidades oferecidas por este protocolo são: 1- serviços com QoS; 2- modularidade funcional; e 3- independência de protocolos de transporte e de reserva de recursos.

Serviços com QoS significa que a aplicação do usuário pode solicitar serviços especificando os parâmetros de qualidade desejados. Para oferecer funcionalidades de QoS, o protocolo de comunicação hipermídia proposto nesta dissertação, definiu: 1- API com QoS, oferecendo primitivas para negociar, renegociar e monitorar os parâmetros de QoS; e 2- mecanismos de tratamento de QoS, permitindo: a- negociação dos parâmetros de qualidade; b- alocação dos recursos necessários para o transporte das informações com a qualidade solicitada; e c- monitoração dos parâmetros de QoS negociados.

Modularidade Funcional significa que o protocolo permite que a aplicação do usuário utilize somente as unidades funcionais¹ desejadas, reduzindo a complexidade² na utilização do protocolo. Foram definidas quatro unidades funcionais (genérico, controle de acesso, conteúdo e qualidade de serviço) para o protocolo. A aplicação do usuário pode fazer uso das mais variadas combinações das unidades funcionais.

Independência de Protocolos de Transporte e de Reserva de Recursos significa que o funcionamento do protocolo de comunicação hipermídia não depende de um único protocolo de transporte e de reserva de recursos. O Modelo de Referência para o protocolo de comunicação para transferência de documentos hipermídia define uma camada de adaptação que é responsável por selecionar o protocolo de transporte e de reserva de recursos que serão utilizados.

¹ Uma unidade funcional é um agrupamento lógico de serviços. As unidades funcionais utilizadas por uma aplicação determinam os serviços que estarão disponíveis na conexão [1].

² Por exemplo, uma aplicação hipermídia (*browser WWW*) que não desejar solicitar documentos com garantias de QoS, não precisará ser programada para utilizar esta funcionalidade do protocolo, reduzindo possíveis complexidades na construção da aplicação.

1.5. Organização da Dissertação

Esta dissertação está organizada em cinco capítulos. No Capítulo 2, é feito um apanhado das principais pesquisas relacionadas aos protocolos de comunicação hipermídia e multimídia, e à interfaces de comunicação com QoS. Primeiramente, são reportados alguns trabalhos que descrevem o funcionamento básico dos principais protocolos de comunicação hipermídia e multimídia utilizados no ambiente Internet. As limitações observadas são destacadas. Em seguida, apresentam-se os principais estudos relacionados à interfaces de comunicação com QoS, destacando-se: conceitos, abstrações utilizadas e as limitações existentes.

No Capítulo 3, é especificado o protocolo de comunicação hipermídia com QoS e modularidade funcional, destacando-se: o Modelo de Referência para o protocolo de comunicação transferir documentos hipermídia; as seis etapas realizadas (ambiente de oferecimento de serviços, especificação do serviço, especificação do serviço de nível inferior, vocabulário do protocolo, formato das mensagens - sintaxe, e as regras do protocolo - semântica) na especificação do protocolo; as quatro unidades funcionais definidas para o protocolo; e o estudo de QoS no protocolo proposto, destacando-se: a API com primitivas para negociar, monitorar e renegociar os parâmetros de QoS; e o próprio oferecimento de QoS no protocolo.

No Capítulo 4, são apresentados os quatro principais cenários de uso do protocolo de comunicação para transferência de documentos hipermídia: o primeiro cenário, *iniciação do provedor de serviços*, descreve a infra-estrutura de comunicação que dará suporte aos serviços de comunicação oferecidos pelo protocolo; o segundo cenário, *requisição de serviços com QoS*, descreve todos os passos realizados pelo protocolo para solicitar o transporte de um documento com garantias de QoS; no terceiro cenário, *estabelecimento de contratos de serviço*, descrevem-se todos os passos realizados pelo protocolo de comunicação hipermídia para negociar os parâmetros de QoS desejados durante a transmissão; o quarto e último cenário, *manutenção (controle e gerência) do contrato de serviço*, descreve todos os passos realizados pelo protocolo para monitorar os parâmetros de QoS utilizados durante a transmissão do documento solicitado.

No Capítulo 5, o protocolo proposto é comparado com os protocolos de comunicação hipermídia e multimídia, e as interfaces com QoS apresentados no Capítulo 2. Além disso, apresentam-se as conclusões, objetivos buscados e alcançados, importância do trabalho desenvolvido como um todo, as principais contribuições e sugestões para futuros trabalhos.

2

TRABALHOS RELACIONADOS

Neste capítulo, descrevem-se alguns trabalhos que estão sendo desenvolvidos na área de protocolos de comunicação hipermídia e multimídia, e interfaces de comunicação com QoS. Serão apresentados somente os trabalhos que tiveram maior influência na especificação do protocolo de comunicação hipermídia com QoS e Modularidade Funcional, seja no que se refere à sua modelagem, ou no que concerne aos possíveis casos de uso do mesmo.

2.1 – Protocolos de Comunicação

Diversos grupos de pesquisa estão desenvolvendo novos protocolos de comunicação ou aperfeiçoando os já existentes. De modo a oferecer serviços de comunicação que estejam disponíveis a diversos usuários, alguns protocolos são padronizados por órgãos internacionais.

Este capítulo analisa os protocolos de comunicação hipermídia e multimídia desenvolvidos pela IETF (HTTP, WEBDAV, RTSP e SIP), ITU (H.323) e WAPForum (WSP). Eles foram selecionados devido possuírem funcionalidades semelhantes às do protocolo de comunicação para transferência de documentos hipermídia proposto. No decorrer deste capítulo, é realizada uma análise das principais características e limitações desses protocolos.

2.2 – Protocolos de Comunicação Hipermídia

2.2.1 – HTTP (*HyperText Transfer Protocol*)

O HTTP é o protocolo utilizado para a transferência de documentos hipermídia na WWW. Foi desenvolvido pelo grupo de pesquisa de redes (*Network Working Group*) da IETF e sua especificação completa encontra-se disponível na RFC 2616 [8].

O HTTP está em constante evolução. A primeira versão, 0.9, desenvolvida em 1990, era de um protocolo para a transferência de documentos texto na Internet, sempre do servidor para o cliente WWW. Em 1996, foi proposta a segunda versão do protocolo, 1.0, que oferecia diversas melhorias, tais como: 1- transferência de diversos tipos de mídias, utilizando o formato MIME (*Multipurpose Internet Mail Extensions*); 2- envio de informações do cliente ao servidor WWW; 3- autenticação de usuários. Apesar das melhorias oferecidas pela nova versão do HTTP, sérias limitações ainda permaneciam inalteradas, sendo que a principal era o fato de ser sempre necessário estabelecer uma nova conexão TCP para cada troca de mensagens requisição/resposta entre o cliente e o servidor WWW.

Em 1997, foi desenvolvida a terceira versão do HTTP, 1.1, oferecendo diversas melhorias em relação à versão 1.0, tais como: 1- oferecer conexão de transporte persistente, permitindo que uma única conexão TCP fosse utilizada para a troca de diversas mensagens requisições e respostas entre o cliente e o servidor WWW; 2- oferecer novos métodos, como: DELETE, OPTIONS, PUT e TRACE; e 3- oferecer negociação de conteúdo, permitindo selecionar as diferentes formas de apresentação de um documento hipermídia.

O HTTP/1.1 é baseado no modelo requisição/resposta, onde o cliente envia requisições ao servidor WWW e este responde com as informações solicitadas. Esta comunicação é sempre *unicast*. A Figura 2.2.1 ilustra a operação básica do protocolo HTTP/1.1.

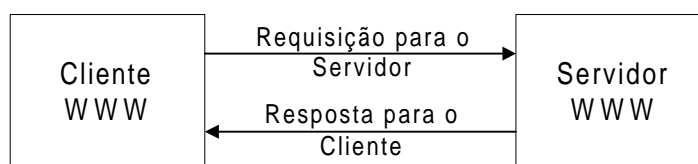


Figura 2.2.1 – Funcionamento Básico do HTTP

Todas as mensagens HTTP trocadas entre o cliente e o servidor WWW possuem o seguinte formato:

```
mensagem_HTTP
{
  linha_início,
  cabeçalhos,
  CRLF,
  corpo_mensagem
}
```

onde:

- `linha_início` - indica se a mensagem é enviada pelo cliente (`linha_início = linha_requisição_cliente`) ou pelo servidor (`linha_início = linha_status_servidor`);
- `cabeçalhos` (opcional) - cabeçalhos HTTP/1.1. São utilizados para descrever maiores detalhes da comunicação HTTP/1.1 e das informações enviadas e recebidas. Maiores detalhes podem ser encontrados em [9].
- `CRLF` - linha em branco, padrão US-ASCII;
- `corpo_mensagem` (opcional) - contém a entidade da mensagem.

A (`linha_requisição_cliente`) é composta pelos seguintes elementos: Método SP³ URI_Requisitada SP Versão_HTTP. O primeiro argumento, Método, determina qual ação será realizada pelo servidor WWW. A Tabela 2.2.1.A descreve os métodos definidos no protocolo HTTP/1.1 [8, 9]. O segundo argumento, URI_Requisitada, identifica a URI (*Uniform Resource Identifier*) do recurso desejado pela aplicação cliente. O terceiro argumento, Versão_HTTP, determina qual é a versão utilizada do protocolo HTTP.

A (`linha_resposta_servidor`) é composta pelos seguintes elementos: Versão_HTTP SP Código SP Frase_Explicativa. O argumento Código identifica o resultado de processamento realizado pelo servidor para uma dada requisição. O argumento Frase_Explicativa descreve, resumidamente, o que significa o argumento Código. A Tabela 2.2.1.B descreve os códigos e frases-explicativas definidas no protocolo HTTP/1.1.

³Significa espaço no padrão US-ASCII.

Tabela 2.2.1.A - Métodos HTTP/1.1

Método	Descrição
Connect	Definido na especificação HTTP/1.1. Utilizado somente para proxing SSL (<i>Socket Security Layer</i>).
Delete	Utilizado para apagar um recurso armazenado em um servidor WWW.
Get	Utilizado para solicitar um documento na WWW. Necessário especificar a URI (<i>Uniform Resource Identifier</i>).
Head	Utilizado para verificar a validade de uma URI na WWW. É muito semelhante ao método GET, exceto pelo fato do servidor responder sem utilizar o corpo de mensagem.
Options	Utilizado para acessar informações sobre as opções de comunicação disponíveis. Por exemplo: codificação utilizada, linguagem, etc.
Post	Utilizado para enviar informações a uma determinada URI.
Put	Utilizado para armazenar uma nova entidade (por exemplo, uma página HTML) em uma determinada URI. A diferença com o POST reside no fato que o PUT cria uma entidade, enquanto que o POST envia informações para uma entidade já existente.
Trace	Utilizado para diagnosticar a comunicação cliente / servidor.

Tabela 2.2.1.B - Códigos e Frases-Explicativas HTTP/1.1

Código	Frase-Explicativa	Significado
1xx	Informational	Indica que o servidor recebeu corretamente a requisição do cliente.
2xx	Successful	Indica que o servidor recebeu e realizou corretamente a requisição do cliente.
3xx	Redirection	Indica que outro servidor deverá receber a requisição do cliente.
4xx	Client error	Indica que o cliente enviou uma requisição errada ao servidor.
5xx	Server error	Indica que o servidor é incapaz de executar corretamente a requisição do cliente.

Apesar das diversas melhorias introduzidas na terceira versão, as várias implementações do HTTP ainda possuem algumas limitações, tais como:

1- A implementação do HTTP sempre utiliza o protocolo de transporte TCP que não é o protocolo ideal para o transporte de mídias contínuas no tempo, como áudio e vídeo. Desta forma, o HTTP torna-se inadequado para o transporte de documentos com informações multimídia contínuas no tempo;

2- Não oferece mecanismos para a solicitação de serviços com garantias de qualidade. Desta forma, não é possível solicitar um documento na WWW especificando os parâmetros de qualidade de serviço desejados⁴. A qualidade de serviço, se desejada, deve ser especificada via outro protocolo;

3- Não define métodos para trabalho cooperativo em documentos hipermídia. O HTTP é inadequado para ser utilizado em aplicações hipermídia com funcionalidades de trabalho cooperativo (CSCW - *Computer Supported Cooperative Work*);

4- Não define métodos para solicitar e controlar o envio de informações multimídia. O HTTP é inadequado para ser utilizado em aplicações que solicitam o transporte de informações multimídia contínuas no tempo;

5- Não oferece composição de unidades funcionais. Todo cliente e servidor WWW devem implementar todas as funcionalidades oferecidas pelo protocolo HTTP;

6- Não permite que o servidor WWW envie requisições de serviço. No HTTP, somente clientes podem solicitar serviços;

7- Não oferece suporte à comunicação multicast. No HTTP, toda troca de informações é sempre realizada entre um cliente e um servidor WWW.

2.2.2 - WEBDAV (*WWW Distributed Authoring and Versioning - DAV*)

Devido o HTTP/1.1 não oferecer métodos para trabalho cooperativo na WWW, o grupo de pesquisa WEBDAV da IETF, propôs diversas extensões ao protocolo HTTP/1.1, objetivando oferecer operações para autoria cooperativa na WWW. As extensões propostas foram aceitas, originando um novo protocolo de comunicação hipermídia conhecido como WEBDAV ou DAV. A especificação deste protocolo encontra-se disponível na RFC 2518 [57].

⁴ Esta é uma grande limitação do HTTP, visto que várias propostas têm surgido objetivando o oferecimento de serviços com garantias de qualidade na Internet.

O WEBDAV utiliza todas as funcionalidades especificadas no HTTP/1.1 e define novos métodos e cabeçalhos para ser possível realizar autoria cooperativa na WWW. A Tabela 2.2.2 descreve os novos métodos definidos.

As principais características do protocolo WEBDAV são [10, 11]: 1- proteção de sobrescrita (*overwrite protection*), definindo novos métodos para controlar o acesso aos recursos existentes em um servidor WEBDAV; 2- gerenciamento de recursos (*resource management*), oferecendo a noção de conjunto de recursos. Desta forma, os diversos métodos definidos no WEBDAV podem ser aplicados a um simples recurso (uma página HTML, por exemplo) ou a um recurso que contenha outros recursos (composição de páginas HTML, por exemplo); 3- propriedades de documentos (*document properties*), definindo "informações adicionais"⁵ a todo recurso WEBDAV, como: título, assunto, criador, data de criação, etc. *Por exemplo, em um documento hipermídia, estas informações adicionais podem ser: nome do autor e nome do último autor que editou o documento.*

Tabela 2.2.2 - Novos Métodos Definidos no WEBDAV

Métodos	Descrição
PropFind	Utilizado para acessar as propriedades (autor, data de criação, etc.) de um recurso DAV (por exemplo, uma página HTML).
PropPatch	Utilizado para modificar (adicionar ou remover) propriedades DAV.
MKcol	Utilizado para criar um coleção de recursos.
Copy	Utilizado para duplicar um recurso ou uma coleção que possua diversos recursos.
Move	Utilizado para mover recursos ou uma coleção que possua diversos recursos.
Lock	Utilizado para bloquear o acesso a um recurso.
UnLock	Utilizado para desbloquear o acesso a um recurso.

O formato das mensagens WEBDAV é semelhante ao padrão utilizado no HTTP/1.1. A principal diferença é que o argumento `corpo_mensagem` está especificado em XML. O uso de XML foi motivado pela facilidade em se adicionar novas propriedades DAV de um documento. Todo cliente e servidor WEBDAV utilizam um *parser* XML para poder

⁵ Também conhecido como propriedade DAV.

interpretar o argumento `corpo_mensagem`. A Figura 2.2.2 ilustra uma mensagem WEBDAV.

```
PROPFIND /WEBDAVDOCS/ HTTP/1.1
Host: www.telemidia.puc-rio.bar
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
<?xml version="1.0" encoding="utf-8" ?>
<d:propfind xmlns:d="DAV:">
  <d:allprop/>
</d:propfind>
```

Figura 2.2.2 - Exemplo de Mensagem PROPFIND

O WEBDAV é uma extensão do HTTP/1.1. Desta forma, herda várias das suas limitações, tais como as enumeradas 1, 2, 4, 5, 6 e 7 do HTTP (ver Seção 2.2.1). Além dessas limitações, existe o problema de *lost-update* que pode ocorrer quando clientes HTTP e WEBDAV editam um mesmo documento na WWW. Maiores detalhes podem ser encontrados em [57].

2.2.3 - WSP (*Wireless Session Protocol*)

WAP (*Wireless Application Protocol*) é um padrão mundial utilizado para apresentação e envio de informações em um ambiente sem fio, utilizando terminais móveis, tais como: telefones celulares, pagers, pda's (*personal digital assistants*), entre outros [51].

A especificação WAP define um conjunto de protocolos necessários para ter acesso aos serviços oferecidos na Internet sem fio. Estes protocolos e suas funcionalidades estão organizadas em um modelo de camadas, semelhante ao RM-OSI. A Figura 2.2.3.A ilustra a pilha de protocolos WAP.

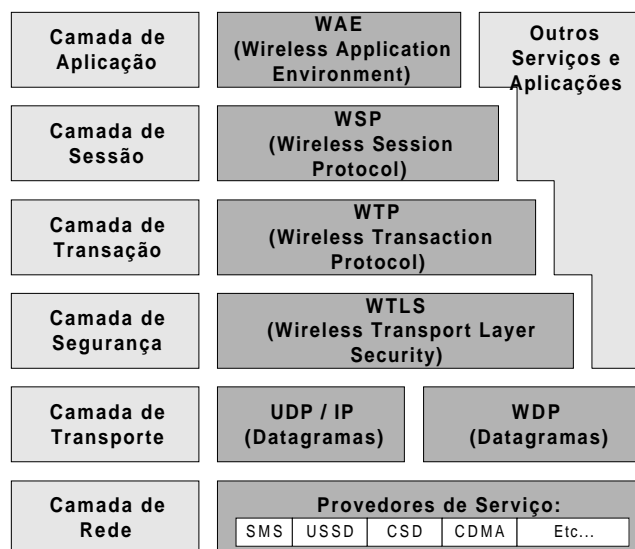


Figura 2.2.3.A - Pilha de Protocolos WAP

Cada protocolo especificado na arquitetura WAP é responsável por realizar uma determinada função. O WSP é o protocolo de comunicação responsável por organizar as trocas de informações entre aplicações cliente e servidor WAP, oferecendo [50]: 1- criação e finalização de sessões de trabalho confiáveis; 2- suspensão e reinício de uma sessão; 3- envio assíncrono de várias requisições de serviço; 4- negociação das funcionalidades a serem utilizadas pela aplicação durante o estabelecimento de uma sessão; 5- utilização de dois níveis de serviço: orientado à conexão, utilizando um serviço de transporte confiável (WTP), e não-orientado à conexão, utilizando um serviço de transporte não-confiável (WDP); e 6- uso do formato binário HTTP para troca de informações, minimizando o uso de banda passante, muito limitado em um ambiente sem fio.

A sessão de trabalho WSP é responsável por fornecer os meios necessários para que as aplicações WAP possam organizar e sincronizar as trocas de dados. No serviço orientado à conexão, a sessão WSP provê os meios necessários para que seus usuários possam estabelecer conexões, transmitir e receber dados, e encerrar conexões de forma ordenada. No serviço não-orientado à conexão, não existe nenhuma garantia na entrega das informações. As Tabelas 2.2.3.A e 2.2.3.B descrevem as primitivas definidas nos dois tipos de serviço.

Tabela 2.2.3.A - Primitivas do Serviço Orientado à Conexão

Primitiva de Serviço	Descrição
S_Connect	Inicia o estabelecimento de uma Sessão de Trabalho
S_Disconnect	Finaliza uma Sessão de Trabalho
S_Suspend	Suspende o uso de uma Sessão de Trabalho. O tempo de suspensão depende da implementação.
S_Resume	Reinicia uma Sessão de trabalho
S_Exception	Indica a ocorrência de alguma exceção, como: - Mudança do protocolo de transporte - Problemas na camada de segurança
S-MethodInvoke	Solicita que uma determinada operação seja realizada pelo servidor
S-MethodResult	Retorna uma resposta quando a primitiva S-MethodInvoke é chamada
S-MethodAbort	Finaliza o envio de uma requisição que não foi completada
S-Push	Envia informações não solicitadas, do servidor para o cliente, sem aguardar confirmação
S-ConfirmedPush	Envia informações não solicitadas, do servidor para o cliente, aguardando confirmação
S-PushAbort	Rejeita uma operação Push

Tabela 2.2.3.B - Primitivas do Serviço Não-Orientado à Conexão

Primitiva de Serviço	Descrição
S-Unit-MethodInvoke	Solicita a realização de um método no servidor, sem aguardar mensagem de confirmação
S-Unit-MethodResult	Retorna o resultado do S-UnitMethodInvoke, sem aguardar mensagem de confirmação
S-Unit-Push	Envia informações não solicitadas, do servidor para o cliente, sem aguardar confirmação

Durante a criação de uma Sessão WSP, as aplicações cliente e servidor negociam as funcionalidades de comunicação desejadas, utilizando a seguinte estratégia:

1. o cliente solicita a abertura de uma Sessão, determinando o conjunto de funcionalidades desejadas;
2. o provedor de serviços verifica se pode oferecer o conjunto de funcionalidades especificadas pelo cliente;
3. o servidor recebe a solicitação para criação de uma Sessão com um conjunto de funcionalidades solicitadas pela aplicação cliente e que podem ser oferecidas pelos provedores de serviços;
4. o servidor verifica quais são as funcionalidades solicitadas que podem ser oferecidas e cria a Sessão.

A lista de funcionalidades que podem ser negociadas durante o estabelecimento de uma Sessão WSP são [50]: 1-uso de *aliases*; 2- tamanho máximo dos pacotes; 3- métodos HTTP/1.1 que podem ser utilizados na Sessão; 4- cabeçalhos de mensagens utilizados; 5- número máximo de métodos chamados assincronamente; entre outros.

Todas as mensagens WSP possuem pelo menos três argumentos: TID (*Transaction Identifier*), associa pedidos de requisições com respostas em uma Sessão não-orientada à conexão; Tipo_Pdu (*Type*), identifica o tipo de mensagem enviada; e Conteúdo (*Type-Specific Contents*), representa o conteúdo da mensagem enviada. O formato das mensagens WSP está ilustrado na Figura 2.2.3.B. Todas as mensagens WSP estão listadas na Tabela 2.2.3.C.

TID	Tipo_PDU	Conteúdo
-----	----------	----------

Figura 2.2.3.B - Formato da Mensagem WSP

Como principais limitações do WSP, podemos destacar: 1- só é possível o envio e recebimento de informações do tipo texto e imagem, não sendo possível a utilização de mídias contínuas no tempo, devido a limitação de banda passante; e 2- não define primitivas de QoS, não sendo possível requisitar um serviço com os parâmetros de QoS desejados.

Tabela 2.2.3.C - Mensagens WSP

Mensagem WSP	Descrição
Connect	Inicia o estabelecimento de uma Sessão de Trabalho. Gerado quando a primitiva S_Connect.req é enviada pela aplicação cliente.
ConnectReply	Indica que a Sessão de Trabalho foi iniciada corretamente. Gerado quando a primitiva S_Connect.res é enviada pela aplicação servidora.
Redirect	Indica que não foi possível estabelecer a Sessão de Trabalho com o servidor. Redireciona para outros possíveis servidores. Gerado quando a primitiva S_Disconnect.req é enviada pela aplicação servidora.
Reply	É uma mensagem genérica que retorna informações do servidor em resposta a uma requisição.
Disconnect	Finaliza uma Sessão de Trabalho. Gerado quando a primitiva S_Disconnect.req é enviada pela aplicação cliente.
Suspend	Suspende uma Sessão de Trabalho. Gerado quando a primitiva S_Suspend.req é enviada pela aplicação cliente.
Resume	Reinicia uma Sessão de Trabalho. Gerado quando a primitiva S_resume.req é enviada pela aplicação cliente.
Method	Solicita que uma determinada operação seja realizada pelo servidor. Gerado quando a primitiva S_MethodInvoke.req é enviada pela aplicação cliente.
Push	Envia informações não-solicitadas do servidor para o cliente, sem aguardar mensagens de confirmação. Gerado quando a primitiva S_Push.req é enviada pela aplicação servidora.
ConfirmedPush	Envia informações não-solicitadas do servidor para o cliente, aguardando mensagens de confirmação. Gerado quando a primitiva S_ConfirmedPush.req é enviada pela aplicação servidora.

2.3 – Protocolos de Comunicação Multimídia

2.3.1 – RTSP (*Real Time Streaming Protocol*)

O RTSP é um protocolo de aplicação utilizado para solicitar e controlar o envio de mídias contínuas no tempo na Internet. Foi desenvolvido pelo grupo de pesquisa multimídia (*Multiparty Multimedia Session Control*) da IETF e sua especificação completa encontra-se disponível na RFC 2326 [5].

O RTSP foi projetado para ser utilizado nas situações em que o uso do protocolo HTTP/1.1 não é o mais adequado (transmissão de mídias contínuas no tempo, ver Seção 2.2.1). O funcionamento do RTSP foi projetado para ser similar ao do HTTP/1.1, possuindo algumas semelhanças, tais como: 1- utiliza a mesma sintaxe de mensagem -

Método SP URI_Requisitada SP Versão_RTSP, para mensagens de requisição e Versão_RTSP SP Código SP Frase_Explicativa, para mensagens de resposta; 2- faz uso da maioria dos cabeçalhos definidos nas mensagens; 3- utiliza os mesmos mecanismos de autenticação; e 4- permite a negociação de conteúdo.

Apesar das semelhanças acima destacadas, o RTSP e o HTTP/1.1 diferem em diversos aspectos [63]: 1- o RTSP define novos métodos para ser possível solicitar e controlar o envio de informações multimídia (ver Tabela 2.3.1); 2- tanto o cliente como o servidor RTSP podem solicitar serviços; 3- suporta transmissão *multicast*; 4- mantém o estado de todas requisições enviadas; 5- utiliza os protocolos TCP e UDP (*User Datagram Protocol*) para enviar mensagens RTSP e o protocolo RTP (*Real-Time Transport Protocol*) para transportar os dados em tempo-real; 6- utiliza o conceito de sessão RTSP, para ser possível enviar e receber informações multimídia; 7- pode ser facilmente integrado com o SIP [61], utilizando este protocolo para convidar usuários a participar de uma sessão multimídia e o RTSP para controlar a apresentação das informações na sessão; e 8- o RTSP define um *schema* URI próprio (["rtsp:" | "rtspu:"] "/" host [":" port] [abs_path]).

Por exemplo, rtsp://midia.exemplo.com:554/anos_dourados.mpg, onde: a- rtsp, indica que as mensagens RTSP são enviadas em uma conexão TCP. Se fosse utilizado o UDP, teríamos rtspu; b- midia.exemplo.com.br, indica o endereço do servidor de vídeo; c – 554, porta na qual o servidor recebe as solicitações dos clientes; e d - anos_dourados.mpg, nome do vídeo a ser transmitido.

Todas as mídias a serem transportadas em uma sessão RTSP devem estar descritas em um arquivo de configuração. Várias linguagens de descrição são utilizadas, sendo a RTSL (*Real Time Session Language*) a mais usada [6]. A Figura 2.3.1 ilustra a descrição de uma sessão RTSP que define a transmissão de um áudio PCM e de um vídeo H.261.

Como principais limitações do RTSP, podemos destacar: 1- só é possível o envio e recebimento de mídias contínuas no tempo, como áudio e vídeo; e 2- não define composição de unidades funcionais.

Tabela 2.3.1 - Métodos Definidos no RTSP

Mensagem	Descrição
DESCRIBE	Solicita a descrição da <i>stream</i> a ser enviada pelo servidor RTSP.
PAUSE	Indica ao servidor para parar, momentaneamente, de transmitir os dados.
PLAY	Indica ao servidor para começar a enviar os dados
RECORD	Indica ao servidor qual apresentação deve ser armazenada.
REDIRECT	Informa ao cliente que ele deve se conectar a outro servidor.
SETUP	Inicia a criação de uma sessão RTSP.
SET_PARAMETER	Informa qual parâmetro deve ser definido em uma URI.
GET_PARAMETER	Solicita o valor de um parâmetro de uma dada URI.
ANNOUNCE	Identifica qual o arquivo de apresentação a ser utilizado.
OPTIONS	Verifica os métodos disponíveis.
TEARDOWN	Indica ao servidor para finalizar o envio dos dados e liberar todos os recursos alocados.

```

<session>
  <group>
    <track src="rtsp://audio.com/audio1.pcm">
    <track src="rtsp://video.com/video1.h261">
  </group>
</session>

```

Figura 2.3.1 – Descrição de uma Sessão RTSP utilizando a linguagem RTSL

2.3.2 – SIP (*Session Initiation Protocol*)

SIP é o protocolo de controle utilizado para estabelecer, modificar e finalizar sessões multimídia na Internet. Estas sessões podem ser utilizadas no oferecimento de serviços de conferência multimídia, educação à distância, telefonia IP, entre outros. Foi desenvolvido pelo grupo de pesquisa de redes (*Network Working Group*) da IETF e sua especificação completa encontra-se disponível na RFC 2543 [61].

Toda sessão multimídia pode possuir diversos fluxos de dados. Um fluxo representa uma *stream* de dados, que pode variar quanto ao tipo (texto, áudio, vídeo, imagem, sinalização, controle), localização (arquivo, captura por câmera de vídeo, microfone, etc.) e distribuição (*unicast* ou *multicast*). Todos os fluxos criados estão associados a um

determinado protocolo de transporte. A Figura 2.3.2.A ilustra a relação entre sessão, fluxo de dados e protocolo de transporte.

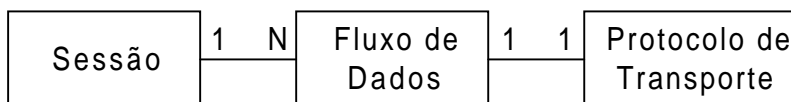


Figura 2.3.2.A – Relação Sessão, Fluxo de Dados e Protocolo de Transporte

Toda sessão multimídia pode: 1- possuir um ou vários participantes; 2- estar associada a transferência de diversos tipos de dados, como áudio e vídeo; e 3- permitir comunicação *unicast* e *multicast*. O SIP utiliza o protocolo SDP (*Session Description Protocol* – RFC 2327 [62]) para descrever as sessões multimídia. Cada descrição SDP possui: 1- os fluxos de dados que serão criados; 2- os endereços e portas dos destinatários dos fluxos; 3- os formatos de mídia utilizados durante a transmissão; 4- os horários de início e término da sessão; e 5- a estação origem responsável pela transmissão das informações multimídia. A Figura 2.3.2.B ilustra um arquivo de configuração SDP [62].

```

v=0 (indicador de versão da sessão)
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
(identificador único da sessão multimídia)
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
(url para maiores informações sobre a sessão)
t=2873397496 2873404696
(tempo inicial e final)
m=audio 49170 RTP/AVP 0
(fluxo de dados áudio)
a=rtpmap:96 VDVI/8000/1
(tipo de conteúdo do fluxo)
  
```

Figura 2.3.2.B – Exemplo de um arquivo SDP

O protocolo SIP é baseado no modelo requisição/resposta, possuindo uma arquitetura similar à do protocolo HTTP/1.1 (ver Seção 2.2.1), onde o cliente SIP (conhecido como UAC – *User Agent Client*) envia requisições ao servidor SIP (conhecido como UAS – *User Agent Server*), este as processa e retorna uma resposta ao cliente. Este processo de requisição/resposta é conhecido como transação SIP [61]. A Figura 2.3.2.C ilustra exemplos de clientes e servidores SIP.

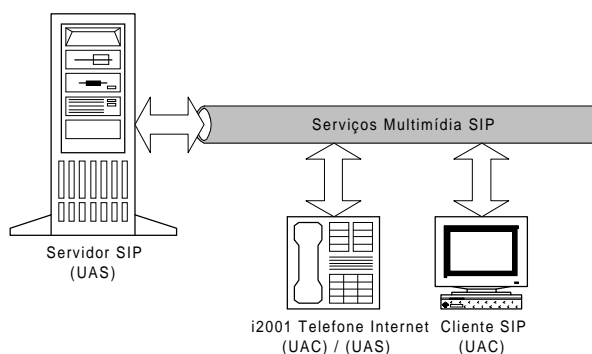


Figura 2.3.2.C – Exemplo de uma Sessão SIP

Conforme ilustra a Figura 2.3.2.C, uma aplicação para telefonia IP deverá possuir dois *software's* SIP: um cliente (UAC) e outro servidor (UAS), para ser possível “telefonar” e receber “telefonemas”, respectivamente.

Os principais serviços oferecidos pelo protocolo SIP são: 1- tradução de nome e localização do usuário, pois toda entidade SIP possui um nome no formato *usuario@host* que será traduzido em um endereço IP; 2- iniciação de chamadas, permitindo que qualquer participante de uma sessão possa convidar outros participantes a entrar ou sair de uma sessão multimídia; 3- tratamento de chamadas, onde qualquer participante pode ajustar os parâmetros definidos em uma sessão multimídia; e 4- gerenciamento de chamadas, permitindo que qualquer participante realize tarefas como transferência e finalização de chamadas.

Para oferecer todas as funcionalidades acima destacadas, o protocolo SIP define um conjunto de mensagens que serão utilizadas na comunicação cliente / servidor SIP. Estas mensagens possuem o mesmo formato das mensagens definidas no HTTP: Método SP URI_Requisitada SP Versão_SIP, para mensagens de requisição e Versão_SIP SP Código SP Frase_Explicativa, para mensagens de resposta. As Tabela 2.3.2.A e 2.3.2.B descrevem os métodos, os códigos e frases-explicativas definidas no protocolo SIP [60, 61].

Tabela 2.3.2.A – Métodos SIP

Método	Descrição
INVITE	Utilizado para convidar um usuário a participar de uma sessão. Utiliza um arquivo SDP que descreve os fluxos de dados utilizados na sessão multimídia.
ACK	Utilizado para confirmar o recebimento de uma mensagem INVITE.
OPTIONS	Utilizado para obter informações sobre as capacidades de um servidor SIP.
BYE	Utilizado para indicar que um usuário está saindo de uma sessão.
CANCEL	Utilizado para cancelar uma solicitação inicial pendente.
REGISTER	Utilizado para registrar o endereço de um usuário com um servidor SIP.

Tabela 2.3.2.B - Códigos e Frases-Explicativas HTTP/1.1

Códigos	Frases-Explicativa	Significado
1xx	Informational	Indica que o servidor recebeu corretamente a requisição do cliente.
2xx	Successful	Indica que o servidor recebeu e realizou corretamente a requisição do cliente.
3xx	Redirection	Indica que outro servidor deverá receber a requisição do cliente.
4xx	Client error	Indica que o cliente enviou uma requisição errada ao servidor.
5xx	Server error	Indica que o servidor é incapaz de executar corretamente a requisição do cliente.
6xx	Global Failure	Indica que a requisição não pôde ser executada em nenhum servidor.

As mensagens SIP são enviadas utilizando-se os protocolos de transporte UDP ou TCP. É importante ressaltar que apesar do protocolo SIP fazer parte de uma família de protocolos especificados pela IETF para a transmissão de informações multimídia, tais como: RTSP (ver Seção 2.3.1), RTP, RTCP (*Real Time Control Protocol*), RSVP (*Resource ReSerVation Setup Protocol*), entre outros, o funcionamento do SIP não depende destes protocolos. Desta forma, após o estabelecimento de sessões multimídia utilizando o SIP, é possível utilizar os mais diversos protocolos de comunicação.

Como principais limitações do SIP, podemos destacar: 1- não define métodos para suspender e reiniciar uma sessão; e 2- não define composição de unidades funcionais.

2.3.3 – H.323

O H.323 é um padrão para a transmissão de informações multimídia em tempo real nas redes comutadas por pacotes. Foi desenvolvido pelo grupo de estudos (SG16) do setor de telecomunicações do ITU. A especificação completa deste padrão encontra-se disponível na recomendação H.323 v4 [58].

A recomendação H.323 está na sua quarta versão. A primeira versão, conhecida como *Visual Telephone Systems and Equipment for Local Area Networks (LANs) that do not provide guaranteed Quality of Service (QoS)*, foi desenvolvida em outubro de 1996 e definia um conjunto de protocolos de comunicação multimídia para redes locais. Percebendo que o uso do H.323 podia ser estendido para ser utilizado em redes mundiais, o grupo de estudo (SG16) do ITU propôs, em janeiro de 1998, a segunda versão do H.323, conhecida como *Packet-based Multimedia Communications Systems*. Esta nova recomendação foi desenvolvida para oferecer um conjunto de componentes e protocolos de comunicação que deveriam ser adotados pelas empresas de telecomunicações para ser possível oferecer o serviço de telefonia IP. A terceira versão, aprovada em setembro de 1999, realizou pequenas melhorias na segunda versão, tais como: 1- reuso das chamadas de conexões; 2- controle de dispositivos remotos; e 3- desenvolvimento de novas extensões (anexos): a- *Annex E/H.323 – UDP signaling*; b- *Annex F/H.323 – Simple endpoint type* e c- *Annex G/H.225.0 – Communication between administrative domains*. A quarta versão foi aprovada em novembro de 2000, oferecendo diversas extensões à versão anterior, tais como: 1- permitir que pontos finais H.323 estabeleçam circuitos virtuais em redes ATM; e 2- integrar o protocolo de reservas de recursos (RSVP) no modelo de referência do H.323.

Para fazer uso do H.323 em um ambiente de comunicação, quatro componentes devem ser utilizados: terminais, *gateways*, *gatekeepers*⁶ e Unidades de Controle Multiponto (MCUs - *Multipoint Control Units*): 1- os **terminais** H.323 são equipamentos de *hardware* (telefones, videofones, etc.) ou sistemas de *software* (*Microsoft NetMeeting*) utilizados para oferecer suporte à comunicação multimídia bidirecional em tempo real; 2- os **gateways** são equipamentos utilizados na interligação de redes H.323 com outras

⁶ O uso do componente *gatekeeper* é opcional [58].

redes, como a rede de telefonia pública (PSTN – *Public Switched Telephone Network*); 3- os *gatekeepers* são os equipamentos fundamentais para o perfeito funcionamento de uma rede H.323, oferecendo as seguintes funcionalidades: a- tradução de endereços; b- gerenciamento de banda passante; c- monitoração da rede; e d- autorização e autenticação de terminais e *gateways*; e 4- os **MCUs** fornecem suporte para realização de conferências entre três ou mais terminais H.323. Todos os terminais participantes de uma conferência estabelecem uma conexão com um MCU que gerência os recursos necessários para realizar a conferência e negocia os codificadores / decodificadores de áudio e vídeo a serem utilizados pelos terminais H.323. Maiores detalhes podem ser encontrados em [58, 59]. A Figura 2.3.3.A ilustra os componentes da arquitetura H.323.

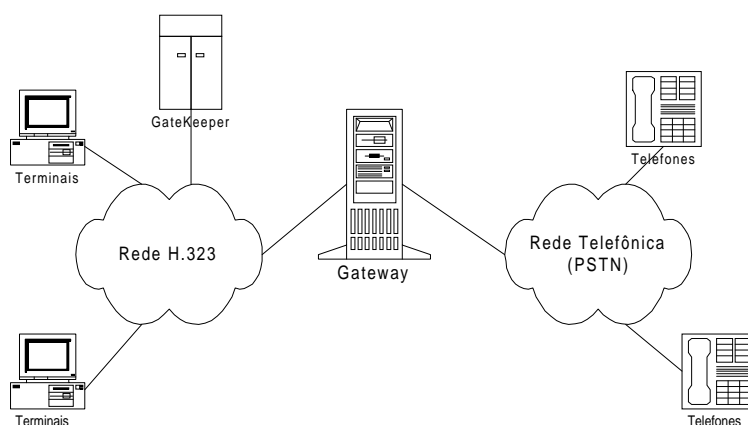


Figura 2.3.3.A – Arquitetura H.323

A arquitetura H.323 também define um conjunto de protocolos de comunicação e padrões de codificação que devem ser utilizados na comunicação multimídia. A Figura 2.3.3.B ilustra o Modelo de Referência do H.323.

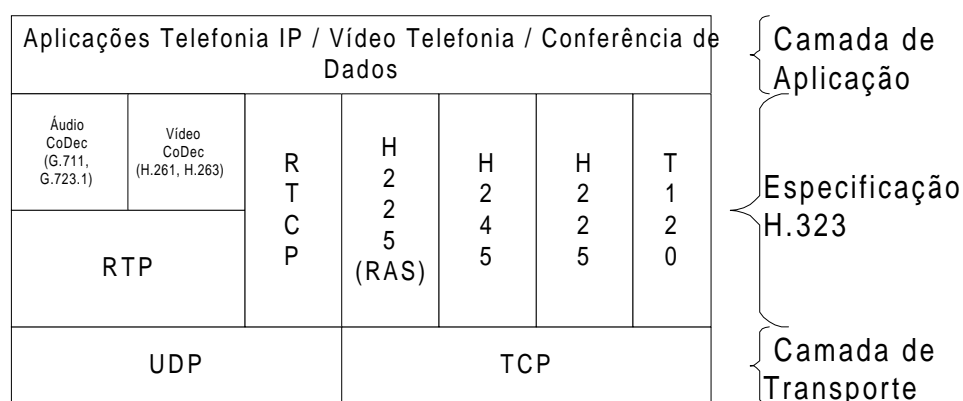


Figura 2.3.3.B – Modelo de Referência H.323

Conforme ilustra a Figura 2.3.3.B, a especificação H.323 utiliza: 1- um conjunto de codificadores de áudio (G.711, G.722, G.723.1, G.728 e G.729) e vídeo (H.261 e H.263); 2- o RTP para transportar mídias contínuas no tempo; 3- o RTCP para monitorar os parâmetros de QoS da transmissão multimídia; 4- o H.225 RAS⁷ (*Registration, Admission, Status*) para permitir a comunicação entre pontos finais (terminais e gateways) e o *gatekeeper*. A Tabela 2.3.3.A descreve as suas mensagens; 5- o H.225⁸ para iniciar e finalizar uma chamada entre dois pontos finais. A sinalização de chamadas H.225 é trocada diretamente entre os pontos participantes de uma chamada quando não existe *gatekeeper*. Quando existe, as mensagens podem ser roteadas através dele. A Tabela 2.3.3.B descreve as suas mensagens; 6- o H.245 para trocar mensagens de controle entre os pontos finais H.323. Este protocolo fornece as seguintes funcionalidades: a- capacidade de troca, permitindo que os pontos finais H.323 tenham diferentes capacidades de transmissão e recepção; e b - abertura e fechamento de canais lógicos, permitindo a criação de diversas conexões para a transferência de *stream* de dados. A Tabela 2.3.3.C descreve as suas mensagens; 7- o protocolo T.120 para oferecer suporte à conferência de dados em tempo real, como transferência de arquivos e transmissão de fax. A Tabela 2.3.3.D lista os codificadores e protocolos de comunicação definidos no Modelo de Referência H.323 [60].

⁷ O uso deste protocolo só é necessário quando existe um componente *gatekeeper* no ambiente de comunicação.

⁸ Utiliza um subconjunto do protocolo de sinalização Q.931 para este propósito.

Tabela 2.3.3.A – Mensagens H.225 RAS

Mensagem	Descrição
RegistrationRequest (RRQ)	Requisição de um ponto terminal para se registrar em um gatekeeper.
AdmissionRequest (ARQ)	O terminal requisita o acesso à rede de pacotes.
BandwidthRequest (BRQ)	O terminal requisita ao gatekeeper que seja feita uma alteração na alocação de banda.
DisengageRequest (DRQ)	Requisição para finalizar chamada.
InfoRequest (IRQ)	Requisição de informações de estado.
InfoRequestResponse (IRR)	Resposta a uma mensagem IRQ.
RAS Timers and Request in Progress (RIP)	Valor padrão de timeout para resposta a mensagens RAS e subsequente reenvio da mensagem se a resposta não é recebida.

Tabela 2.3.3.B – Mensagens H.225

Mensagem	Descrição
Alerting	Usuário destino está sendo alertado sobre o recebimento de uma chamada.
Call Proceeding	Requisição de estabelecimento de chamada sendo iniciado.
Connect	Chamada sendo processada foi aceita pelo usuário destino.
Setup	Indica que uma entidade origem deseja iniciar uma conexão com uma entidade destino.
Release Complete	Indica a liberação de uma chamada se o canal de sinalização H.225.0 (Q.931) estiver aberto.
Status	Resposta a uma mensagem de sinalização desconhecida.
Status Inquiry	Requisita informações da chamada.

Vários estudos tem sido desenvolvidos para permitir que pontos finais H.323 estabeleçam uma conexão especificando os parâmetros de QoS desejados. Na recomendação H.323 v4, anexo C (*Annex C - H.323 on ATM*), está detalhado o uso do protocolo de transporte AAL5 para permitir o estabelecimento de circuitos virtuais com QoS em redes ATM. Maiores detalhes podem ser encontrados em [58]. No apêndice II desta recomendação (*APPENDIX II - Transport Level Resource Reservation Procedures*), está descrito como o padrão H.323 pode incorporar o protocolo de reserva de recursos RSVP no seu Modelo de Referência, para permitir o transporte de dados multimídia na Internet com garantias de qualidade. Utilizando o protocolo RSVP, terminais H.323 podem estabelecer canais de comunicação com a QoS especificada. A mensagem Open Logical Channel do H.245 possui o atributo `qOSCapability` que determina quais os parâmetros de QoS desejados para o estabelecimento da conexão. Maiores detalhes podem ser encontrados em [58].

Tabela 2.3.3.C – Mensagens H.245

Mensagem	Descrição
Master-Slave Determination	Determina qual terminal é o mestre e qual é o escravo.
Terminal Capability Set	Contém informações sobre as capacidades (atributos) de um terminal para transmitir e receber fluxos multimídia.
Open Logical Channel	Abre um canal lógico para transporte de informações de dados, áudio e vídeo.
Close Logical Channel	Fecha um canal lógico entre dois pontos.
Request Mode	Utilizado por um receptor para requisitar modos particulares de transmissão a um transmissor.
Send Terminal Capability Set	Solicita que um terminal confirme que está recebendo e transmitindo capacidades através do envio de uma ou mais mensagens Terminal Capability Sets.
End Session Command	Indica o fim de uma sessão H.245. Após a transmissão desta mensagem, o terminal não enviará mais nenhuma mensagem H.245.

Tabela 2.3.3.D – Padrões H.323

Número da Recomendação	Título (Data de Aceitação)
Codificadores de Áudio	
G.711	Codificador de voz (11/1988).
G.723.1	Codificador de voz para aplicações multimídia transmitindo entre 5.3 e 6.3 Kbps (03/1996).
G.728	Codificador de voz a taxas de 16 kbps usando baixo retardo e predição linear (09/1992).
G.729	Codificador de voz a taxas de 8 kbps usando codificação algébrica e predição linear (03/1996).
Codificadores de Vídeo	
H.261	Codificadores de vídeo a taxas de p x 64 kbps (03/1993).
H.263	Codificação de vídeo para comunicações com baixa taxa de Dados (02/1998).
Conferência de Dados	
T.120	Protocolo para conferências multimídia (07/1996).
Controle	
H.245	Protocolo de controle para comunicação multimídia (09/1998).
H.225.0	Protocolos para sinalização de chamadas e empacotamento de dados para sistemas de comunicação multimídia baseados em pacotes (02/1998).
Transporte em tempo real	
RTP	Protocolo de transporte em tempo real.
RTCP	Protocolo de controle de tempo real.

Como principais limitações do H.323, podemos destacar: 1- especificação grande e complexa, permitindo diferentes interpretações, e, conseqüentemente, possibilitando o desenvolvimento de diversas implementações do padrão que podem ser incompatíveis; e 2- para estabelecer conexões com garantias de qualidade, os terminais H.323 devem ter conhecimento dos parâmetros de QoS nível de transporte especificados pelo protocolo RSVP.

2.4 – Interfaces de Comunicação com QoS

Após o estudo dos principais protocolos de comunicação hipermídia e multimídia relacionados a este trabalho, é importante analisar como é feita a solicitação de serviços especificando os parâmetros de QoS desejados. Em [33, 35, 36] foi desenvolvida uma API (*Application Program Interface*) com QoS para a solicitação de serviços parametrizando os requisitos de qualidade desejados em um *middleware*. Em [43] foi desenvolvida uma API com QoS para que as aplicações de trabalho cooperativo (CSCW) requisitassem serviços com os parâmetros de qualidade desejados. As próximas duas seções descrevem esses trabalhos.

2.4.1 - API com QoS do *Middleware* Da CaPo++

O *middleware* Da CaPo++ [33] foi desenvolvido para oferecer suporte de comunicação às aplicações multimídia, tais como: videofonia, video-conferência, tele-seminário, servidor multimídia, entre outras. Foi projetado no Instituto Federal de Tecnologia da Suíça - *Swiss Federal Institute of Technology Zurich* (ETHZ).

O Da CaPo++ especifica uma API com QoS, onde as aplicações multimídia solicitam serviços ao *middleware* especificando os parâmetros de qualidade desejados. Baseado na QoS solicitada, diversos protocolos de comunicação podem ser utilizados. A arquitetura deste *middleware* está ilustrada na Figura 2.4.1.A.

Para oferecer simplicidade de uso, a API com QoS desenvolvida utiliza as abstrações de sessão e fluxo de dados, apresentadas na Seção 2.3.2.



Figura 2.4.1.A – Arquitetura Da CaPo++

Para ser possível a transferência de informações entre as aplicações multimídia que utilizam o Da CaPo++, deve-se criar uma sessão entre as mesmas, utilizando-se para isto a primitiva de serviço *Session()*. Os fluxos de informações desejados serão criados nesta sessão, utilizando-se para isto a primitiva de serviço *FlowJoin()*. Os requisitos de QoS de cada fluxo criado serão definidos através do uso da primitiva *SetReqFlow(QoS)*. A Tabela 2.4.1 descreve as primitivas definidas na API com QoS do *middleware* Da CaPo++ [33, 35, 36]. Outra forma utilizada para a criação de sessão e fluxos de dados no *middleware* é através do uso de arquivos de configuração, selecionados pela aplicação do usuário quando se desejar utilizar um determinado serviço de comunicação no *middleware*.

Por exemplo, para ser possível utilizar uma aplicação de videofonia no middleware Da CaPo++, uma sessão unicast deverá ser criada entre a aplicação cliente e a aplicação servidora. Quatro fluxos de dados simplex serão criados: dois fluxos de áudio e dois fluxos de vídeo (um para cada direção). A Figura 2.4.1.B [33] ilustra um arquivo de configuração utilizado para a criação da sessão com quatro fluxos.

É importante ressaltar que qualquer aplicação só pode solicitar serviços ao *middleware* Da CaPo++ após ter sido autenticado pelo mesmo e ter recebido a chave de autenticação (*secret_key*).

A principal limitação observada é que a API com QoS deste *middleware* não define uma primitiva para indicar possíveis violações da QoS estabelecida.

Tabela 2.4.1 - Primitivas da API

Primitiva de Serviço	Descrição
Session()	Solicita a criação de uma sessão.
Connect()	Conecta a sessão a uma porta e um endereço (unicast ou multicast).
Configure()	Configura cada fluxo pertencente a uma mesma sessão.
Activate()	Inicia o transporte de dados de um fluxo pertencente a uma sessão.
Deactivate()	Finaliza o transporte de dados de um fluxo pertencente a uma sessão.
FlowJoin()	Cria um novo fluxo em uma sessão.
FlowLeave()	Elimina um fluxo de uma sessão.
Close()	A sessão é finalizada e todos os recursos são desalocados.
GetFlowDescriptor()	Retorna o identificador de um fluxo.
SetReqFlow()	Define os parâmetros de QoS de um fluxo.
GetReqFlow()	Retorna os parâmetros de QoS de um fluxo.
SendDataFlow()	Envia dados do usuário em um determinado fluxo.
RecvDataFlow()	Recebe dados do usuário em um determinado fluxo.
SendCtrlFlow()	Envia informações de controle em um determinado fluxo.
RecvCtrlFlow()	Recebe informações de controle em um determinado fluxo.

```

SESSION CREATOR UNICAST PicturePhone;
FLOW VIDEO_SEND_DEVICE VideoOut;
    THROUGHPUT 4.5;
    FPS 5;
    DELAY 0.2;
ENDFLOW;
FLOW AUDIO_SEND_DEVICE AudioOut;
    THROUGHPUT 1.4;
    DELAY 0.1;
ENDFLOW;
FLOW VIDEO_RECV_DEVICE VideoIn;
    THROUGHPUT 4.5;
    FPS 5;
    DELAY 0.2;
ENDFLOW;
FLOW AUDIO_RECV_DEVICE AudioIn;
    THROUGHPUT 1.4;
    DELAY 0.1;
ENDFLOW;
ENDSESSION;

```

Figura 2.4.1.B - Exemplo de um Arquivo de Configuração

2.4.2 - API com QoS para Aplicações de Trabalho Cooperativo (CSCW)

Esta interface de comunicação com QoS permite a integração de aplicações CSCW em diversos sistemas de comunicação, e foi projetada por pesquisadores do Instituto Técnico de Lisboa e da Universidade de Coimbra [43].

Esta proposta utilizou as abstrações de sessão de cooperação, sessão de transferência de informação, sessão de transporte de dados e sessão de sinalização, visando oferecer simplicidade de uso às aplicações CSCW que utilizam a API de comunicação.

A **sessão de cooperação** está associada ao transporte de informações de um determinado tipo de mídia. Este transporte é realizado pelas **sessões de transferência de informações**. Cada sessão de transferência é identificada por um endereço IP *multicast* e por um valor de porta. Cada sessão de transferência possui uma **sessão de transporte** e duas **sessões de sinalização**. A sessão de transporte é implementada utilizando-se o protocolo de transporte RTP. A sessão RTP é identificada pelo mesmo endereço IP *multicast* e porta definida na sessão de transferência de informações. A transferência de informações de controle é realizada pelo protocolo RTCP, utilizando-se o mesmo endereço IP *multicast* e porta + 1. A sessão de sinalização é implementada utilizando o protocolo de reserva de recursos RSVP. A Figura 2.4.2.A ilustra as abstrações de sessão definidas naquele trabalho.

As primitivas de serviço definidas na API com QoS estão descritas na Tabela 2.4.2. Esta API foi implementada pelo módulo 'Gestor de Ligações', responsável por receber e processar todos os pedidos que chegam à interface. Este módulo é um protocolo de comunicação que implementa as primitivas definidas na API. Este módulo utiliza os serviços oferecidos por outros dois módulos: Controlador de QoS e Conversor de QoS. A Figura 2.4.2.B ilustra a arquitetura especificada.

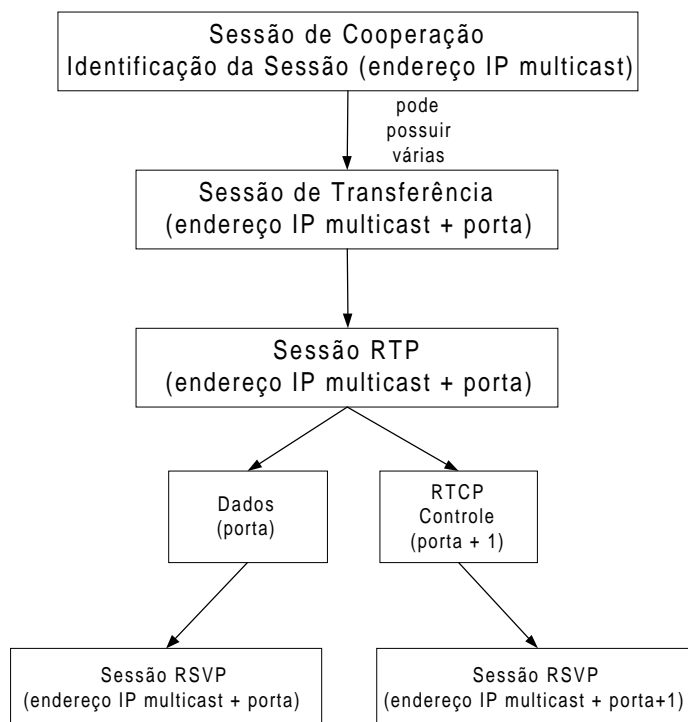


Figura 2.4.2.A - Abstrações de Sessão

Tabela 2.4.2 - Primitivas da API

Primitiva de Serviço	Descrição
Cria_Sessão()	Cria uma sessão de cooperação. Em consequência, uma sessão de transferência de informações é criada.
Cliente_Emissor()	Indica que o cliente é o transmissor da informação.
Cliente_Receptor()	Indica que o cliente é o receptor da informação.
Envia_Dados()	Utilizado para enviar dados.
Recebe_Dados()	Utilizado para receber dados.
Emissor_rec_QoS()	Define quais os parâmetros de QoS utilizados pelo transmissor.
Receptor_rec_QoS()	Indica quais os parâmetros de QoS utilizados pelo receptor.
Emissor_adpt_QoS()	Comunica ao transmissor qualquer alteração que haja no nível de qualidade desejado.
Sair_Sessão()	Finaliza a sessão de cooperação.

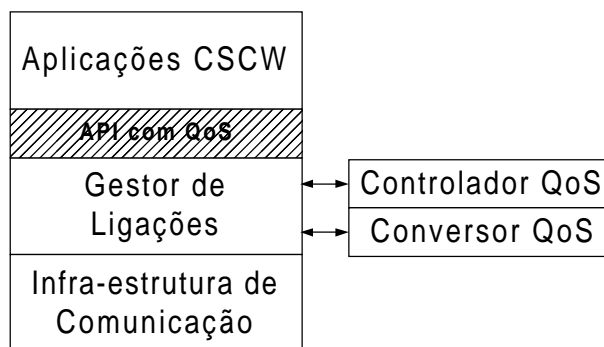


Figura 2.4.2.B - Arquitetura Definida em [43]

O módulo 'Controlador de QoS' tem como objetivo principal a detecção no emissor, de alterações do nível de qualidade requerido pelo receptor, para uma dada sessão de cooperação. Este módulo efetua a monitoração dos níveis de qualidade, no emissor, a partir das informações de controle oferecidas pelo protocolo RTCP.

O módulo 'Conversor de QoS' é responsável por mapear os parâmetros de QoS do nível de aplicação para o nível de transporte. Utilizou-se o mecanismo de mapeamento de parâmetros de qualidade definidos por Klara Nahrstedt [43]. A Figura 2.4.2.C ilustra a regra de mapeamento utilizada.

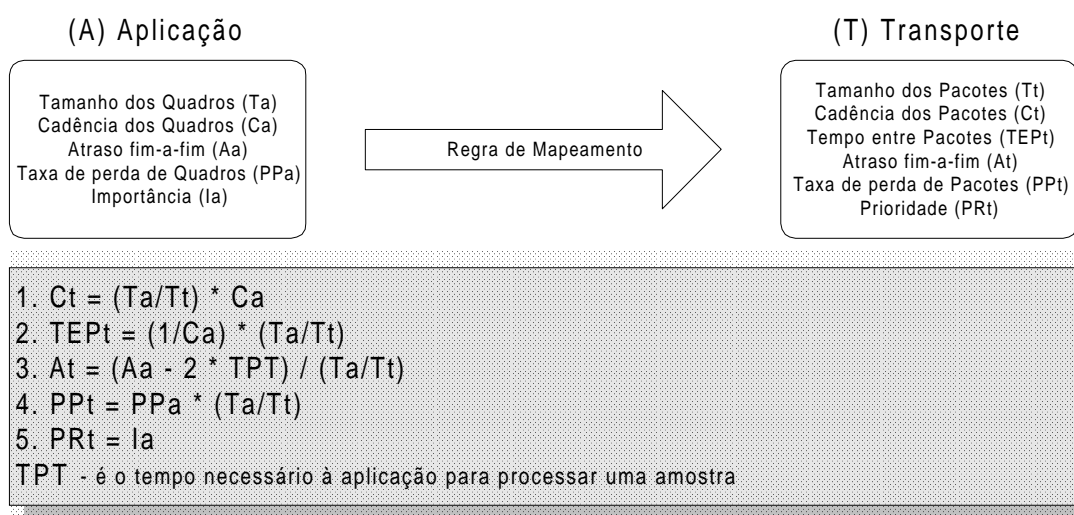


Figura 2.4.2.C – Mapeamento de Parâmetros de QoS [43]

T_t (Tamanho dos Pacotes) é um parâmetro específico do protocolo de transporte utilizado. Se houver necessidade de fragmentar um quadro em vários pacotes, isto é, T_a

(Tamanho do Quadro) ser maior que T_t , o cálculo do A_t (Atraso fim-a-fim) será diretamente influenciado, visto que aumenta-se o tempo necessário para processar as informações.

Como principal limitação, podemos destacar que os conceitos de sessão de transferência de informações e sessão de transporte são semelhantes. Nesta caso, para facilitar o uso da API, deveria ser definida uma única abstração que fosse genérica o suficiente para representar o transporte de informações.

3

ESPECIFICAÇÃO DO PROTOCOLO

Este Capítulo apresenta o protocolo de comunicação hipermídia com QoS e Modularidade Funcional. Inicialmente, as principais características do protocolo, seu Modelo de Referência, as suas quatro unidades funcionais, sua sintaxe e sua semântica são apresentados. Em seguida, o tratamento de QoS no protocolo hipermídia é definido.

3.1- Modelo de Referência do Protocolo de Comunicação Hipermídia

O Modelo de Referência do protocolo de comunicação hipermídia proposto está ilustrado na Figura 3.1. Para facilitar o seu entendimento, resumimos e exemplificamos as etapas realizadas pelo protocolo:

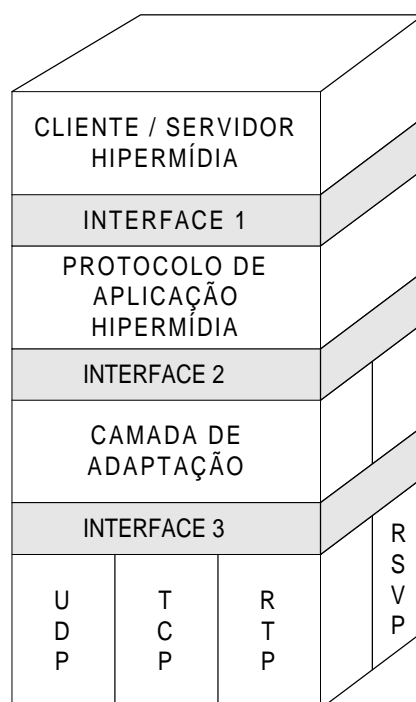


Figura 3.1 – Modelo de Referência do Protocolo de Comunicação Hipermídia

1- as **aplicações hipermídia (cliente e servidor)** solicitam serviços ao protocolo hipermídia, especificando os parâmetros de QoS desejados, utilizando primitivas definidas na interface 1. A especificação completa desta interface (API) pode ser encontrada no Apêndice A.

Por exemplo, a aplicação cliente solicita um vídeo especificando os parâmetros de qualidade desejados (frame_rate - número de quadros por segundo, frame_size - tamanho do vídeo, resolution - resolução desejada, video_codification - codificação utilizada) e um texto com qualquer qualidade (melhor-esforço). Na solicitação do vídeo, utiliza-se a primitiva HYP-Reserve (... , vídeo, frame_rate, frame_size, resolution, video_codification). No caso do texto, utiliza-se a primitiva HYP-Get (... , texto).

2- o **protocolo hipermídia** recebe as requisições de serviço das aplicações, analisa essas requisições, analisa as especificações de QoS, mapeia as especificações de QoS ao nível da aplicação para o nível dos protocolos de transporte e solicita serviços de comunicação à camada de adaptação, usando primitivas definidas na interface 2.

Continuando o exemplo anterior, o protocolo ao receber a requisição de um vídeo com qualidade (frame_rate, frame_size, resolution, video_codification), mapeia estes parâmetros para os de nível de transporte (peakBandwidth - taxa de pico, tokenBucketSize - tamanho do bucket, tokenRate - taxa do bucket).

No caso do documento texto com qualquer qualidade (melhor-esforço), não será realizado nenhum mapeamento.

3- a **camada de adaptação** recebe as requisições de serviços de transporte do protocolo hipermídia; analisa essas requisições; reserva (se for o caso), através da utilização de um protocolo específico, os recursos necessários para garantir o transporte da informação com a QoS especificada; seleciona o protocolo de transporte disponível que mais se adequie ao tipo de informação a ser transportada; e solicita serviços de comunicação aos protocolos de transporte, utilizando primitivas definidas na interface 3.

Seguindo o exemplo anterior, a camada de adaptação recebe a solicitação de um vídeo com os parâmetros de QoS nível de transporte (peakBandwidth, tokenBucketSize, tokenRate) e requisita, ao protocolo RSVP, a reserva de todos os recursos necessários para a transmissão do vídeo com os requisitos de QoS especificados. Após a reserva dos recursos pelo RSVP, seleciona o protocolo RTP como o mais adequado para o recebimento do vídeo.

No caso do documento texto, não é realizada nenhuma reserva de recursos, visto que a QoS solicitada (melhor-esforço) é oferecida pela Internet e seleciona o protocolo TCP como o mais adequado para o recebimento do documento texto.

Para validar as idéias propostas neste trabalho, foram utilizados os protocolos de transporte TCP, UDP, RTP e o protocolo para reserva de recursos RSVP. É importante ressaltar que o Modelo de Referência, apresentado na Figura 3.1 com esses protocolos, pode utilizar outros protocolos de comunicação, tais como os definidos no padrão ATM e os definidos pelo ITU que estão especificados no padrão H.323. Neste último caso, o Modelo de Referência utilizaria os protocolos TCP, UDP, RTP, T.120 e o protocolo H.225 (RAS) para a reserva de recursos. No caso do padrão ATM, o Modelo de Referência utilizaria as AALs de transmissão (AAL1, AAL2, AAL3/4 e AAL5) e o plano de sinalização ATM.

4- o **protocolo de transporte utilizado** recebe indicações da chegada de informações e utiliza um protocolo de comunicação responsável por realizar o processo de monitoração de QoS das informações recebidas.

Dando continuidade ao exemplo anterior, o protocolo RTP irá receber indicações da chegada de pacotes de vídeo e utilizará o protocolo RTCP⁹ para monitorar quais são os parâmetros de qualidade da transmissão do vídeo solicitado. Os parâmetros de QoS monitorados são: 1. número de pacotes perdidos; 2. número de pacotes fora de ordem; e 3. número de pacotes recebidos.

⁹ É importante ressaltar que o RTCP faz parte do RTP, por isso não foi especificado no Modelo de Referência do protocolo de comunicação hipermídia.

3.2- Especificação do Protocolo de Comunicação

A especificação de um protocolo de comunicação requer, segundo [2], as seguintes seis etapas:

1. ambiente de oferecimento de serviços, **para descrever o ambiente de oferecimento de serviços (AOS) onde o protocolo é utilizado;**
2. especificação do serviço, **para determinar os serviços de comunicação oferecidos pelo protocolo;**
3. especificação do serviço de nível inferior utilizado, **para determinar os serviços de comunicação usados pelo protocolo;**
4. vocabulário do protocolo, **para definir todas as suas mensagens;**
5. formato das mensagens (sintaxe), **para determinar como cada mensagem do protocolo é codificada;**
6. regras do protocolo (semântica), **para descrever a sua máquina de estados.**

Essas etapas estão descritas nas seções a seguir.

3.2.1- Ambiente de Oferecimento de Serviços (AOS)

O AOS é constituído de: (i) **usuários**, que correspondem a entidades que utilizam diretamente os serviços, e (ii) **provedores**, que são os responsáveis pela disponibilização dos serviços durante sua utilização.

A Figura 3.2.1.A ilustra um AOS que possui duas entidades e um provedor de serviços, utilizando os conceitos, princípios e terminologia adotados no Modelo de Referência Unificado (SCM - *Service-Composition Model*) [12].

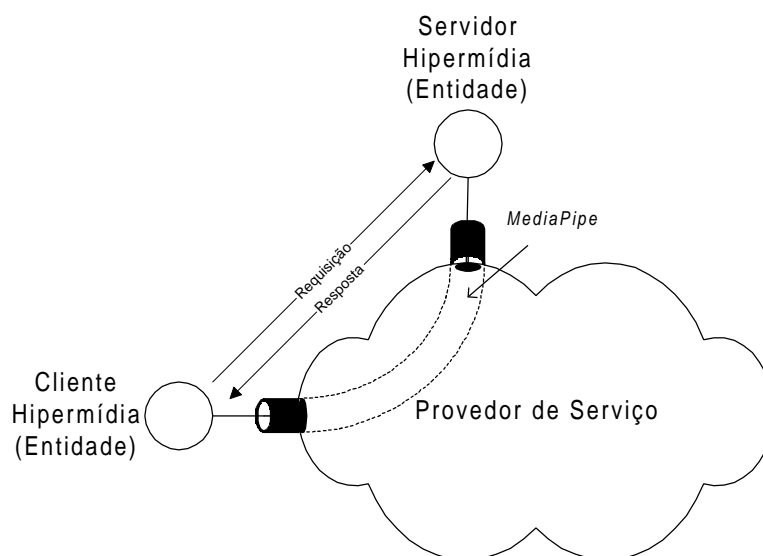


Figura 3.2.1.A – Exemplo de Ambiente de Oferecimento de Serviços

A entidade cliente solicita serviços enviando mensagens de requisição do protocolo hipermedia. A entidade servidora responde a solicitação de serviços enviando mensagens de resposta. As trocas de informações (mensagens ou *stream* de dados¹⁰) só são possíveis após a criação de *pipes* de comunicação. Um *pipe* representa todos os componentes (roteadores, comutadores, repetidores, provedores de infra-estrutura) que fazem parte do "caminho" fim-a-fim entre a entidade cliente e a entidade servidora hipermedia.

As entidades (aplicações do usuário) podem solicitar serviços ao protocolo hipermedia especificando a QoS desejada. Neste caso, torna-se necessário a criação de *pipes* que atendam aos parâmetros de QoS definidos pela aplicação. Estes *pipes* são conhecidos como *MediaPipes* [12].

Uma entidade pode solicitar ao protocolo hipermedia diversos serviços com requisitos de QoS diferentes, tornando necessário a criação de diversos *MediaPipes*. O protocolo de comunicação hipermedia proposto oferece a possibilidade de criação de diversos *MediaPipes*, o que significa reservar todos os recursos necessários para que uma aplicação funcione com um desempenho aceitável. *MediaPipes* são criados pelo

¹⁰ A principal diferença entre mensagens e *stream* de dados é que na primeira as informações estão delimitadas em blocos de tamanho determinado. No caso de *stream* de dados, as informações possuem tamanho indeterminado, exigindo tratamento temporal e contínuo [12].

protocolo hipermídia quando a aplicação do usuário solicita o envio ou recebimento de *stream* de dados, utilizando para isso as primitivas de serviço: *HYP_CreateEntity*, *HYP_GetEntityProperties*, *HYP_Get* e *HYP_Reserve* (ver Seção 3.2.2). A Figura 3.2.1.B ilustra um AOS com dois *MediaPipes*, um para mídia áudio e outro para mídia texto.

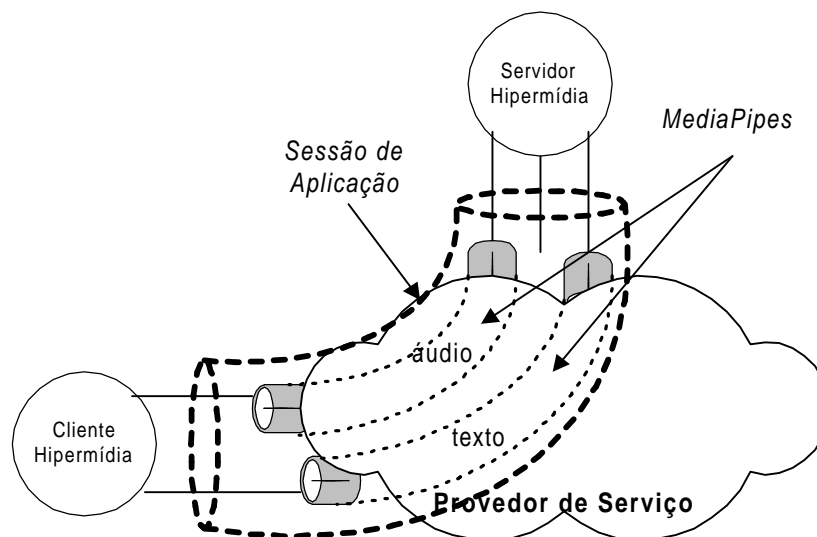


Figura 3.2.1.B – AOS com dois MediaPipes: para áudio e texto

Para facilitar o gerenciamento de diversos *MediaPipes* criados entre as mesmas entidades cliente e servidora, utilizou-se o conceito de **Sessão de Aplicação** [35, 36] para agrupar diversos *MediaPipes*. Por uma questão de segurança, uma aplicação do usuário só pode ter acesso ao(s) *MediaPipe(s)* criado(s) na sua Sessão de Aplicação.

O protocolo proposto oferece a possibilidade de criação de diversas Sessões de Aplicação agrupando vários *MediaPipes* que reservem todos os recursos necessários para que uma aplicação funcione com um desempenho aceitável.

O cliente hipermídia só pode solicitar serviços de um servidor, após criar uma Sessão de Aplicação com o mesmo. Utilizando a primitiva *HYP_OpenSession*, o cliente solicita ao protocolo hipermídia a criação de uma Sessão de Aplicação. Essa Sessão só é criada após

o cliente hipermídia ter sido autenticado¹¹ pelo servidor. A Figura 3.2.1.C mostra as características da Sessão de Aplicação criada para a situação da Figura 3.2.1.B.



Figura 3.2.1.C – Sessão de Aplicação com dois *MediaPipes*

A Figura 3.2.1.C foi obtida utilizando o protótipo de comunicação desenvolvido em [53]. Maiores detalhes sobre o protótipo desenvolvido serão discutidos no Capítulo 4. O cliente hipermídia, identificado por *tradicao*, estabelece uma Sessão de Aplicação com um servidor hipermídia, identificado por *imperio*. A aplicação cliente possui dois *MediaPipes* criados com o servidor. O *MediaPipe texto (Control)* é utilizado para a troca de mensagens do protocolo. O *MediaPipe áudio (Data)* é utilizado para enviar e receber informações, como por exemplo, solicitar o recebimento de um áudio em tempo-real.

Formalmente, define-se qualidade de serviço (q) em um *MediaPipe* através da seguinte expressão [15]:

$$q = (r_1, \dots, r_n) \in \mathbf{R} = R_1 \times R_2 \times \dots \times R_n,$$

¹¹ O processo de autenticação verifica se o cliente pode ter acesso aos serviços oferecidos por um servidor.

com R sendo o conjunto de recursos disponíveis (memória, banda passante, cpu), e as n-tuplas (r_1, \dots, r_n) como sendo os parâmetros específicos de qualidade de serviço.

O perfil de QoS de uma Sessão de Aplicação é representado pela seguinte expressão:

$$Q(\text{sessão_aplicação}) = (q_1, \dots, q_m),$$

onde cada $MediaPipe_k$ $k=1, \dots, m$ de uma sessão possui os seus próprios requisitos de qualidade $q_k, k=1, \dots, m$.

Por exemplo, o perfil de QoS de uma Sessão de Aplicação com um MediaPipe Áudio é representado pela seguinte expressão:

$$Q(a) = (q_{\text{áudio}}),$$

onde:

$$q_{\text{áudio}} = (r_{\text{memoria}}, r_{\text{cpu}}, r_{\text{banda_passante}}) \in \mathbf{R} = R_{\text{host}} \times R_{\text{roteadores}}$$

3.2.2- Especificação do Serviço

O principal objetivo do protocolo de comunicação hipermídia é oferecer um serviço de transferência de documentos com QoS. Este serviço pode ser *unicast* (um transmissor e um receptor) ou *multicast* (um transmissor e vários receptores); pode ser *half-duplex* (transferência de informações nos dois sentidos, porém apenas uma por vez) ou *full-duplex* (transferência de informações nos dois sentidos simultaneamente); e a troca de mensagens é síncrona (o transmissor envia uma requisição e aguarda uma resposta).

O protocolo hipermídia proposto possui quatro unidades funcionais: **genérico, controle de acesso, conteúdo e qualidade de serviço**. A Tabela 3.2.2 descreve as primitivas de serviço definidas pelas unidades funcionais do protocolo hipermídia.

Tabela 3.2.2 – Descrição das Primitivas de Serviço das Unidades Funcionais

Primitivas de Serviço	Descrição
Unidade Funcional Genérico	
HYP_OpenSession	Abre uma Sessão de Aplicação
HYP_CloseSession	Fecha uma Sessão de Aplicação
HYP_SessionStatus	Consulta o <i>status</i> de um Sessão
HYP_CreateEntity	Cria uma Entidade
HYP_CreateEntityProperty	Cria uma propriedade ¹² de uma Entidade
HYP_GetEntityProperties	Consulta às propriedades de uma Entidade
HYP_GetEntityProperty	Consulta à propriedade de uma Entidade
HYP_SetEntityProperty	Modifica a propriedade de uma Entidade
HYP_Error	Indica a ocorrência de Erro
Unidade Funcional Controle de Acesso	
HYP_LockEntityProperty	Bloqueia o acesso às propriedades de uma Entidade
HYP_UnLockEntityProperty	Desbloqueia o acesso às propriedades de uma Entidade
HYP_IsLocked	Verifica o tipo de acesso a uma propriedade
HYP_GetEntityPermission	Consulta o tipo de permissão de uma Entidade
HYP_SetEntityPermission	Modifica o tipo de permissão de uma Entidade
HYP_GetPropertyPermission	Consulta o tipo de permissão de uma propriedade de uma Entidade
HYP_SetPropertyPermission	Modifica o tipo de permissão de uma propriedade de uma Entidade
Unidade Funcional Conteúdo	
HYP_Get	Solicita a transferência de um documento
HYP_Pause	Solicita a parada, momentânea, da transferência de um documento
HYP_Resume	Solicita o reinício de uma transmissão
HYP_Stop	Solicita o término da transferência de um documento
Unidade Funcional Qualidade de Serviço	
HYP_GetQoS	Solicita a transferência de um documento com garantias de qualidade
HYP_Reserve	Solicita a reserva de todos os recursos necessários para a criação do <i>MediaPipe</i> com a QoS especificada
HYP_RenegotiateQoS	Renegocia a QoS especificada para a transferência de um documento
HYP_AlertQoS	Indica que a QoS especificada para a transferência de um documento não está sendo oferecida pelo provedor de serviços

A unidade funcional **Genérico** (*Kernel*) oferece primitivas de serviço para abrir e fechar uma Sessão de Aplicação; consultar o *status* da Sessão; criar Entidade; criar propriedades

¹² Toda Entidade possui um conjunto de propriedades que representam seus atributos. Por exemplo, uma Entidade Áudio possui como propriedades: codificação, tamanho, duração, entre outros.

de uma Entidade; consultar as propriedades de uma determinada Entidade; modificar as propriedades de uma determinada Entidade; e indicar a ocorrência de qualquer erro.

A unidade funcional **Controle de Acesso** (*Access Control*) define primitivas para controlar o acesso às propriedades de uma determinada Entidade; verificar o tipo de acesso a uma determinada propriedade; solicitar o tipo de permissão de uma Entidade; definir o tipo de permissão de uma Entidade; solicitar o tipo de permissão de uma propriedade de uma Entidade; e definir o tipo de permissão de uma propriedade de uma Entidade.

A unidade funcional **Conteúdo** (*Content*) define primitivas para iniciar e finalizar a transferência de informações.

A unidade funcional **Qualidade de Serviço** (*QoS - Quality of Service*) define primitivas para solicitar a transferência de informações com garantias de QoS; renegociar e monitorar os parâmetros de QoS estabelecidos.

É importante ressaltar que qualquer aplicação que utilize o protocolo hipermídia deve usar a unidade funcional **Genérico**. As outras unidades funcionais são usadas opcionalmente conforme negociação estabelecida durante a criação de uma Sessão de Aplicação entre o cliente e o servidor hipermídia.

Utilizando uma especificação orientada a objetos, pode-se fazer o uso do padrão de projeto (*design pattern*¹³) *Composite* [39], para definir o oferecimento das mais variadas combinações de unidades funcionais no protocolo de comunicação hipermídia especificado, conforme ilustrado na Figura 3.2.2.A.

¹³ *Design Patterns* podem ser vistos como uma forma de documentar conjuntos de regras que descrevem decisões de projetos recorrentes em vários sistemas, facilitando assim a reutilização de soluções já existentes [14].

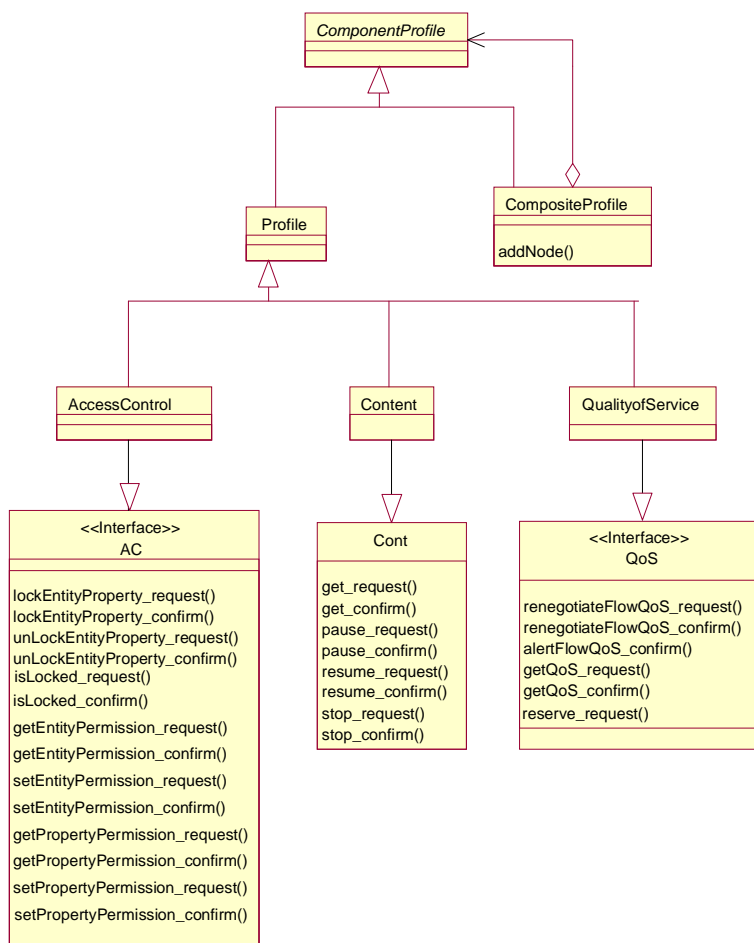


Figura 3.2.2.A – *Pattern Composite* adaptado para o Protocolo Hipermídia

A Figura 3.2.2.A ilustra o *pattern Composite* adaptado para definir as unidades funcionais oferecidas pelo protocolo, utilizando notação gráfica baseada na linguagem UML (*Unified Modeling Language*) [38]. Na descrição textual, as classes e métodos abstratos são representados em itálico. As sub-classes *AccessControl*, *Content* e *QualityofService* representam as três unidades funcionais opcionais oferecidas pelo protocolo hipermídia. Cada sub-classe implementa um conjunto de primitivas da API do protocolo hipermídia.

Por exemplo, uma aplicação cliente pode negociar o uso das unidades funcionais Conteúdo e Qualidade de Serviço. Neste caso, a aplicação passa a ter acesso a novas primitivas da interface do protocolo, além das primitivas definidas na unidade funcional Genérico.

No Apêndice A, todas as primitivas de serviço, definidas na interface do protocolo hipermídia, estão especificadas, usando a seguinte expressão geral [50]:

HYP-Serviço.tipo (parâmetros)

onde **Serviço** representa a primitiva utilizada; **tipo** indica o tipo de primitiva (*request*, *indication*, *response* e *confirm*); e **parâmetros** indicam os atributos utilizados nas primitivas.

Por exemplo, para iniciar uma Sessão de Aplicação, o cliente hipermídia (localizado na camada N+1) solicita serviços de comunicação ao protocolo hipermídia (camada N), através do envio da primitiva HYP_OpenSession.req (1), conforme ilustrado na Figura 3.2.2.B. Ao receber esta solicitação de serviço, o protocolo hipermídia cliente envia ao protocolo hipermídia servidor a mensagem OpenSession.xml (2).

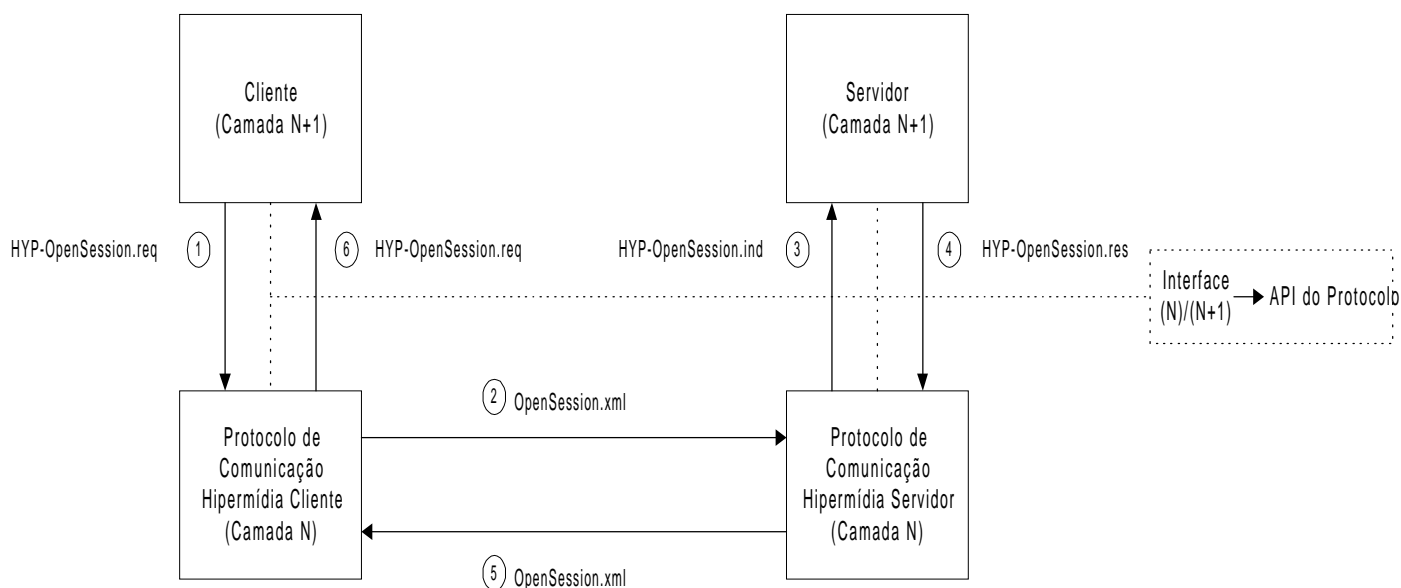


Figura 3.2.2.B - Troca de Mensagens para Abrir uma Sessão de Aplicação

O protocolo hipermídia servidor, ao receber a mensagem OpenSession.xml, envia a primitiva HYP_OpenSession.ind (3), indicando ao Servidor, camada N+1, a solicitação de um serviço para abertura de uma Sessão de Aplicação. O Servidor realiza o serviço solicitado e, através da primitiva (HYP_OpenSession.res)(4), retorna sua resposta ao protocolo hipermídia servidor. Este, ao receber a primitiva, envia a mensagem

OpenSession.xml (5) para o cliente, confirmando a criação da Sessão de Aplicação solicitada. O protocolo hipermídia cliente, ao receber a mensagem de confirmação, *OpenSession.xml*, envia a primitiva *Hyp_OpenSession.cnf* (6) ao **Cliente** hipermídia, indicando que o serviço solicitado para abertura de uma Sessão de Aplicação foi realizado com sucesso.

3.2.3- Especificação do Serviço de Nível Inferior Utilizado

Conforme foi discutido na Seção 3.1, o Modelo de Referência do protocolo de comunicação hipermídia define uma camada de adaptação que é responsável por determinar qual o serviço de transporte, confiável ou não-confiável, a ser utilizado pelo protocolo especificado. Este serviço será escolhido dependendo do tipo de informação a ser transportada. *Por exemplo, para transportar um áudio em tempo-real, o protocolo de comunicação hipermídia deverá solicitar à camada de adaptação um serviço de transporte não confiável, pois não é desejável a retransmissão dos pacotes de áudio perdidos durante uma transmissão com requisitos de tempo real.*

O protocolo de comunicação hipermídia utiliza as primitivas de serviço de nível inferior especificadas na interface da camada de adaptação. A especificação completa desta interface pode ser encontrada no Apêndice B.

3.2.4- Vocabulário do Protocolo

Nesta Seção especifica-se o vocabulário usado para definir todas as mensagens do protocolo, representado através da seguinte expressão [2]:

$$V(\text{protocolo}) = \{mensagem_1, \dots, mensagem_n\},$$

onde V representa o vocabulário e $\{mensagem_1, \dots, mensagem_n\}$ representam as n mensagens do protocolo.

Quando o vocabulário de um protocolo de comunicação é extenso com muitas mensagens definidas, utiliza-se o conceito de classes de mensagens para agrupar as mensagens relacionadas. Nesse caso, o vocabulário V do protocolo é representado por todas as classes de mensagens, conforme ilustra a expressão abaixo:

$$V(\text{protocolo}) = \{ \text{classe_mensagem}_1, \dots, \text{classe_mensagem}_n \},$$

O protocolo hipermídia proposto define catorze mensagens organizadas em quatro classes¹⁴: **genérico**, **controle_ acesso**, **conteúdo** e **qualidade_serviço**. O vocabulário *V* do protocolo é representado pela seguinte expressão:

$$V(\text{protocolo}) = \{ \text{genérico}, \text{controle_acesso}, \text{conteúdo}, \text{qualidade_serviço} \}$$

A classe de mensagem **genérico** define mensagens para abrir e fechar uma Sessão de Aplicação (mensagens `OpenSession` e `CloseSession`); criar uma Entidade (`CreateEntity`); consultar as propriedades de uma Entidade (`GetEntityProperties`); criar, consultar e modificar as propriedades de uma Entidade (`EntityProperty`); e a mensagem (`Error`) para indicar a ocorrência de qualquer problema durante o uso do protocolo.

A classe **controle_ acesso** define mensagens para controlar o acesso às propriedades de uma determinada Entidade e verificar o tipo de acesso à uma propriedade (`EntityPropertyAccess`); solicitar o tipo de permissão de uma Entidade e definir o tipo de permissão de uma Entidade (`EntityPermission`); solicitar o tipo de permissão de uma propriedade de uma Entidade e definir o tipo de permissão de uma propriedade de uma Entidade (`PropertyPermission`).

A classe **conteúdo** especifica mensagens para iniciar e finalizar a transferência de informações (`Get` e `ThreatPlay`).

A classe **qualidade_serviço** define mensagens para negociar e renegociar QoS (`GetQoS` e `RenegotiateQoS`); e reservar todos os recursos necessários para a criação do *MediaPipe* com a QoS especificada (`Reserve`). Maiores detalhes sobre a especificação de cada uma das mensagens do protocolo podem ser encontrados no Apêndice C.

¹⁴ Todas as classes de mensagens definidas estão diretamente relacionadas às Unidades Funcionais especificadas pelo protocolo hipermídia (ver Seção 3.2.2).

3.2.5- Formato das Mensagens

No protocolo de comunicação hipermídia proposto, optou-se por codificar todas as mensagens definidas em XML (*EXtensible Markup Language*), pois ela oferece as seguintes vantagens: 1- pode utilizar elementos opcionais nas mensagens; 2- é legível para o ser-humano; e 3 - é um padrão industrial utilizado pelos maiores fabricantes de informática. Maiores detalhes da linguagem XML podem ser encontrados em [26, 27, 28, 30].

Todas as mensagens a serem enviadas pelo protocolo serão convertidas em um documento XML válido (*valid*) e bem formado (*well-formedness*) [26, 27] de acordo com a DTD (*Document Type Definition*) - **HypProtocol.dtd**, cuja especificação completa encontra-se no Apêndice D. A Figura 3.2.5 ilustra o formato da mensagem `OpenSession` do protocolo.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE OpenSession SYSTEM "HypProtocol.dtd">
<OpenSession>
  <user>abc</user>
  <password>sampaio</password>
  <client_id>127.0.0.1</client_id>
  <port_client>1234</port_client>
  <server_id>127.0.0.1</server_id>
  <port_server>1234</port_server>
  <profile>generic</profile>
  <session_key></session_key>
</OpenSession>
```

Figura 3.2.5 – Exemplo da mensagem `OpenSession.xml`

3.2.6- Regras do Protocolo

O protocolo de comunicação hiperfídia definiu um conjunto de regras para ser possível gerenciar o funcionamento das Sessões de Aplicação e *MediaPipes* criados. Para especificar essas regras, foram construídas duas máquinas de estados para o protocolo cliente, ilustradas nas Figuras 3.2.6.A e 3.2.6.C, e duas máquinas de estados para o protocolo servidor, ilustradas nas Figuras 3.2.6.B e 3.2.6.D.

Para facilitar o entendimento das máquinas de estados especificadas, fez-se uso da seguinte notação: (sinal número), onde: sinal pode ser (+), indicando o envio de mensagens, ou (-), indicando o recebimento¹⁵ de mensagens; e número representa a ação do protocolo.

A Figura 3.2.6.A ilustra os seis estados que uma Sessão de Aplicação pode assumir (INIT, A.iP, OPEN, CS – CreateSession, CMP - CreateMediaPipe e CLOSE) e as ações realizadas no protocolo hiperfídia cliente. A seguir descrevemos essas ações:

- (+ 1) o estado inicial da Sessão é INIT. Quando a aplicação do usuário envia qualquer primitiva, exceto HYP-OpenSession.req, o estado da Sessão não é alterado e nenhuma mensagem é enviada;
- (+ 2) o estado inicial da Sessão é INIT. Quando a aplicação do usuário envia a primitiva HYP-OpenSession.req, o protocolo envia a mensagem OpenSession.xml e modifica o estado da Sessão para A.iP (Authentication in Process), indicando que está sendo realizado o processo de autenticação do cliente hiperfídia;
- (+ 3) o estado inicial da Sessão é A.iP. Quando a aplicação do usuário envia qualquer primitiva, exceto HYP-OpenSession.req, o estado da Sessão não é alterado e nenhuma mensagem é enviada;
- (+ 4) o estado inicial da Sessão é A.iP. Quando a aplicação do usuário envia a primitiva HYP-OpenSession.req, especificando outro servidor, o protocolo modifica

¹⁵ O evento de *timeout* é representado como se fosse o recebimento de uma mensagem.

o estado da Sessão atual para CS, cria uma nova instância da máquina de estados da Sessão de Aplicação (estado inicial INIT), envia a mensagem OpenSession.xml, modifica o estado da nova Sessão para A.iP e retorna o estado da Sessão atual para A.iP;

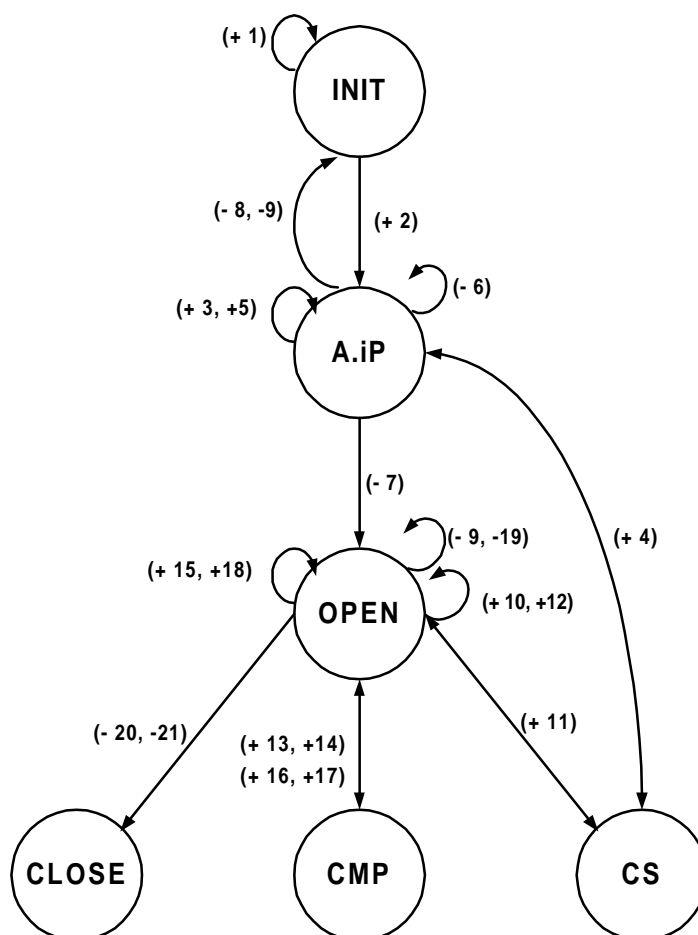


Figura 3.2.6.A – Máquina de Estados da Sessão de Aplicação (Cliente)

- (+ 5) o estado inicial da Sessão é A.iP. Quando a aplicação do usuário envia a primitiva HYP-OpenSession.req, especificando o mesmo servidor da Sessão de Aplicação existente, o protocolo não envia qualquer mensagem e o estado da Sessão não é alterado;
- (- 6) o estado da Sessão é A.iP. Quando o protocolo recebe qualquer primitiva, exceto HYP-OpenSession.cnf e HYP-Error.cnf, o estado da Sessão não é alterado;

- (- 7) o estado da Sessão é A.iP. Quando o protocolo recebe a primitiva HYP-OpenSession.cnf, modifica o estado da Sessão para OPEN;
- (- 8) o estado da Sessão é A.iP. Quando o protocolo recebe a primitiva HYP-Error.cnf, modifica o estado da Sessão para INIT;
- (- 9) o estado da Sessão é A.iP. Quando o protocolo recebe a indicação de *timeout*, modifica o estado da Sessão para INIT;
- (+ 10) o estado da Sessão é OPEN. Quando a aplicação do usuário envia qualquer primitiva, exceto HYP-OpenSession.req, HYP-CreateEntity.req, HYP-GetEntityProperties.req, HYP-Get.req e HYP-Reserve.req, o protocolo envia a mensagem relacionada, sem modificar o estado da Sessão;
- (+ 11) o estado inicial da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-OpenSession.req, especificando outro servidor, o protocolo modifica o estado da Sessão atual para CS, cria uma nova instância da máquina de estados da Sessão de Aplicação, envia a mensagem OpenSession.xml, modifica o estado da nova Sessão para A.iP e retorna o estado da Sessão atual para OPEN;
- (+ 12) o estado inicial da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-OpenSession.req, especificando o mesmo servidor da Sessão de Aplicação existente, o protocolo não envia qualquer mensagem e o estado da Sessão não é alterado;
- (+ 13) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-CreateEntity.req, HYP-GetEntityProperties.req ou HYP-Get.req e não existe *MediaPipe* criado com QoS *best-effort*, o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe* (estado inicial OPEN - maiores detalhes ver Figura 3.2.6.C), envia a mensagem relacionada e retorna ao estado OPEN;

- (+ 14) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-CreateEntity.req, HYP-GetEntityProperties.req ou HYP-Get.req e existe *MediaPipe* criado com QoS *best-effort* sendo utilizado (estado do *MediaPipe* PLAY), o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe*, envia a mensagem relacionada e retorna ao estado OPEN;
- (+ 15) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-CreateEntity.req, HYP-GetEntityProperties.req ou HYP-Get.req e existe *MediaPipe* criado com QoS *best-effort* sem ser utilizado (estado do *MediaPipe* OPEN), o estado da Sessão não é alterado e envia a mensagem relacionada;
- (+ 16) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-Reserve.req e não existe *MediaPipe* criado os requisitos de QoS especificados, o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe*, envia a mensagem relacionada e retorna ao estado OPEN;
- (+ 17) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-Reserve.req e existe *MediaPipe* criado com os requisitos de QoS especificados sendo utilizado, o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe*, envia a mensagem relacionada e retorna ao estado OPEN;
- (+ 18) o estado da Sessão é OPEN. Quando a aplicação do usuário envia a primitiva HYP-Reserve.req e existe *MediaPipe* criado com os requisitos de QoS especificados sem ser utilizado, o estado da Sessão não é alterado e não envia a mensagem relacionada;
- (- 19) o estado da Sessão é OPEN. Quando o protocolo recebe qualquer primitiva, exceto HYP-CloseSession.cnf, não modifica o estado da Sessão;

- (- 20) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-CloseSession.cnf, modifica o estado da Sessão para CLOSE;
- (- 21) o estado da Sessão é OPEN. Quando o protocolo recebe a indicação de *timeout*, modifica o estado da Sessão para CLOSE e destrói todos os *MediaPipes* criados nesta Sessão;

A Figura 3.2.6.B ilustra os seis estados que uma Sessão de Aplicação pode assumir (INIT, A.iP, OPEN, CS – CreateSession, CMP - CreateMediaPipe e CLOSE) e as ações realizadas no protocolo hiperfídia servidor. A seguir descrevemos essas ações:

- (- 1) o estado inicial da Sessão é INIT. Quando o protocolo recebe qualquer primitiva, exceto HYP-OpenSession.ind, o estado da Sessão não é alterado e a mensagem Error.xml é retornada;
- (- 2) o estado inicial da Sessão é INIT. Quando o protocolo recebe a primitiva HYP-OpenSession.ind, modifica o estado da Sessão para A.iP e solicita ao servidor que realize o processo de autenticação: (a) se o cliente hiperfídia for autenticado, modifica o estado da Sessão para OPEN e envia a mensagem OpenSession.xml, indicando que a abertura da Sessão foi realizada com sucesso; (b) caso contrário, modifica o estado da Sessão para INIT e envia Error.xml;
- (- 3) o estado da Sessão é OPEN. Quando o protocolo recebe qualquer primitiva, exceto HYP-OpenSession.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind, HYP-Get.ind, HYP-Reserve.ind e HYP-CloseSession.ind, o protocolo não modifica o estado da Sessão e retorna a mensagem relacionada;
- (- 4) o estado inicial da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-OpenSession.ind, especificando outro cliente, o protocolo modifica o estado da Sessão atual para CS, cria uma nova instância da máquina de estados da Sessão de Aplicação (estado inicial INIT), realiza o processo de autenticação nesta nova Sessão e envia a mensagem OpenSession.xml, caso a Sessão tenha sido criada corretamente,

ou envia a mensagem `Error.xml`, caso contrário, e retorna o estado da Sessão atual para `OPEN`;

- (- 5) o estado da Sessão é `OPEN`. Quando o protocolo recebe a primitiva `HYP-CreateEntity.ind`, `HYP-GetEntityProperties.ind` ou `HYP-Get.ind` e não existe *MediaPipe* criado com *QoS best-effort*, o protocolo modifica o estado da Sessão para `CMP`, cria uma nova instância da máquina de estados do *MediaPipe*, envia a mensagem relacionada e retorna o estado da Sessão para `OPEN`;

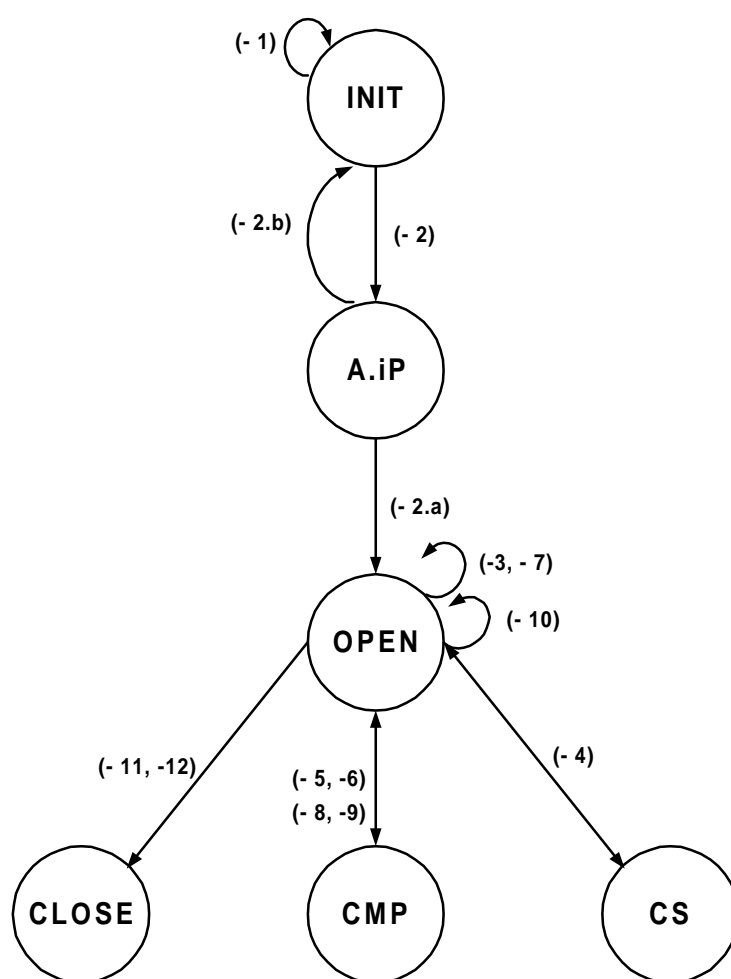


Figura 3.2.6.B – Máquina de Estados da Sessão de Aplicação (Servidor)

- (- 6) o estado da Sessão é `OPEN`. Quando o protocolo recebe a primitiva `HYP-CreateEntity.ind`, `HYP-GetEntityProperties.ind` ou `HYP-Get.ind` e existe *MediaPipe* criado com *QoS best-effort* sendo utilizado, o protocolo modifica o estado da Sessão

para CMP, cria uma nova instância da máquina de estados do *MediaPipe*, envia a mensagem relacionada e retorna o estado da Sessão para OPEN;

- (- 7) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-CreateEntity.ind, HYP-GetEntityProperties.ind ou HYP-Get.ind e existe *MediaPipe* criado com QoS *best-effort* sem ser utilizado, envia a mensagem relacionada e não altera o estado da Sessão;
- (- 8) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-Reserve.ind e não existe *MediaPipe* criado os requisitos de QoS especificados, o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe* e retorna o estado da Sessão para OPEN;
- (- 9) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-Reserve.ind e existe *MediaPipe* criado com os requisitos de QoS especificados sendo utilizado, o protocolo modifica o estado da Sessão para CMP, cria uma nova instância da máquina de estados do *MediaPipe* e retorna o estado da Sessão para OPEN;
- (- 10) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-Reserve.ind e existe *MediaPipe* criado com os requisitos de QoS especificados sem ser utilizado, o estado da Sessão não é alterado;
- (- 11) o estado da Sessão é OPEN. Quando o protocolo recebe a primitiva HYP-CloseSession.ind, modifica o estado da Sessão para CLOSE, destrói todos os *MediaPipes* criados nesta Sessão e envia a mensagem CloseSession.xml;
- (- 12) o estado da Sessão é OPEN. Quando o protocolo recebe a indicação de *timeout*, modifica o estado da Sessão para CLOSE e destrói todos os *MediaPipes* criados nesta Sessão;

A Figura 3.2.6.C ilustra os seis estados que um *MediaPipe* pode assumir (OPEN, CLOSE, PLAY, STOP, PAUSE e RESUME) e as ações realizadas no protocolo hipermídia cliente. A seguir descrevemos estas ações:

- (+ 1) o estado inicial do *MediaPipe* é OPEN. Quando a aplicação do usuário envia qualquer primitiva, o protocolo envia a mensagem relacionada e não modifica o estado do *MediaPipe*;
- (- 2) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe qualquer primitiva, exceto HYP-CloseSession.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf, HYP-Get.cnf e HYP-GetQoS.cnf, não modifica o estado do *MediaPipe*;
- (- 3) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a primitiva HYP-CloseSession.cnf, modifica o estado do *MediaPipe* para CLOSE;
- (- 4) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a primitiva HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf ou HYP-GetQoS.cnf, modifica o estado do *MediaPipe* para PLAY;
- (- 5) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *ttl* do *MediaPipe* (ver Seção 3.3.1), modifica o estado do mesmo para CLOSE;
- (+ 6) o estado inicial do *MediaPipe* é PLAY. Quando a aplicação do usuário envia qualquer primitiva, o protocolo não modifica o estado do *MediaPipe*;
- (- 7) o estado inicial do *MediaPipe* é PLAY. Quando o protocolo recebe qualquer primitiva, exceto HYP-Stop.cnf, HYP-Pause.cnf e HYP-Resume.cnf, não modifica o estado do *MediaPipe*;

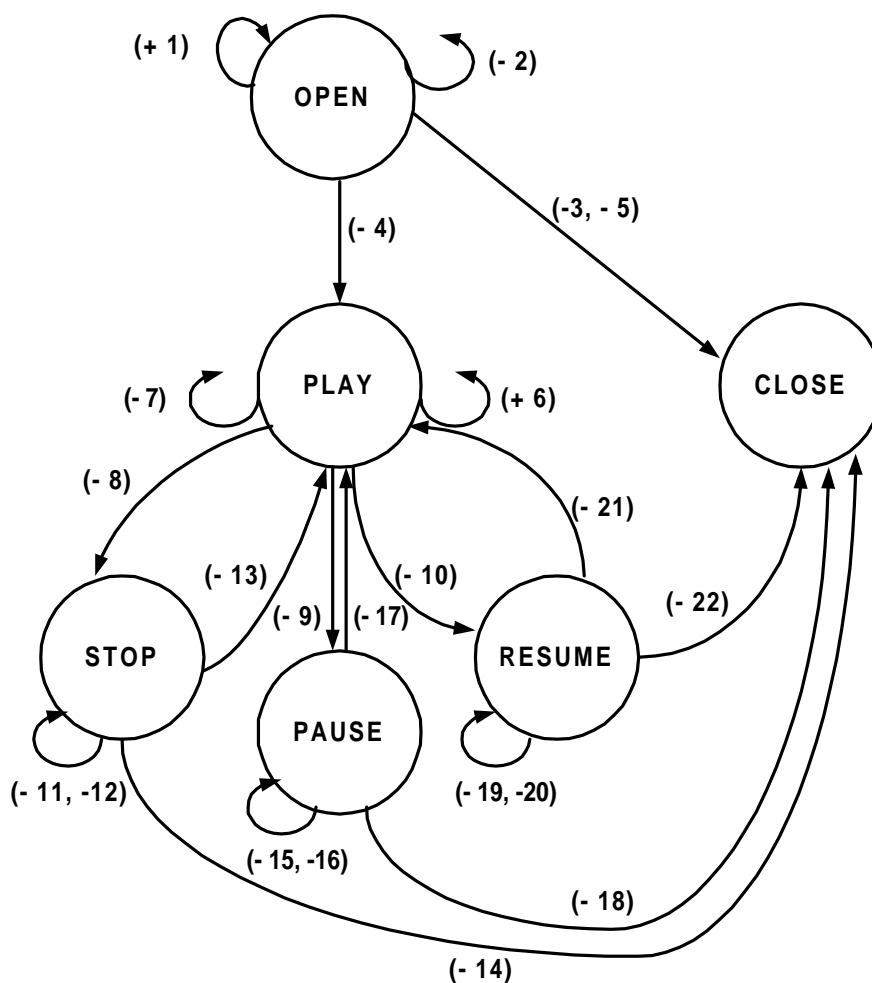


Figura 3.2.6.C – Máquina de Estados do *MediaPipe* (Cliente)

- (- 8) o estado inicial do *MediaPipe* é PLAY. Quando o protocolo recebe a primitiva HYP-Stop.cnf, modifica o estado do *MediaPipe* para STOP;
- (- 9) o estado inicial do *MediaPipe* é PLAY. Quando o protocolo recebe a primitiva HYP-Pause.cnf, modifica o estado do *MediaPipe* para PAUSE;
- (- 10) o estado inicial do *MediaPipe* é PLAY. Quando o protocolo recebe a primitiva HYP-Resume.cnf, modifica o estado do *MediaPipe* para RESUME;
- (- 11) o estado inicial do *MediaPipe* é STOP. Quando a aplicação do usuário envia qualquer primitiva, o protocolo não modifica o estado do *MediaPipe*;

- (- 12) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf e HYP-GetQoS.cnf, não modifica o estado do *MediaPipe*;
- (- 13) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe a primitiva HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf ou HYP-GetQoS.cnf, modifica o estado do *MediaPipe* para PLAY;
- (- 14) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *ttl* do *MediaPipe*, modifica o estado do mesmo para CLOSE;
- (- 15) o estado inicial do *MediaPipe* é PAUSE. Quando a aplicação do usuário envia qualquer primitiva, o protocolo não modifica o estado do *MediaPipe*;
- (- 16) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf e HYP-GetQoS.cnf, não modifica o estado do *MediaPipe*;
- (- 17) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe a primitiva HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf ou HYP-GetQoS.cnf, modifica o estado do *MediaPipe* para PLAY;
- (- 18) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *ttl* do *MediaPipe*, modifica o estado do mesmo para CLOSE;
- (- 19) o estado inicial do *MediaPipe* é RESUME. Quando a aplicação do usuário envia qualquer primitiva, o protocolo não modifica o estado do *MediaPipe*;

- (- 20) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf e HYP-GetQoS.cnf, não modifica o estado do *MediaPipe*;
- (- 21) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe a primitiva HYP-Get.cnf, HYP-CreateEntity.cnf, HYP-GetEntityProperties.cnf ou HYP-GetQoS.cnf, modifica o estado do *MediaPipe* para PLAY;
- (- 22) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *ttl* do *MediaPipe*, modifica o estado do mesmo para CLOSE;

A Figura 3.2.6.D ilustra os seis estados que um *MediaPipe* pode assumir (OPEN, CLOSE, PLAY, STOP, PAUSE e RESUME) e as ações realizadas no protocolo hipermídia servidor. A seguir descrevemos estas ações:

- (- 1) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe qualquer primitiva, exceto HYP-CloseSession.ind, HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind e HYP-GetQoS.ind, não modifica o estado do *MediaPipe* e retorna a mensagem relacionada;
- (- 2) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a primitiva HYP-CloseSession.ind, modifica o estado do *MediaPipe* para CLOSE e retorna a mensagem CloseSession.xml;
- (- 3) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a primitiva HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind ou HYP-GetQoS.ind, modifica o estado do *MediaPipe* para PLAY e retorna a mensagem relacionada;
- (- 4) o estado inicial do *MediaPipe* é OPEN. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *ttl* do *MediaPipe*, modifica o estado do mesmo para CLOSE;

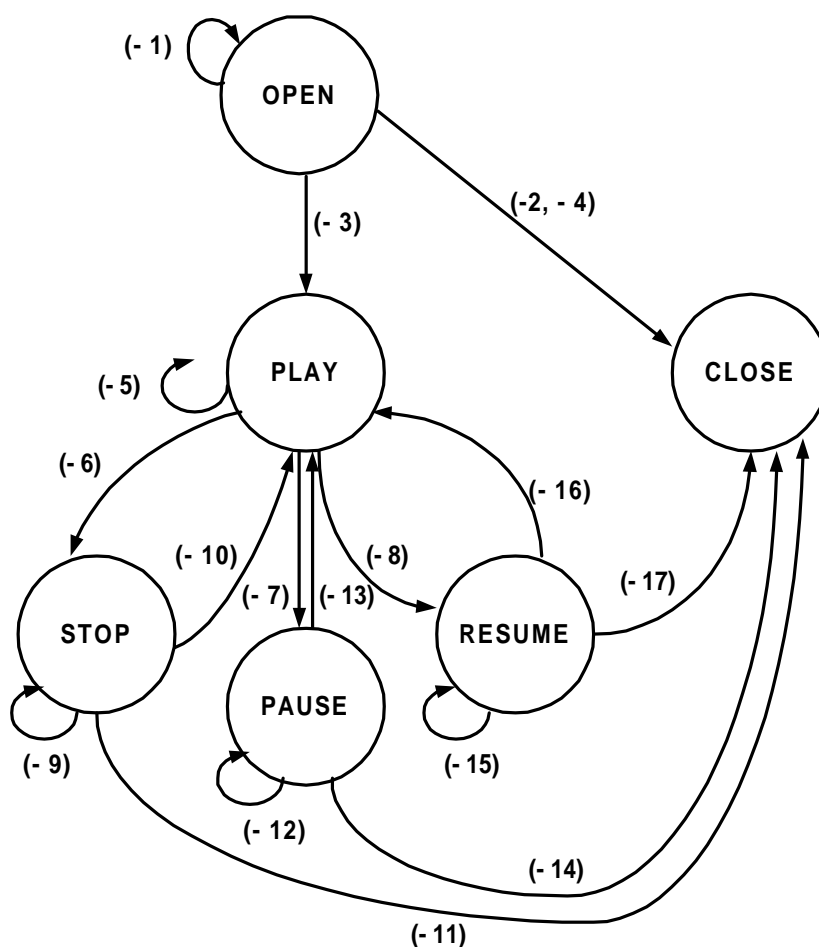


Figura 3.2.6.D – Máquina de Estados do MediaPipe (Servidor)

- (- 5) o estado inicial do *MediaPipe* é **PLAY**. Quando o protocolo recebe qualquer primitiva, exceto HYP-Stop.ind, HYP-Pause.ind e HYP-Resume.ind, não modifica o estado do *MediaPipe* e retorna a mensagem relacionada;
- (- 6) o estado inicial do *MediaPipe* é **PLAY**. Quando o protocolo recebe a primitiva HYP-Stop.ind, modifica o estado do *MediaPipe* para **STOP** e retorna a mensagem Stop.xml;
- (- 7) o estado inicial do *MediaPipe* é **PLAY**. Quando o protocolo recebe a primitiva HYP-Pause.ind, modifica o estado do *MediaPipe* para **PAUSE** e retorna a mensagem Pause.xml;

- (- 8) o estado inicial do *MediaPipe* é PLAY. Quando o protocolo recebe a primitiva HYP-Resume.ind, modifica o estado do *MediaPipe* para RESUME e retorna a mensagem Resume.xml;
- (- 9) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind e HYP-GetQoS.ind, não modifica o estado do *MediaPipe* e retorna a mensagem relacionada;
- (- 10) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe a primitiva HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind ou HYP-GetQoS.ind, modifica o estado do *MediaPipe* para PLAY e retorna a mensagem relacionada;
- (- 11) o estado inicial do *MediaPipe* é STOP. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *tll* do *MediaPipe*, modifica o estado do mesmo para CLOSE;
- (- 12) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind e HYP-GetQoS.ind, não modifica o estado do *MediaPipe* e retorna a mensagem relacionada;
- (- 13) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe a primitiva HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind ou HYP-GetQoS.ind, modifica o estado do *MediaPipe* para PLAY e retorna a mensagem relacionada;
- (- 14) o estado inicial do *MediaPipe* é PAUSE. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *tll* do *MediaPipe*, modifica o estado do mesmo para CLOSE;

- (- 15) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe qualquer primitiva, exceto HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind e HYP-GetQoS.ind, não modifica o estado do *MediaPipe* e retorna a mensagem relacionada;
- (- 16) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe a primitiva HYP-Get.ind, HYP-CreateEntity.ind, HYP-GetEntityProperties.ind ou HYP-GetQoS.ind, modifica o estado do *MediaPipe* para PLAY e retorna a mensagem relacionada;
- (- 17) o estado inicial do *MediaPipe* é RESUME. Quando o protocolo recebe a indicação de *timeout*, isto é, expirou *tll* do *MediaPipe*, modifica o estado do mesmo para CLOSE.

3.3- QoS no Protocolo Hiperfídia

As aplicações solicitam serviços de comunicação com QoS a partir de categorias ou classes de serviços padronizadas. O que vai diferenciar uma categoria de serviço de outra, será o grau de comprometimento do provedor com a QoS solicitada, ou seja, o nível de garantia do serviço fornecido [31].

Dependendo do tipo de informação contida em um hiper-documento, as aplicações hiperfídia podem ser classificadas como **elásticas**, **tempo-real tolerantes** ou **tempo-real intolerantes**. As **aplicações elásticas**, como por exemplo a transferência de *um e-mail*, não dependem tanto dos instantes de entrega dos pacotes, admitindo variações no desempenho da rede. As **aplicações em tempo-real tolerantes**, como por exemplo a transmissão de uma vídeo-conferência, são flexíveis no que diz respeito à QoS solicitada e efetivamente fornecida pela rede. As **aplicações em tempo-real intolerantes**, como por exemplo o recebimento de um vídeo contendo informações médicas para uso na telemedicina, exigem um serviço que forneça garantias em relação aos requisitos de QoS solicitados.

Para atender às necessidades dessas classes de aplicações, o protocolo hipermídia com QoS proposto oferece três categorias de serviço¹⁶ [16, 18, 31]: *best-effort*, *controlled-load* e *guaranteed*.

O tipo de serviço *best-effort* (melhor-esforço) é caracterizado pela ausência de garantias em relação à QoS efetivamente oferecida pela infra-estrutura de comunicação e é indicado às **aplicações elásticas**.

O tipo de serviço *controlled-load* (carga controlada) oferece um retardo baixo, próximo ao oferecido por um serviço *best-effort* em uma infra-estrutura de rede sem congestionamento e é indicado às **aplicações em tempo-real**.

O tipo de serviço *guaranteed* (garantido) garante um retardo máximo fim-a-fim, sem perda de pacotes. Os recursos necessários ao fornecimento deste tipo de serviço devem ser reservados exclusivamente, o que torna este serviço mais caro em comparação com os demais. Este serviço é indicado às **aplicações em tempo-real intolerantes**.

Para viabilizar QoS em um protocolo de comunicação, deve-se:

- definir uma API com QoS, ou seja, especificar primitivas para negociar, monitorar e renegociar os parâmetros de QoS;
- definir mecanismos de tratamento de QoS responsáveis pelo mapeamento, controle de admissão, monitoração, sintonização e reserva de recursos.

¹⁶ Utilizando a abordagem adotada pelo grupo de pesquisas IntServ do IETF.

3.3.1- API com QoS

A API com QoS é oferecida pela unidade funcional Qualidade de Serviço (ver Seção 3.2.2). As primitivas dessa API estão exemplificadas na Figura 3.3.1.A.

```

HYP_GetQoS (String session_key, String entity_id)

HYP_Reserve (String session_key, String entity_id, ServiceCategory
serviceQoS)

HYP_RenegotiateQoS (String session_key, String mediaPipe_id,
ServiceCategory serviceQoS)

HYP_AlertQoS (String session_key, String mediaPipe_id,
ServiceCategory serviceQoS)

```

Figura 3.3.1.A – Primitivas de QoS

A primitiva **HYP_GetQoS** é utilizada pelo cliente hipermídia para solicitar um documento com garantias de qualidade. A primitiva **HYP_Reserve** é utilizada para solicitar a criação de um *MediaPipe* com a QoS especificada. O protocolo hipermídia recebe a solicitação do serviço e verifica se existe algum *MediaPipe* criado com a QoS desejada. Se houver, verifica se não existe outro cliente hipermídia utilizando o *MediaPipe*. Não havendo outro cliente utilizando o *MediaPipe*, este será utilizado para o transporte das informações solicitadas. Caso não exista *MediaPipe* disponível com a QoS solicitada, o protocolo verifica a existência de recursos para a criação de um novo *MediaPipe* com a QoS especificada.

Durante o fornecimento do serviço, mudanças de QoS podem ser necessárias. Neste caso, utiliza-se a primitiva **HYP_RenegotiateQoS** para renegociar os parâmetros de QoS inicialmente especificados em um *MediaPipe*. Mecanismos de monitoração¹⁷ verificam se os parâmetros de qualidade especificados estão sendo efetivamente oferecidos pelos provedores de serviços. Em caso de violação da QoS, as aplicações do usuário devem receber alertas através da primitiva **HYP_AlertQoS**.

¹⁷ Na implementação do protótipo [53], utilizou-se o protocolo RTCP [3, 4, 7] para monitorar a QoS dos *MediaPipes* criados.

A Figura 3.3.1.B ilustra um AOS que possui uma Sessão de Aplicação com três *MediaPipes*, cada um com uma qualidade de serviço associada (QoS1, QoS2 e QoS3). Sabe-se que não é possível especificar um único requisito de QoS que satisfaça às exigências de qualidade para todas as mídias contidas em um hiper-documento. Por exemplo, as **mídias não contínuas no tempo**, como a **mídia texto**, toleram retardo na transferência de documentos mas não perda de dados; entretanto, as mídias contínuas no tempo, como as **mídias áudio e vídeo**, toleram perda de dados mas não retardo na transferência. Assim, não é desejável a transmissão da mídia texto, áudio e vídeo utilizando um único *MediaPipe*. Por isso, o protocolo de comunicação hipermídia proposto estabeleceu que todo *MediaPipe* é específico para um determinado tipo de mídia.

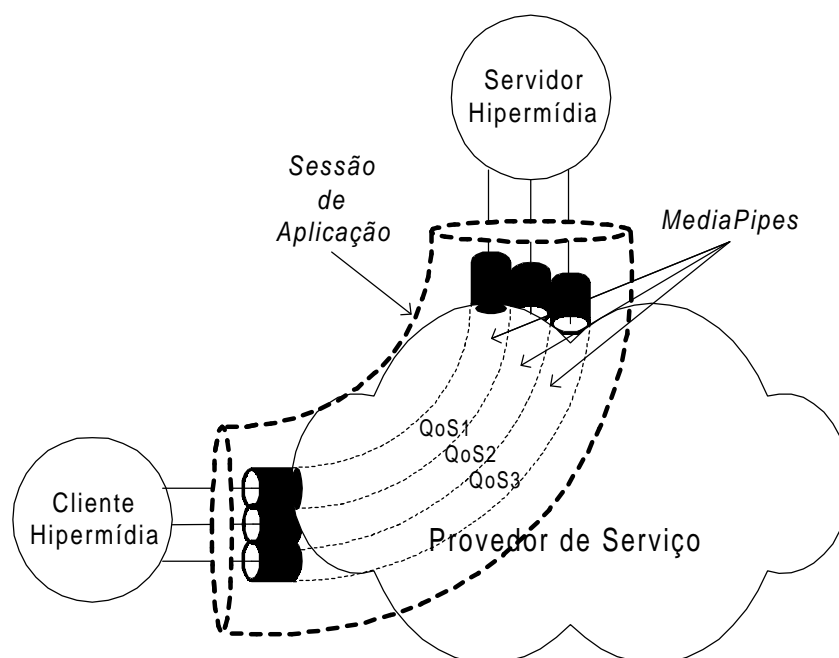


Figura 3.3.1.B – AOS com Sessão de Aplicação e *MediaPipes*

Quando houver a necessidade de transmitir várias informações do mesmo tipo, por exemplo a transmissão de vários documentos áudio, pode-se utilizar um único *MediaPipe*, caso a QoS oferecida seja satisfatória para a transmissão de todos os áudios, ou criar novos *MediaPipes*, cada um específico para determinada transmissão de áudio.

A seguir descreve-se algumas convenções adotadas pelo protocolo sobre *MediaPipes*:

- 1- todo *MediaPipe* possui um tempo de vida (*ttl* – *time-to-live*). O valor de *ttl* é atualizado quando há troca de informações no *MediaPipe*. Quando expirar este tempo, isto é, não houver transmissão de informações durante um intervalo de tempo maior ou igual a *ttl*, todos os recursos reservados na comunicação fim-a-fim serão liberados;
- 2- todo *MediaPipe* possui um requisito de QoS associado. Caso a aplicação do usuário não especifique os parâmetros de qualidade desejados, a QoS *best-effort* é utilizada;
- 3- o *MediaPipe* é classificado em dois tipos: *data* (para transporte de *stream* de informações) e *control* (para transporte de mensagens XML do protocolo hipermídia).
- 4- a solicitação para criar *MediaPipes* é sempre realizada pela aplicação hipermídia cliente, seguindo a abordagem adotada pelas arquiteturas QoS-A, XRM-Comet e Tenet [1,17].

3.3.2. Protocolo com QoS

Para permitir o tratamento de QoS no protocolo hipermídia, utilizou-se o *framework para Provisão de QoS em Ambientes Genéricos de Processamento e Comunicação*, desenvolvido no laboratório Telemídia da Puc-Rio [14]. Este *framework* é composto pelos seguintes elementos:

- A. *Um Framework para Parametrização de Serviços*;
- B. *Frameworks para Compartilhamento de Recursos*, que incluem:
 - B.1. *Um Framework para Escalonamento de Recursos* e
 - B.2. *Um Framework para Alocação de Recursos*;
- C. *Frameworks para Orquestração de Recursos*, que incluem:

C.1. *Um Framework para Negociação de QoS e*

C.2. *Um Framework para Sintonização de QoS.*

Para oferecer tratamento de QoS no protocolo de comunicação hipermídia, foi necessário instanciar o *framework para Provisão de QoS*, isto é, implementar os *hot spots* do *framework* [52].

A Figura 3.3.2 ilustra todas as fases de desenvolvimento do *framework* de QoS. Cada elemento que compõe o *framework para Provisão de QoS* foi instanciado. As próximas seções descrevem os módulos de *software* que foram gerados.

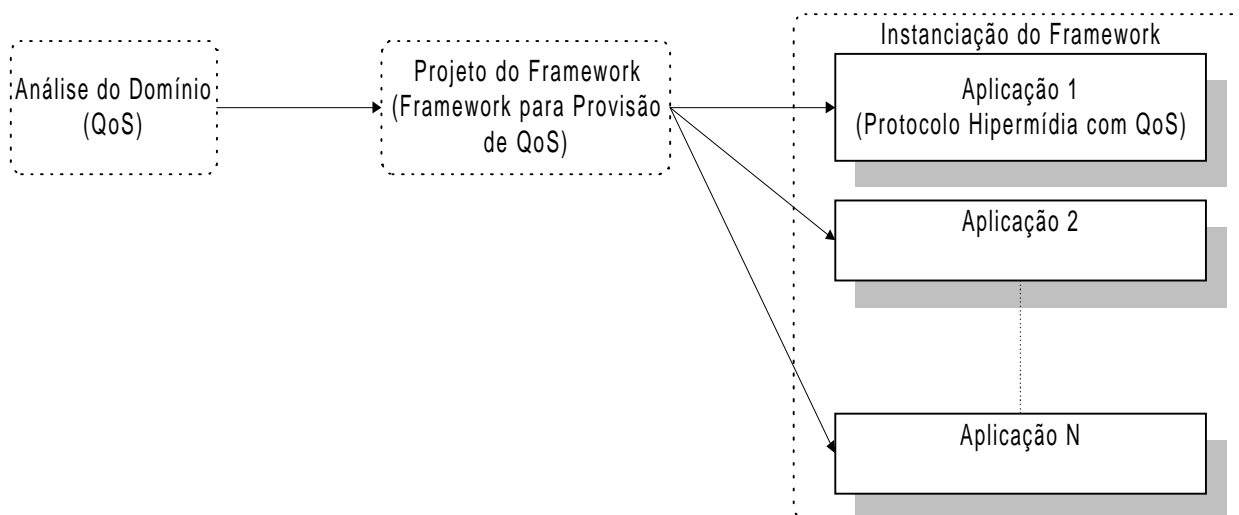


Figura 3.3.2 – Processo de Desenvolvimento de *Frameworks*

3.3.2.1. Instânciação do *Framework para Parametrização de Serviços*

O *Framework para Parametrização de Serviços* oferece um modo de se estruturar dados adequadamente para a representação de parâmetros de QoS e do nível de serviço desejados por uma aplicação [14].

A Figura 3.3.2.1 apresenta a instância do *Framework* para Parametrização de Serviços, cuja instanciação gerou o *software* conhecido como *ServiceCategory*¹⁸. O nível de serviço desejado é representado pela super-classe *ServiceCategory* (*hot-spot* do *framework*) e os parâmetros de QoS são especificados pela super-classe *Parameters* (*hot-spot* do *framework*). Estas classe estão relacionadas através do atributo *parameterList* que define o conjunto de parâmetros de QoS associados a uma determinada categoria de serviço.

A super-classe *ServiceCategory* especifica as categorias de serviço desejadas pelas aplicações hipermídia que irão utilizar o protocolo com QoS. Essas categorias de serviço estão definidas em duas sub-classes¹⁹: *Guaranteed* e *Controlled-load*. Como os *MediaPipes* oferecidos pelo protocolo são específicos para um determinado tipo de mídia, foram criadas sub-classes de *Guaranteed* e *Controlled-load*, específicas para um determinado tipo de mídia. *Por exemplo, a sub-classe VideoGuaranteed será utilizada quando a aplicação do usuário solicitar a transferência de um vídeo com todas as garantias de qualidade.*

Todas as sub-classes de *ServiceCategory* possuem métodos que definem quais parâmetros de QoS devem ser utilizados.

Continuando o exemplo anterior, a sub-classe VideoGuaranteed possuirá cinco parâmetros de QoS: max_delay, frame_size, frame_rate, resolution e videoCodification.

A super-classe *Parameters* especifica os parâmetros de qualidade que serão utilizados pelo protocolo de comunicação hipermídia. Esses parâmetros de QoS são sub-classes de *Parameters*.

É importante ressaltar que o protocolo hipermídia irá utilizar parâmetros de QoS nível de aplicação e nível de transporte, devido à necessidade de realizar mapeamento dos

¹⁸ Todas as primitivas da API com QoS (Seção 3.4.1) possuem um argumento do tipo *ServiceCategory* para especificar os parâmetros de qualidade e o nível de serviço desejados.

¹⁹ A categoria de serviço *best-effort* é o nível de serviço padrão oferecido pelo protocolo hipermídia, não sendo necessário a sua modelagem.

requisitos de QoS de um nível para o outro. Dessa forma, existem sub-classes de *Parameters* que representam parâmetros de QoS nível de aplicação e nível de transporte. Os parâmetros de QoS nível de transporte estão agrupados em um retângulo pontilhado na Figura 3.3.2.1.

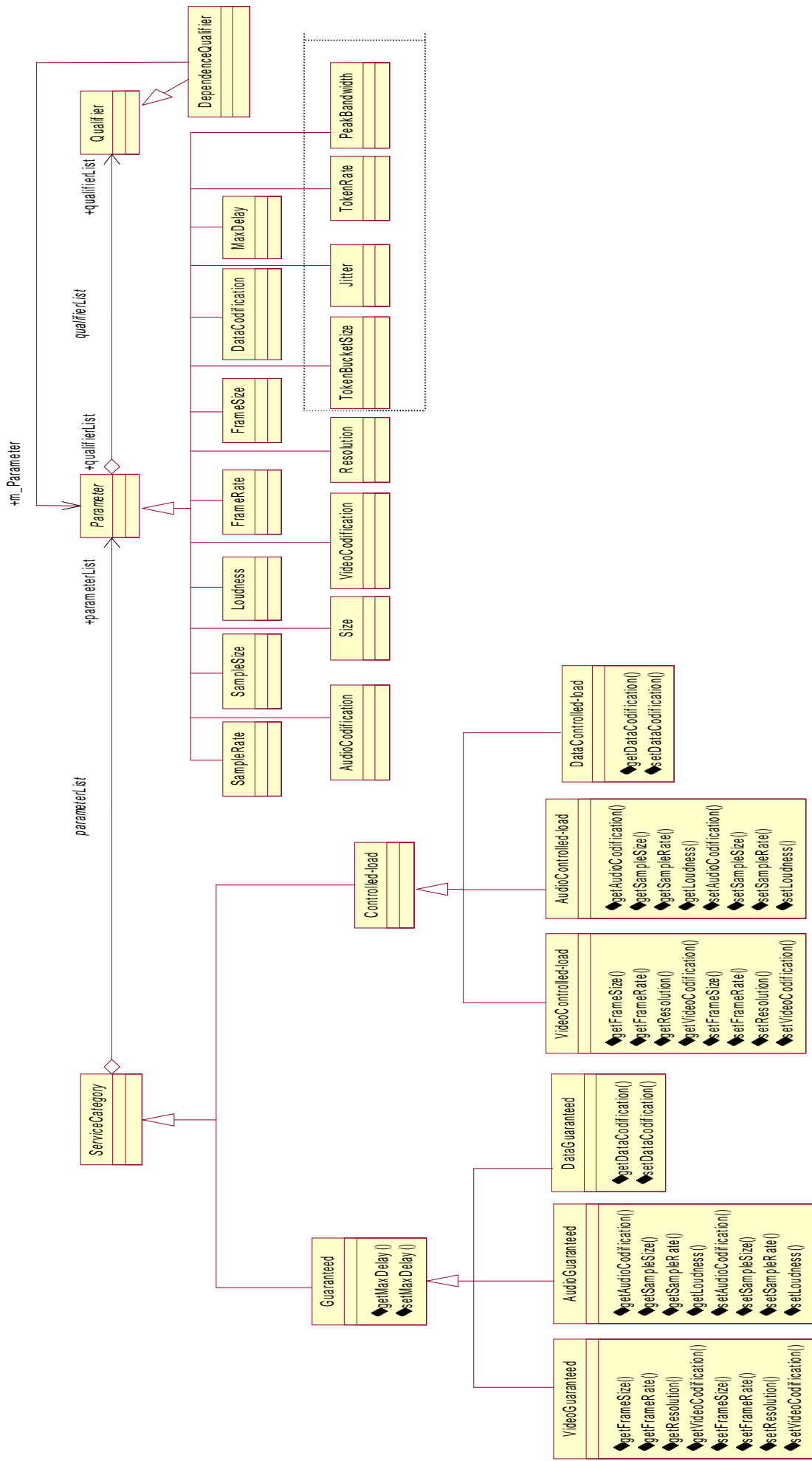


Figura 3.3.2.1 – Instância do Framework para Parametrização de Serviços (Módulo de software ServiceCategory)

3.3.2.2. Instânciação dos *Frameworks para Compartilhamento e Orquestração de Recursos*

A Figura 3.3.2.2 apresenta as instâncias dos *Frameworks para Compartilhamento e Orquestração de Recursos*, gerando o módulo de *software* conhecido como *Quality*. Para facilitar o entendimento desta figura, todas as instanciações realizadas foram delimitadas por áreas geométricas que identificam o *framework* instanciado.

3.3.2.2.1. Instânciação dos *Frameworks para Compartilhamento de Recursos*

O *Framework para Compartilhamento de Recursos* inclui um *Framework para Escalonamento de Recursos* e outro *Framework para Alocação de Recursos*. O primeiro modela os mecanismos de escalonamento com base no conceito de árvores de recursos virtuais e o segundo modela os mecanismos de alocação com base no conceito de criação de recursos virtuais.

O conceito de *árvores de recursos virtuais* representa a parcela de utilização de um recurso real pelos usuários de um sistema de comunicação. Este conceito é extensível a todo ambiente de comunicação, envolvendo as entidades de comunicação (computadores, por exemplo) e os provedores de serviço (roteadores, enlaces de comunicação, etc.). No caso do protocolo de aplicação hipermídia com QoS, as *árvores de recursos virtuais* envolvem o escalonamento de recursos a nível de sistema operacional, como por exemplo, o escalonamento de processadores em sistemas operacionais da família UNIX, e a nível de rede, o escalonamento de recursos envolvidos na comunicação, como por exemplo, largura de banda dos roteadores utilizados.

Na especificação do protocolo hipermídia com QoS, a utilização do conceito de árvores de recursos virtuais está limitado à utilização de escalonadores de recursos de rede já existentes. A nível de sistema operacional, o protótipo implementado do protocolo não realiza qualquer escalonamento de recursos, pois aumentaria a complexidade deste trabalho, ficando como assunto para futuros trabalhos.

O *Framework para Alocação de Recursos* é responsável pela criação de recursos virtuais. Criar recursos virtuais é responsabilidade da classe *InfrastructureQoSNegotiator*. Esta classe implementa os métodos definidos na interface *PrimitivesResourceReservation*, para realizar a alocação de recursos virtuais.

A classe *InfrastructureQoSNegotiator* é responsável por realizar a reserva dos recursos necessários para o estabelecimento de *MediaPipes*. Esta classe possui o atributo *HypInf* que define o negociador de QoS utilizado pelo protocolo hipermídia (representado pela classe *ProtocolQoSNegotiator*).

3.3.2.2.2. Instânciação dos Frameworks para Orquestração de Recursos

O *Framework para Orquestração de Recursos* inclui um *Framework para Negociação da QoS* e outro *Framework para Sintonização da QoS*. O primeiro modela os mecanismos de negociação e mapeamento, assim como os de controle de admissão e o segundo modela os mecanismos de de monitoração e sintonização.

As classes *QoSAdmissionController* e *QoSMapper* são responsáveis por realizar o controle de admissão de novos *MediaPipes* e mapear os parâmetros de qualidade a nível de aplicação para os de transporte, respectivamente. A classe *QoSMonitor* é utilizada para monitorar a QoS estabelecida por um *MediaPipe*, verificando se está de acordo com o nível de serviço solicitado pelo usuário. Caso não esteja, mecanismos de sintonização²⁰ podem ser acionados (classe *QoSTuner*). Todas estas classes estão relacionadas com o negociador de QoS utilizado pelo protocolo hipermídia (*ProtocolQoSNegotiator*).

A classe principal do sistema de *software* gerado, *Quality*, é *ProtocolQoSNegotiator*. Esta classe pode ser analisada como sendo uma extensão do protocolo de comunicação hipermídia, sendo responsável por negociar o oferecimento de serviços com a qualidade especificada pelas aplicações.

A classe *QoSMapper* utiliza as regras de mapeamento definidas na interface *MappingStrategy*. As sub-classes definidas em *MappingStrategy* são utilizadas para definir regras de mapeamento específicas para cada tipo de mídia que pode ser utilizada. Neste trabalho, foram definidas duas regras de mapeamento: uma para áudio PCM e a outra para vídeo H.261. Maiores detalhes sobre a primeira regra serão mostrados no Capítulo 4, Seção 4.3.2.

A classe *QoSAdmissionController* utiliza as regras de controle de admissão definidas na interface *AdmissionStrategy*. As sub-classes definidas são específicas para cada mecanismo de controle de admissão.

²⁰ Mecanismos de sintonização são responsáveis pela manutenção da orquestração dos recursos com a QoS negociada, sem que haja necessidade de interrupção, isto é, renegociação do fornecimento do serviço.

A classe *QoSMonitor* utiliza as regras de monitoração definidas na interface *MonitoringStrategy*. A sub-classe *MonitoringStrategyRTCP* é responsável por monitorar os pacotes de sinalização RTCP.

4

CENÁRIOS DE USO DO PROTOCOLO DE COMUNICAÇÃO HIPERMÍDIA

Este Capítulo apresenta cenários de uso do protocolo de comunicação hipermídia, e ilustra como seu protótipo[53] realiza cada uma das quatro etapas do processo de solicitação de serviço com QoS.

4.1- Funcionamento do Protocolo de Comunicação Hipermídia

Um modelo genérico de operação de um protocolo, relacionado à provisão de QoS, pode ser dividido em quatro fases [12, 14]:

1. **iniciação do provedor de serviços**, para definir a infra-estrutura de comunicação que será utilizada;
2. **requisição de serviços**, para solicitar serviços com QoS ao provedor de serviços;
3. **estabelecimento de contratos de serviço**, para negociar e renegociar os parâmetros de QoS desejados durante uma transmissão;
4. **manutenção (controle e gerência) dos contratos de serviço**, para monitorar e sintonizar a QoS oferecida pelo provedor de serviços.

As próximas seções descrevem os cenários de uso do protocolo e ilustram como cada uma dessas fases é realizada.

4.2. Cenário de Iniciação do Provedor de Serviços

Para tornar um protocolo operacional como um provedor de serviços, é necessário, primeiramente, que seja definida a infra-estrutura que dará suporte aos serviços oferecidos e que determinará a forma de descrição do estado interno do mesmo. A definição desse estado interno inclui informações sobre os recursos disponíveis no ambiente [12].

Para a realização desse estudo, usou-se duas estações Pentium II/333MHz existentes no Laboratório Telemídia da PUC-Rio[56], interconectadas por uma rede *Ethernet 10 Base-T*. A Figura 4.2.A ilustra as estações e a base de dados (BD) utilizados na rede. A BD possui documentos do tipo áudio (A), vídeo (V), texto (T) e imagem (I).

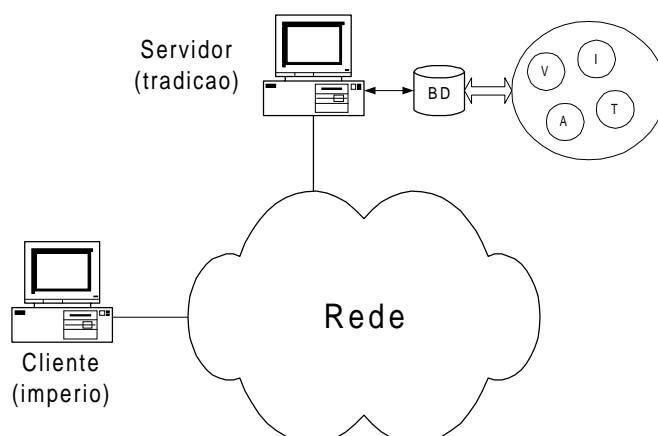


Figura 4.2.A – Estações Cliente e Servidor utilizadas no Laboratório Telemídia da Puc-Rio [56]

Para tornar o protocolo de comunicação hipermídia operacional, todos os elementos que fazem parte do Modelo de Referência especificado no Capítulo 3, tais como: 1- aplicações cliente / servidor; 2- arquivos de configuração²¹ da camada de adaptação; e 3- protocolos de transporte e reserva de recursos, devem ser iniciados.

Para ilustrar os cenários de uso do protocolo de comunicação hipermídia, foi desenvolvido um protótipo de comunicação²² que implementa: 1- Modelo de Referência apresentado na Seção 3.1; 2- Unidades Funcionais definidas na Seção 3.2.2; e 3- os módulos de *software* (*ServiceCategory* e *Quality*) desenvolvidos na Seção 3.3.

*Por exemplo, as aplicações cliente e o servidor desenvolvidos no protótipo foram iniciados na estação **imperio.telemidia.puc-rio.br** (endereço IP:porta → 139.82.95.18:1054) e na estação **tradicao.telemidia.puc-rio.br** (endereço IP:porta →*

²¹ Indica qual o protocolo de transporte disponível que mais se adequa ao tipo de informação a ser transportada.

139.82.95.15:1234), respectivamente. O protocolo de comunicação hipermédia também foi iniciado nestas estações. Dois arquivos de configuração, transport.ini e mediaPipeTransport.ini, foram utilizados. Esses arquivos indicam que a camada de adaptação pode utilizar os protocolos de transporte (RTP- para transportar mídias contínuas no tempo; TCP- para transportar mídias não contínuas no tempo; e UDP- para transportar mensagens do protocolo) e de reserva de recursos (RSVP- para reservar todos os recursos necessários para o transporte de dados com garantias de qualidade) nestas duas estações. A Figura 4.2.B ilustra as aplicações cliente (*imperio*) e servidor (*tradicao*) iniciados no protótipo [53].

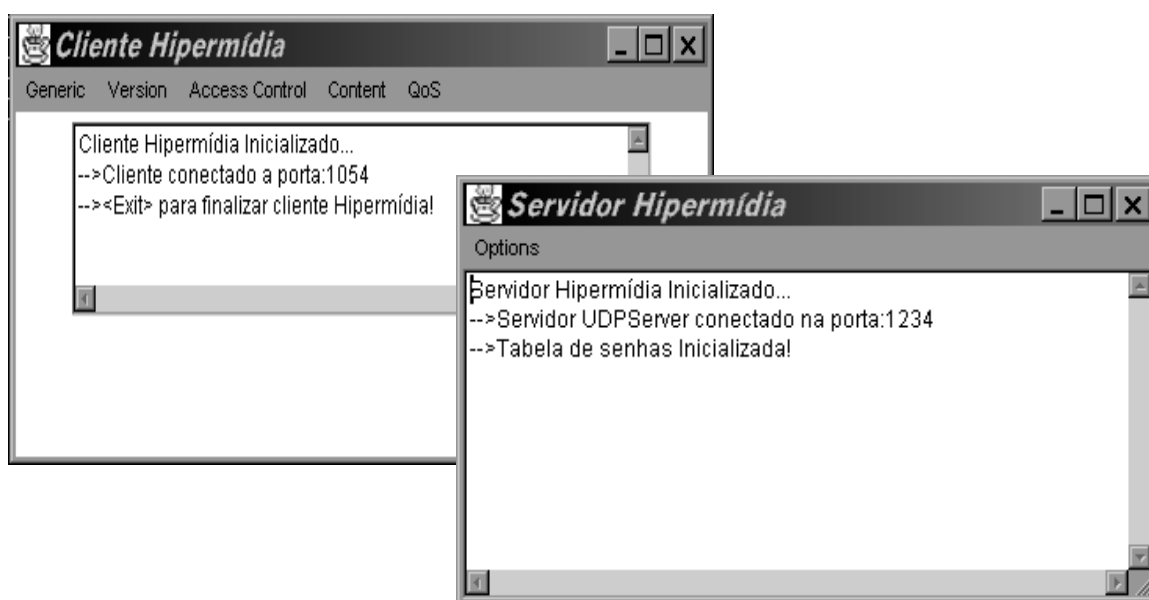


Figura 4.2.B – Tela do Cliente e do Servidor iniciados no protótipo de comunicação

Após a iniciação das aplicações cliente e servidor e do protocolo de comunicação hipermédia, usuários podem requisitar, a qualquer momento, serviços de comunicação ao protocolo.

Nas próximas seções, descrevem-se os demais cenários de uso do protocolo quando a aplicação cliente solicita o transporte de um documento, armazenado na BD do servidor, com garantias de qualidade.

²² Maiores detalhes sobre o desenvolvimento do protótipo e o código fonte das classes e interfaces desenvolvidas podem ser encontrados em [53].

4.3. Cenário de Requisição de Serviços com QoS

Neste Cenário, descrevem-se todos os passos realizados no protocolo de comunicação hipermídia para a *criação de uma Sessão de Aplicação* e a *solicitação do envio de um documento com garantias de qualidade*. As próximas duas sub-seções descrevem, em detalhes, todos os passos realizados.

4.3.1. Criando a Sessão de Aplicação

Para se criar uma Sessão de Aplicação, vários passos devem ser realizados no Cliente e no Servidor. A seguir, descreve-se quais são as etapas necessárias para a criação de uma Sessão de Aplicação entre o cliente (*imperio*) e o servidor (*tradicao*) iniciados anteriormente, utilizando a unidade funcional QoS²³.

Passos Realizados no Cliente:

Passo1: Solicitando a Criação de uma Sessão de Aplicação

A aplicação hipermídia cliente solicita serviços ao protocolo de comunicação, utilizando as primitivas definidas na interface 1 do Modelo de Referência, introduzidas na Seção 3.1 e detalhadas no Apêndice A.

Exemplo01: a aplicação cliente (localizado em imperio.telemidia.puc-rio.br) solicita a abertura de uma Sessão de Aplicação²⁴ com o servidor (localizado em tradicao.telemidia.puc-rio.br). A primitiva `HYP_OpenSession.request(abc, sampa, tradicao, QoS)` é utilizada pela aplicação cliente para solicitar ao protocolo hipermídia a criação de uma Sessão de Aplicação com a unidade funcional QoS. A Figura 4.3.1.A ilustra a interface gráfica utilizada pelo cliente para solicitar a abertura de uma sessão.

²³ Será solicitado a abertura de uma Sessão de Aplicação com duas unidades funcionais: genérico (comum a qualquer sessão) e QoS (especificada pela aplicação do usuário).

²⁴ Conforme foi descrito na Seção 3.2.1, para solicitar qualquer serviço de comunicação ao protocolo hipermídia, o cliente deve criar uma Sessão de Aplicação com o servidor. Na implementação do protótipo, restringiu-se a criação de Sessão de Aplicação somente para comunicação *unicast*.

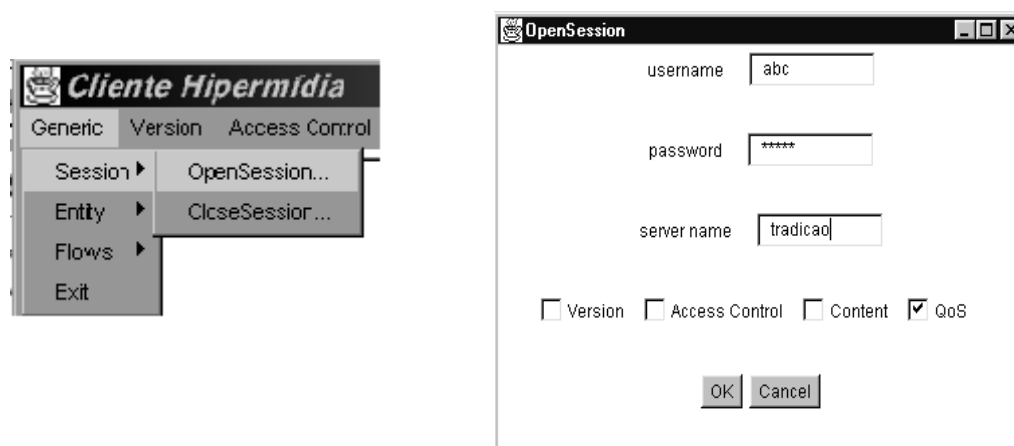


Figura 4.3.1.A – Janela do Protótipo para solicitar a abertura de uma Sessão de Aplicação

Passo2: Verificando a Existência da Sessão

O protocolo hiperMídia cliente, ao receber a solicitação para criar uma Sessão de Aplicação com a unidade funcional QoS, irá verificar se já existe sessão criada com o servidor identificado. Se houver, retorna uma indicação de erro²⁵ ao cliente (primitiva *HYP_Error()*). Caso não exista Sessão de Aplicação criada, os parâmetros da primitiva *HYP_OpenSession.request(user_name, password, server_name, profile)* são codificados, sendo gerado um documento XML válido e bem-formado.

*Continuando o Exemplo01, não existindo uma Sessão criada, o protótipo do protocolo usa a primitiva *HYP_OpenSession.request(abc, sampa, tradicao, QoS)* para gerar a mensagem *OpenSession.XML* (ver Figura 4.3.1.B).*

Passo3: Enviando mensagem XML ao Servidor

Fazendo uso de qualquer protocolo de transporte²⁶, o protocolo hiperMídia cliente envia a mensagem XML.

²⁵ Na implementação do protótipo, restringiu-se que uma aplicação cliente só pode possuir uma única Sessão de Aplicação com o mesmo servidor.

²⁶ Na implementação do protótipo, a camada de adaptação utiliza um arquivo de configuração *transport.ini* que indica qual o protocolo de transporte a ser utilizado. Neste exemplo, utilizou-se o protocolo de transporte UDP.

Seguindo o Exemplo01, a mensagem *OpenSession.xml* é enviada ao servidor.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE OpenSession SYSTEM "HypProtocol.dtd">
<OpenSession>
  <user>abc</user>
  <password>sampa</password>
  <client_id>imperio</client_id>
  <port_client>1054</port_client>
  <server_id>tradicao</server_id>
  <port_server>1234</port_server>
  <profile>QoS</profile>
  <session_key></session_key>
</OpenSession>
```

Figura 4.3.1.B – Mensagem *OpenSession.xml* gerada pelo protocolo hipermídia²⁷

As ações realizadas no Cliente para solicitar a criação de uma Sessão de Aplicação com a unidade funcional QoS estão ilustradas no Diagrama de Seqüência da Figura 4.3.1.C.

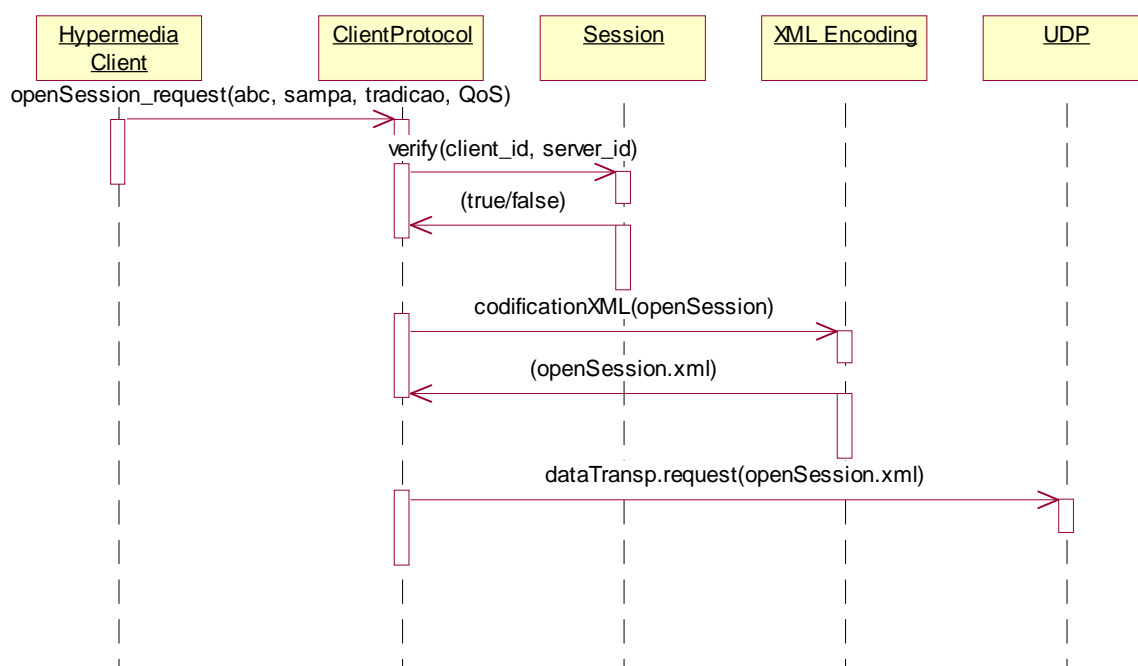


Figura 4.3.1.C – Diagrama de Seqüência realizado no Cliente para solicitar a criação de uma Sessão de Aplicação com a unidade funcional QoS

²⁷ Fez uso do *parser XML4J 1.1.16 (IBM's XML Parser for Java)* [30], processador XML escrito em Java, utilizado para gerar documentos XML válidos e bem-formados [26, 27]

Passos Realizados no Servidor:**Passo4: Recebendo mensagem XML do Cliente**

Utilizando qualquer protocolo de transporte, o servidor recebe a mensagem XML.

Dando sequência ao Exemplo01, o protocolo de transporte recebe a mensagem OpenSession.xml enviada pelo cliente.

Passo5: Decodificando mensagem XML recebida

O protocolo hipermídia servidor recebe do protocolo de transporte uma primitiva de serviço (*dataTransp.indication(message)*), indicando o recebimento de uma mensagem XML. Essa mensagem é decodificada, sendo gerado a primitiva de serviço.

Continuando o Exemplo01, o protocolo servidor recebe a primitiva dataTransp.indication(openSession.xml), indicando o recebimento da mensagem OpenSession.xml. O protocolo hipermídia servidor, realiza o processo de decodificação da mensagem XML e gera a primitiva Hyp_OpenSession.indication(abc, sampa, QoS, imperio, 1054, null).

Passo6: Enviando Primitiva de Serviço ao Servidor

O servidor recebe a primitiva de serviço *HYP_OpenSession.indication (user_name, password, profile, client_id, port_client, session_key)* e verifica se o cliente pode ser autenticado com as unidades funcionais solicitadas. Havendo sucesso no processo de autenticação, o servidor envia a primitiva *HYP_OpenSession.response (session_key²⁸, profile, client_id, port_client)* ao protocolo hipermídia servidor. Caso o cliente não seja autenticado, a primitiva *HYP_Error.response(message²⁹)* é retornada.

Continuando o Exemplo01, o servidor recebe a primitiva HYP_OpenSession.indication (abc, sampa, QoS, imperio, 1054, null), autentica o cliente com a unidade funcional Qualidade de Serviço - QoS, cria uma sessão de aplicação, gera uma chave de

²⁸ O atributo *session_key* identifica a sessão de aplicação criada.

²⁹ O parâmetro *message* justifica o por quê do serviço solicitado não ter sido oferecido.

autenticação (*session_key* - *abc@tradicao1054*) e retorna a primitiva *HYP_OpenSession.response(abc@tradicao1054, QoS, imperio, 1054)*.

Passo7: Codificando mensagem XML

O protocolo hipermídia servidor, ao receber a primitiva enviada pela aplicação servidora, codifica os parâmetros dessa primitiva em um documento XML válido e bem-formatado.

Dando continuidade ao Exemplo01, o protocolo hipermídia servidor recebe a primitiva HYP_OpenSession.response(abc@tradicao1054, QoS, imperio, 1054) e gera a mensagem OpenSession.xml.

Passo8: Enviando mensagem XML ao Cliente

Utilizando qualquer protocolo de transporte, definida pela camada de adaptação do Modelo de Referência do protocolo (ver Capítulo3, Seção 3.1), o servidor envia a mensagem XML.

Seguindo o Exemplo01, o protocolo de transporte envia a mensagem OpenSession.xml para o cliente. A Figura 4.3.1.D exemplifica todo o processo realizado pelo protocolo hipermídia servidor para criar uma Sessão de Aplicação.

Passos Realizados no Cliente:

Passo9: Recebendo mensagem XML do Servidor

O protocolo de transporte cliente, recebe a mensagem XML enviada pelo servidor.

Continuando o Exemplo01, o protocolo de transporte cliente recebe a mensagem OpenSession.xml.

Passo10: Decodificando mensagem XML recebida

O protocolo hipermídia cliente recebe do protocolo de transporte uma primitiva de serviço (*dataTransp.confirm(message)*), indicando o recebimento de uma mensagem XML. A mensagem XML é decodificada, sendo gerado uma primitiva de serviço.

Dando sequência ao Exemplo01, o protocolo cliente hipermídia recebe a primitiva `dataTransp.confirm(openSession.xml)`, indicando o recebimento da mensagem `OpenSession.xml`. O protocolo hipermídia cliente, realiza o processo de decodificação da mensagem XML e gera a primitiva `Hyp_OpenSession.confirm(abc@tradicao1054)`.

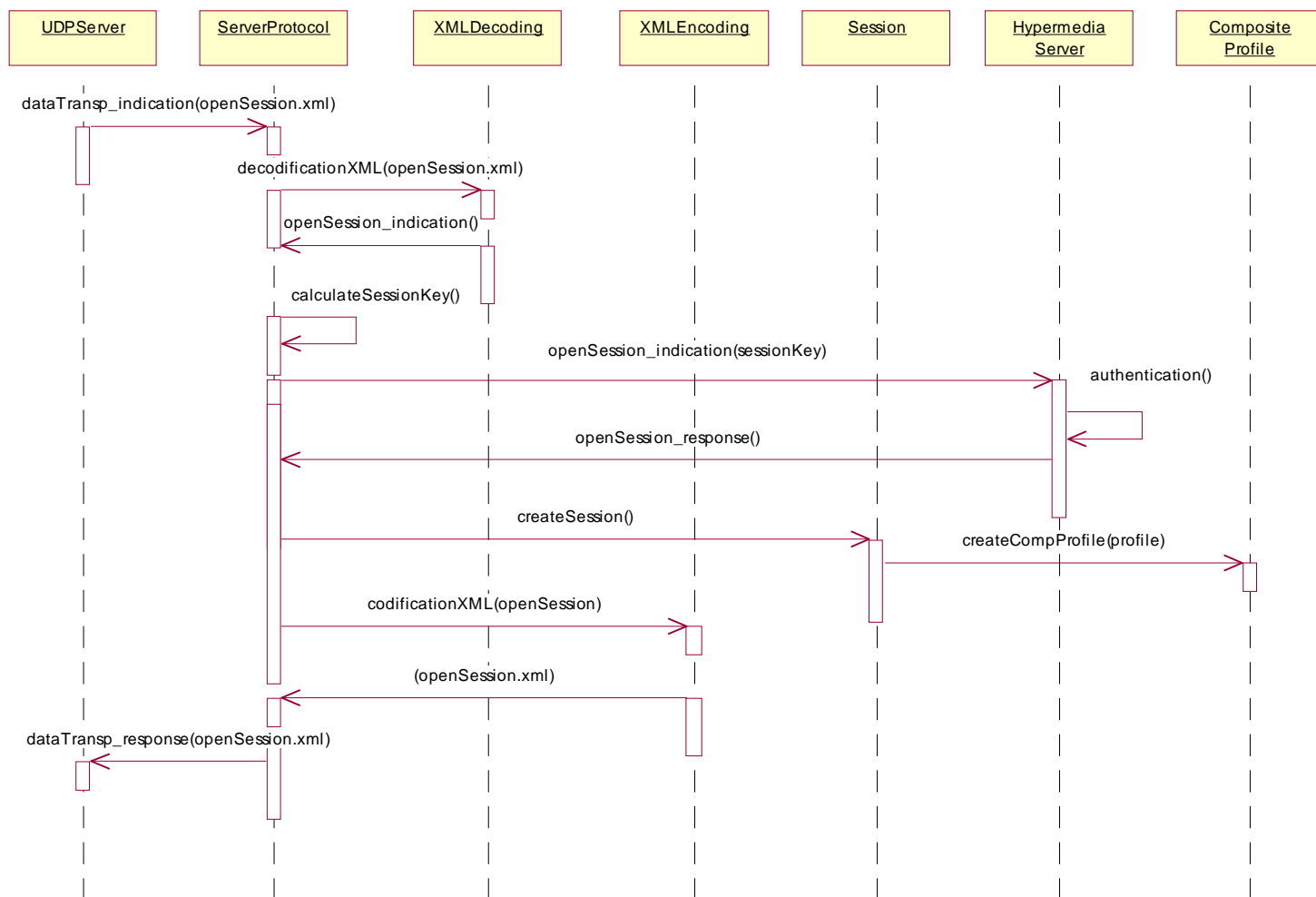


Figura 4.3.1.D – Diagrama de Sequência realizado pelo Servidor para criar uma Sessão de Aplicação

Passo11: Enviando Primitiva de Serviço ao Cliente

O cliente recebe a primitiva de serviço `HYP_OpenSession.confirm(session_key)`.

Continuando o Exemplo01, o cliente recebe a primitiva `HYP_OpenSession.confirm(abc@tradicao1054)`, indicando que o serviço solicitado (abertura de Sessão) foi realizado com sucesso. A Figura 4.3.1.E exemplifica o processo realizado pelo protocolo

hipermídia cliente para receber a confirmação de que a Sessão de Aplicação solicitada foi criada com sucesso.

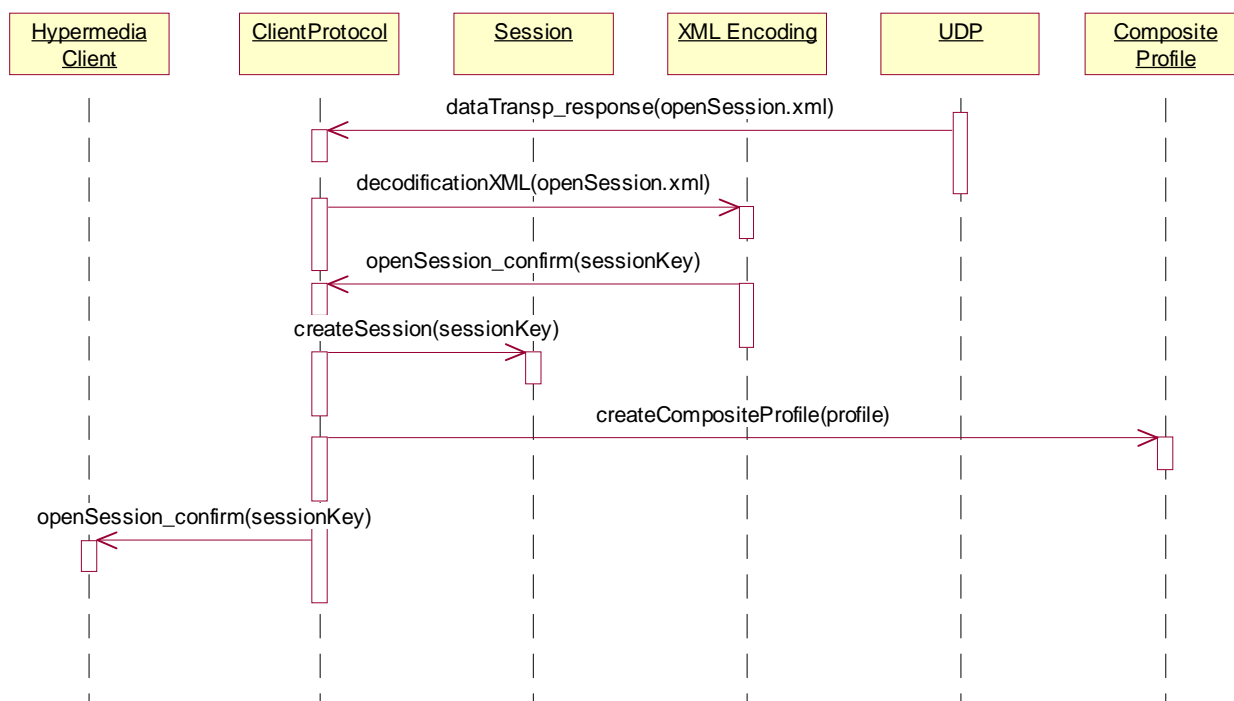


Figura 4.3.1.E – Diagrama de Sequência realizado pelo cliente para criar uma Sessão de Aplicação

Durante o processo de criação da Sessão de Aplicação, um *MediaPipe* de Controle deverá ser estabelecido, para permitir a troca de mensagens do protocolo entre as aplicações cliente e servidor pertencentes a esta sessão. A Figura 4.3.1.F ilustra a Sessão de Aplicação e o *MediaPipe* de Controle que foram criados entre o cliente (*imperio*) e o servidor (*tradicao*).

A Figura 4.3.1.G ilustra a interface gráfica utilizada pelo cliente (*imperio*) para visualizar a Sessão de Aplicação e o *MediaPipe* de Controle que foram criados com o servidor (*tradicao*).

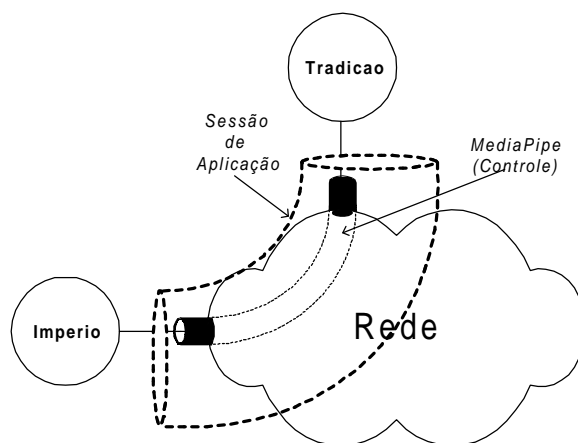


Figura 4.3.1.F – Sessão de Aplicação e o *MediaPipe* de Controle que foram criados entre o cliente (*imperio*) e o servidor (*tradicao*)

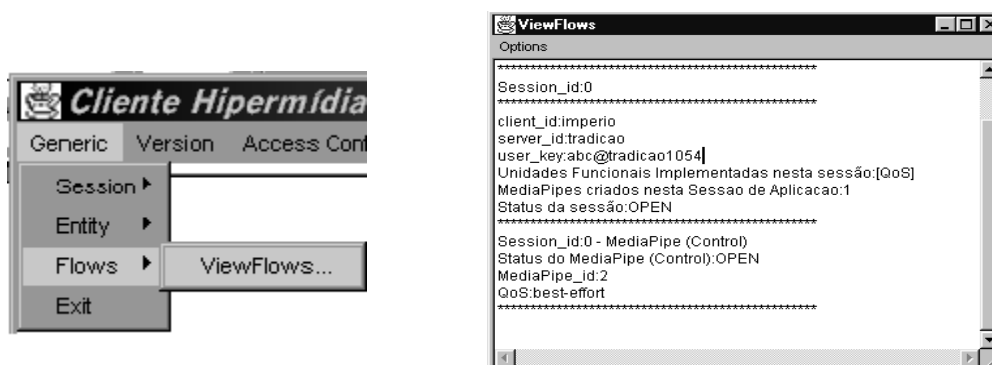


Figura 4.3.1.G – Janela do Protótipo para visualizar as Sessões e *MediaPipes* (Controle e Dados) criados pelo Cliente

Na janela "*ViewFlows*" da Figura 4.3.1.G, são mostradas informações sobre a Sessão de Aplicação e o *MediaPipe* Controle que foram criados entre o cliente (*imperio*) e o servidor (*tradicao*), tais como:

- **Identificador da Sessão** - identifica a Sessão de Aplicação criada. Neste exemplo, criou-se apenas uma sessão de aplicação, cujo identificador é zero (0);
- **Endereço Cliente** - identifica o cliente conectado na Sessão de Aplicação. Neste exemplo, o cliente (*imperio*) está conectado;
- **Endereço Servidor** - identifica o servidor conectado na Sessão de Aplicação. Neste exemplo, o servidor (*tradicao*) está conectado;

- **SessionKey** - chave de autenticação que identifica a Sessão de Aplicação criada. Neste exemplo, o valor de *SessionKey* (*abc@tradicao1054*) foi utilizado;
- **Unidades Funcionais** - identifica quais são as unidades funcionais utilizadas na Sessão de Aplicação criada. Neste exemplo, utilizam-se as unidades funcionais Genérico, obrigatória em qualquer Sessão de Aplicação (ver Seção 3.2.2), e Qualidade de Serviço (QoS);
- **MediaPipes criados** - identifica todos os *MediaPipes* criados na Sessão de Aplicação. Neste exemplo, criou-se apenas o *MediaPipe* Controle para a troca de mensagens XML do protocolo de comunicação hiperfídia;
- **Status da Sessão** - identifica o estado de uma sessão (ver Seção 3.2.6). Neste exemplo, a Sessão está no estado (*Open*);
- **Identifica o MediaPipe criado** - identifica o tipo de *MediaPipe* criado (Controle - *Control* ou Dados - *Data*). Neste exemplo, criou-se o *MediaPipe* Controle;
- **Status do MediaPipe** - identifica o estado de um *MediaPipe* (ver Seção 3.2.6). Neste exemplo, a Sessão está no estado (*Open*);
- **Identificador do MediaPipe** - Identifica o *MediaPipe* criado. Neste exemplo, o identificador utilizado foi dois (2);
- **QoS do MediaPipe** - identifica a QoS do *MediaPipe* criado. Pode assumir os seguintes valores: *best-effort* (padrão Internet); *VideoGuaranteed*; *AudioGuaranteed*; *DataGuaranteed*; *VideoControlled-load*; *AudioControlled-load*; e *DataControlled-load*. Neste exemplo, o *MediaPipe* Controle é *best-effort*.

Após a criação da Sessão de Aplicação, o cliente (*imperio*) pode solicitar quaisquer serviços de comunicação ao protocolo, utilizando as primitivas definidas pela composição das unidades funcionais Genérico e Qualidade de Serviço.

4.3.2. Solicitando o Transporte de um Documento com Garantias de Qualidade

Para solicitar qualquer serviço com QoS, as seguintes etapas devem ser realizadas [55]:

- (a) a aplicação (usuário) define a categoria de serviço (*best-effort*, *guaranteed* ou *controlled-load*) e os parâmetros de qualidade desejados;
- (b) os parâmetros de QoS devem ser negociados;
- (c) os parâmetros de qualidade devem ser traduzidos para cada

uma das diferentes camadas do sistema de comunicação; e (d) os recursos necessários devem ser reservados no caminho "fim-a-fim" entre o transmissor e o receptor.

*Exemplo02: o cliente (**imperio**) deseja receber um documento com garantias de qualidade na Sessão de Aplicação criada no Exemplo01. Desta forma, as etapas descritas nos itens (a), (b), (c) e (d) devem ser realizadas no protocolo de comunicação hipermídia.*

A seguir, descrevem-se quais são os passos necessários para a criação de um *MediaPipe* para transportar o documento solicitado com garantias de qualidade.

Passos Realizados no Cliente:

Passo1: Solicitando o Transporte de um Documento

Utilizando a primitiva *HYP_GetQoS.request (session_key, entity_id)*, o cliente especifica qual a Sessão de Aplicação a ser utilizada para a criação do *MediaPipe (session_key)* e qual o documento a ser recebido (*entity_id*).

Continuando o Exemplo02, a aplicação cliente (imperio.telemidia.puc-rio.br) solicita a transferência de um documento, que possui identificador 10, na Sessão de Aplicação (abc@tradicao1054) criada com o servidor (tradicao.telemidia.puc-rio.br). A Figura 4.3.2.A ilustra a interface gráfica utilizada pelo cliente para solicitar o recebimento do documento.

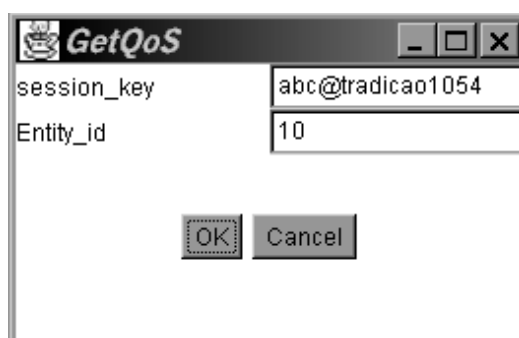


Figura 4.3.2.A – Janela do Protótipo para solicitar o recebimento de um Documento

Passo2: Codificando Mensagem XML

O protocolo hiperfídia cliente, ao receber a primitiva *HYP_GetQoS.request* (*session_key*, *entity_id*), codifica os parâmetros da primitiva, gerando um documento XML válido e bem-formatado.

Seguindo o exemplo anterior, a primitiva HYP_GetQoS.request(abc@tradicao1054, 10) será utilizada para gerar a mensagem GetQoS.XML (ver Figura 4.3.2.B).

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE GetQoS SYSTEM "HypProtocol.dtd">
<GetQoS>
  <session_key>abc@tradicao1054</session_key>
  <entity_id>10</entity_id>
  <TrafficSpec></TrafficSpec>
  <ResourceSpec></ResourceSpec>
</GetQoS>
```

Figura 4.3.2.B – Mensagem GetQoS.xml gerada pelo protocolo hiperfídia

Passo3: Envio da mensagem XML ao Servidor

Utilizando qualquer protocolo de transporte, o protocolo hiperfídia envia a mensagem XML ao servidor.

Dando continuidade ao Exemplo02, a mensagem GetQoS.xml é enviada ao servidor.

As ações realizadas no Cliente para solicitar o transporte do documento com garantias de qualidade estão ilustradas no Diagrama de Seqüência da Figura 4.3.2.C.

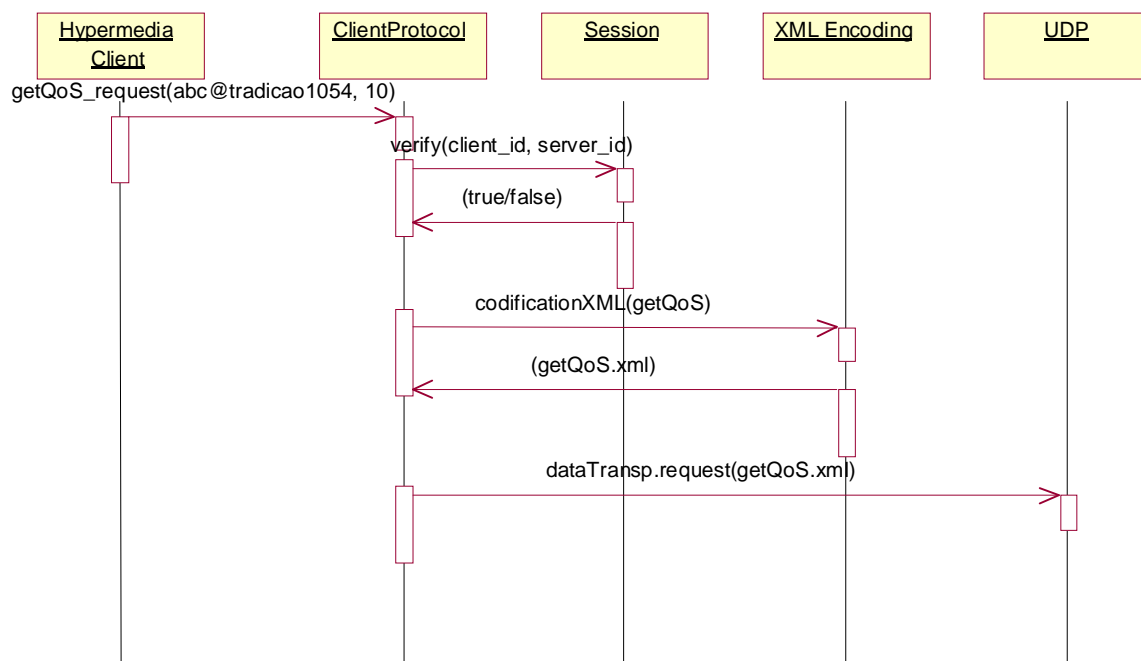


Figura 4.3.2.C – Diagrama de Sequência realizado pelo Cliente para solicitar o transporte de um documento com garantias de qualidade

Passos Realizados no Servidor:

Passo4: Recebendo mensagem XML do Cliente

Fazendo uso de qualquer protocolo de transporte, o protocolo hipermídia servidor recebe a mensagem XML.

Seguindo o Exemplo02, o protocolo de transporte recebe a mensagem GetQoS.xml.

Passo5: Decodificando mensagem XML recebida

O protocolo hipermídia servidor recebe do protocolo de transporte uma primitiva de serviço (*dataTransp.indication(message)*), indicando o recebimento de uma mensagem XML, decodifica-a e gera a primitiva de serviço correspondente.

Continuando o Exemplo02, o protocolo servidor recebe a primitiva dataTransp.indication(getQoS.xml), indicando o recebimento da mensagem GetQoS.xml. O protocolo hipermídia servidor, realiza o processo de decodificação da mensagem XML e gera a primitiva Hyp_GetQoS.indication(abc@tradicao1054, 10).

Passo6: Enviando Primitiva de Serviço ao Servidor

O servidor recebe a primitiva de serviço *HYP_GetQoS.indication* (*session_key*, *entity_id*), verifica se o documento solicitado (*entity_id*) existe e quais os parâmetros de descrição de tráfego associados ao mesmo. Se o documento solicitado existir, a primitiva *HYP_GetQoS.response* (*session_key*, *entity_id*, *trafficSpec*³⁰, *resourceSpec*³¹) é retornada ao protocolo hiperfídia servidor; caso contrário, a primitiva *HYP_Error.response(message)* é retornada.

Seguindo o Exemplo02, o servidor recebe a primitiva HYP_GetQoS.indication(abc@tradicao1054, 10), verifica que o documento solicitado (identificador 10) existe, identifica o seu conteúdo (áudio – ‘sound.wav’) e analisa os parâmetros de descrição de tráfego associados a este documento (sampleRate – 8Khz, sampleSize – 8bits, Loudness – Stereo e audioCodification - PCM). Para a sua transmissão com estes parâmetros de tráfego, o servidor exige um retardo de 4 ms. A primitiva HYP_GetQoS.response (abc@tradicao1054, sound.wav, (8Khz, 8bits, Stereo, PCM), 4 ms) é então retornada.

Passo7: Recebendo Primitiva de Serviço do Servidor

O protocolo hiperfídia servidor recebe a primitiva *HYP_GetQoS.response* (*session_key*, *entity_id*, *trafficSpec*, *resourceSpec*), calcula o retardo que irá oferecer para processar os pacotes de dados a serem transmitidos pelo servidor e inicia o processo de anúncio de tráfego.

*O protocolo hiperfídia recebe a primitiva HYP_GetQoS.response (abc@tradicao1054, sound.wav, (8Khz, 8bits, Stereo, PCM), 4 ms), verifica que exigirá um retardo de 1ms, atualiza o valor do parâmetro resourceSpec, de 4ms para 5 ms (retardo_servidor + retardo_protocolo_hiperfídia_servidor), e inicia o processo de anúncio de tráfego. Para que isto seja feito, o protocolo hiperfídia servidor deverá realizar o **mapeamento dos***

³⁰ Identifica os parâmetros de descrição de tráfego associados a um documento.

³¹ Identifica quais são os parâmetros de desempenho de cada entidade (servidor, protocolo hiperfídia servidor, protocolo hiperfídia cliente e cliente) e de cada provedor de serviços.

parâmetros de descrição de tráfego, para permitir que todos os subsistemas envolvidos na comunicação fim-a-fim compreendam os parâmetros especificados.

A seguir, descrevemos como os mecanismos de **mapeamento de parâmetros e anúncio de tráfego**, modelados no sistema de *software Quality*, instanciado no *Framework para Compartilhamento de Recursos*, são realizados no protótipo [53] do protocolo de comunicação hipermídia desenvolvido.

Passo8: Realizando Mapeamento dos Parâmetros de Descrição de Tráfego

Os parâmetros de descrição de tráfego especificados, nível aplicação, deverão ser mapeados em parâmetros nível transporte, para ser possível iniciar o processo de anúncio de tráfego e verificar qual a quantidade de recursos que serão alocados pela a infraestrutura de comunicação para transportar o documento solicitado com garantias de qualidade.

Seguindo o Exemplo02, os parâmetros de tráfego nível de aplicação, especificados na primitiva HYP_GetQoS.response (abc@tradicao1054, sound.wav, (8Khz, 8bits, Stereo, PCM), 5 ms), deverão ser mapeados em parâmetros nível transporte, para ser possível iniciar o processo de anúncio de tráfego. Neste trabalho, foi definida uma regra de mapeamento específica para áudio PCM, conforme ilustrada na Figura 4.3.2.D.

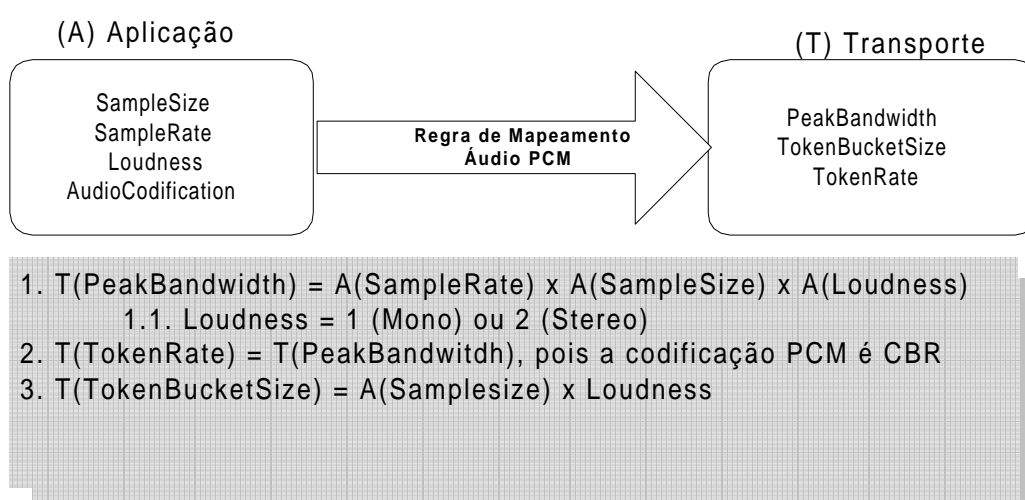


Figura 4.3.2.D – Regra de Mapeamento para Áudio codificação PCM

No exemplo acima descrito, o documento solicitado ('sound.wav') utiliza a codificação PCM. Neste caso, os parâmetros de descrição de tráfego nível de aplicação especificados (8Khz, 8bits, Stereo), deverão ser mapeados em parâmetros de transporte, utilizando para isto a regra definida na Figura 4.3.2.D. Os parâmetros nível de transporte obtidos foram:

- *PeakBandwidth* = 128 Kb/s
- *TokenBucketSize* = 16 bits/pixel
- *TokenRate* = 128 Kb/s

Tendo sido realizado o mapeamento dos parâmetros com sucesso, utiliza-se as primitivas definidas na API IPQoS[54]³² para solicitar à infra-estrutura de comunicação um serviço de transmissão compatível com os parâmetros de tráfego especificados.

Passo9: Iniciando o Anúncio do Tráfego

Uma vez realizado o mapeamento dos parâmetros de qualidade, do nível de aplicação para os de nível de transporte, o protocolo hipermídia servidor descobre, junto à camada de adaptação, qual protocolo de reserva de recursos deve ser utilizado para iniciar o processo de anúncio de tráfego.

Continuando o Exemplo02, o protocolo hipermídia servidor, através de sua camada de adaptação, utiliza o protocolo RSVP para iniciar o processo de anúncio do tráfego para a transmissão do documento ('sound.wav') com os seguintes parâmetros: 128 Kb/s, 16 bits/pixel e 128 Kb/s. Como mencionado, o protótipo desenvolvido usa a API IPQoS, que implementa o RSVP. Assim sendo, para solicitar serviços a esta interface, o protótipo deve mapear os parâmetros de transporte em parâmetros específicos RSVP, utilizando para isto a regra definida na Figura 4.3.2.E. Nas futuras atualizações desta API, novos protocolos de reserva de recursos deverão estar disponíveis.

³² IPQoS é a implementação em Java de uma interface de programação para suporte à QoS em uma infra-estrutura Internet. A versão utilizada, 1.02, inclui somente negociadores de QoS do tipo "intserv-RSVP", isto é, utiliza somente o protocolo de reserva de recursos RSVP.

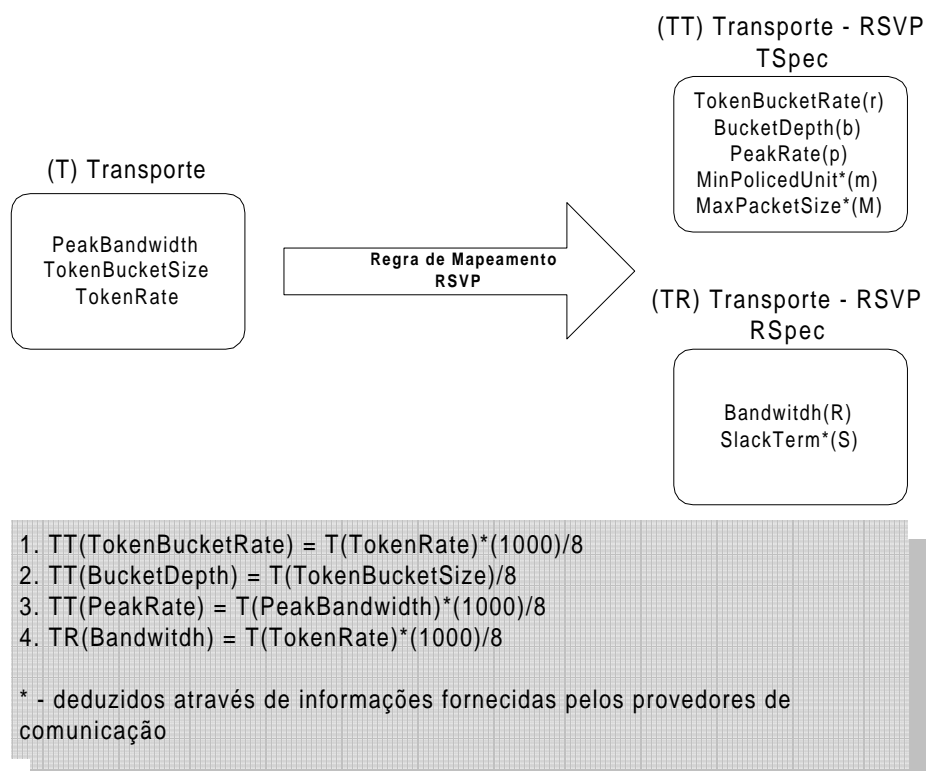


Figura 4.3.2.E – Regra de Mapeamento dos Parâmetros de Transporte para Parâmetros RSVP

Os parâmetros RSVP obtidos foram:

- $\text{TokenBucketRate} = 16000 \text{ B/s}$
- $\text{BucketDepth} = 2 \text{ B}$
- $\text{Bandwidth} = 16000 \text{ B/s}$
- $\text{MinPolicedSize}^{33} = 128 \text{ B}$
- $\text{MaxPacketSize} = 1500 \text{ B}$

Utilizando-se a primitiva `IPQoS.sender(RSVPsession, end_server, TSpec)` da API `IPQoS`, inicia-se o processo de anúncio do tráfego que será gerado pelo servidor hipermídia para transmitir o documento solicitado com garantias de qualidade e que deverá ser suportado pela infra-estrutura de comunicação. Este anúncio de tráfego (16000 B/s, 2 B, 16000 B/s, 128 B, 1500 B) é realizado através do envio de mensagens `PATH` para o

cliente (*imperio*). As mensagens PATH enviadas pelo servidor (*tradicao*) estão ilustradas na Figura 4.3.2.F. Maiores detalhes sobre o funcionamento do RSVP podem ser encontrados em [31, 32, 34, 43, 54].

```

Sinalizador RSVP Servidor
RSVP Inicializado...
-->Sessão RSVP criada
-->Anunciando Tráfego
-->tSpec={r=16000.0;b=2.0;p=16000.0;m=128;M=1500}
-->Enviou mensagem de Sinalização RSVP (PATH)
-->Path Message Send
-> senderNo=1 [139.82.95.15:0]
-> tSpec=[r=16000.0;b=2.0;p=16000.0;m=128;M=1500]
-> adSpec=[hopCount=0;pathBW=0.0;minLatency=0.0;composedMTU=0]

```

Figura 4.3.2.F – Sessão RSVP Servidor enviando mensagens PATH

Após ter realizado com sucesso as etapas de mapeamento de parâmetros e anúncio de tráfego, o protocolo hipermídia servidor irá codificar os parâmetros da primitiva *HYP_GetQoS.response* (*session_key*, *entity_id*, *trafficSpec*, *resourceSpec*) em uma mensagem *GetQoS.xml*³⁴, e enviá-la ao cliente (*imperio*). Caso alguma das etapas não tenha sido realizada com sucesso, a primitiva *HYP_Error(message)* é retornada, sendo gerado a mensagem *Error.xml*.

É importante ressaltar que a mensagem *GetQoS.xml* não foi encapsulada na mensagem PATH porque na implementação do protótipo desenvolvido [53], conforme mencionado, utilizou-se a API *IPQoS* que não oferece suporte à inclusão de elementos opcionais nas mensagens PATH, embora o protocolo RSVP especifique esta funcionalidade.

Passo10: Enviando Mensagem XML ao Cliente

Utilizando qualquer protocolo de transporte, o servidor envia a mensagem XML.

³³ Os parâmetros *MinPolicedSize* e *MaxPacketSize* são definidos pelos provedores de infra-estrutura. Neste exemplo, considerou-se os valores normalmente utilizados em redes Internet [18].

Continuando o Exemplo02, o protocolo de transporte utilizado pelo servidor envia a mensagem *GetQoS.xml* (ver Figura 4.3.2.G).

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE GetQoS SYSTEM "HypProtocol.dtd">
<GetQoS>
  <session_key>abc@tradicao1054</session_key>
  <entity_id>sound.wav</entity_id>
  <TrafficSpec>
    <sampleRate>8KHz</sampleRate>
    <sampleSize>8bits</sampleSize>
    <loudness>Stereo</loudness>
    <audioCodification>PCM</audioCodification>
  </TrafficSpec>
  <ResourceSpec>5ms</ResourceSpec>
</GetQoS>
```

Figura 4.3.2.G – Mensagem *GetQoS.xml* gerada pelo protocolo hipermídia

As ações realizadas no Servidor estão ilustradas no Diagrama de Seqüência da Figura 4.3.2.H. A seguir detalhamos a seqüência de passos realizadas no Cliente.

Passos Realizados no Cliente:

Passo11: Recebendo mensagem XML do Servidor

Fazendo uso de qualquer protocolo de transporte, o cliente recebe a mensagem XML.

Seguindo o Exemplo02, o protocolo de transporte utilizado pelo cliente recebe a mensagem GetQoS.xml.

Passo12: Decodificando mensagem XML recebida

O protocolo hipermídia cliente recebe as primitivas (*dataTransp.confirm(message)*), indicando o recebimento de uma mensagem XML, e (*IPQoS.dispatch(QoSEvent)*), indicando o recebimento da mensagem PATH. Decodifica a mensagem XML e gera outra primitiva de serviço.

³⁴ Esta mensagem possui os parâmetros de descrição de tráfego (*trafficSpec*) e de desempenho dos fornecedores (*resourceSpec* – servidor e protocolo hipermídia servidor) que serão utilizados para transmitir o documento solicitado.

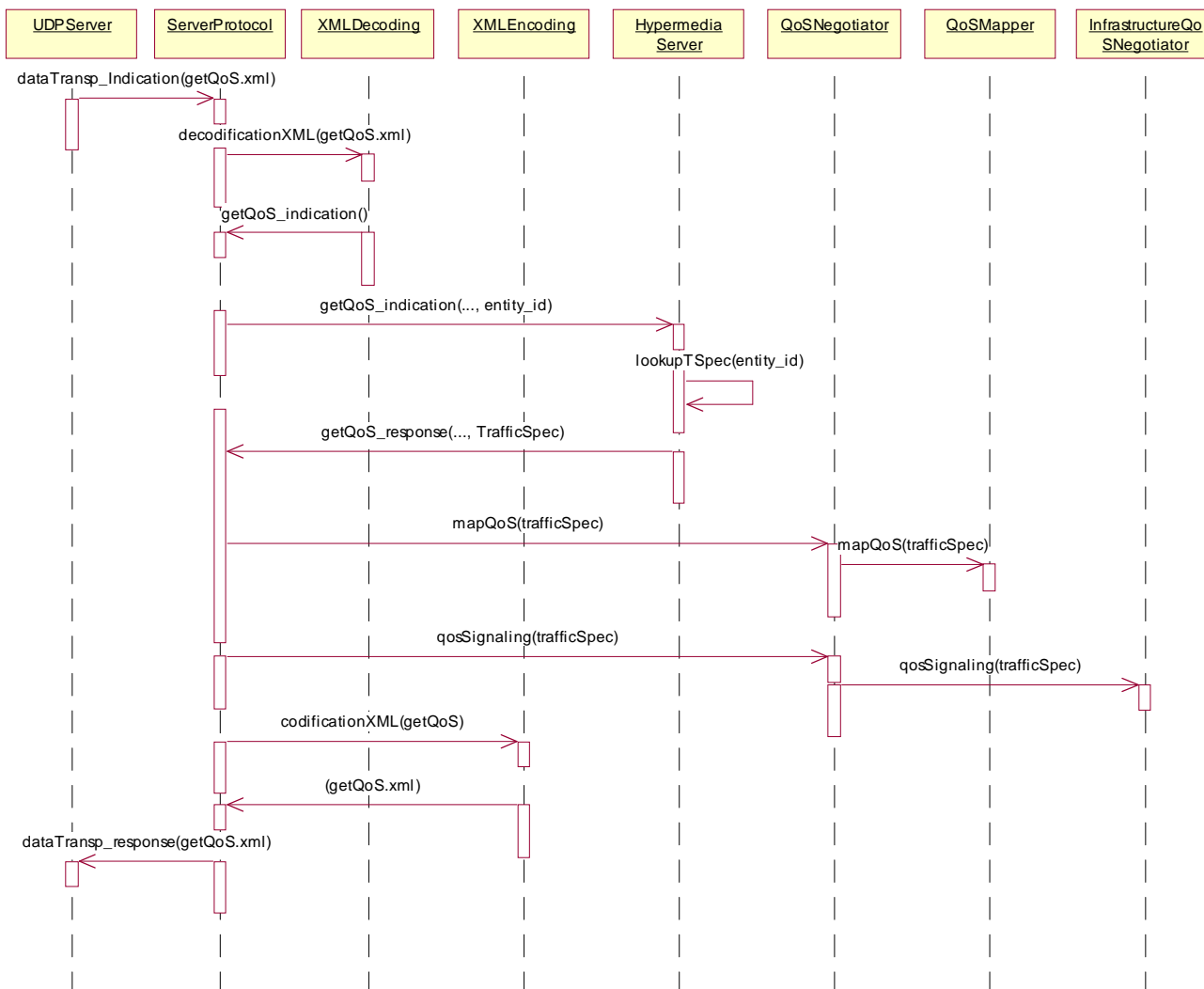


Figura 4.3.2.H – Diagrama de Sequência realizado pelo Servidor para iniciar o anúncio do tráfego que será gerado para transportar o documento solicitado com garantias de qualidade

Dando sequência ao Exemplo02, o protocolo hipermídia cliente recebe as primitivas *dataTransp.confirm(getQoS.xml)*, indicando o recebimento da mensagem *GetQoS.xml*, e *IPQoS.dispatch(PathMessageArrived)*, indicando o recebimento da mensagem *PATH*. A primeira mensagem, *GetQoS.xml*, identifica qual o documento solicitado (áudio 'sound.wav') e quais os parâmetros de descrição de tráfego (8Khz, 8bits, Stereo, PCM) e de desempenho dos fornecedores (retardo de 5 ms) necessários para transmitir este documento com garantias de QoS. A segunda mensagem, *PATH*, identifica qual o tráfego que será gerado na infra-estrutura de comunicação para transmitir este documento. Opcionalmente, pode possuir um objeto *AdsSpec* que identifica quais são as

características³⁵ da infra-estrutura de comunicação no caminho fim-a-fim. A Figura 4.3.2.I ilustra o recebimento da mensagem PATH pelo cliente (imperio).

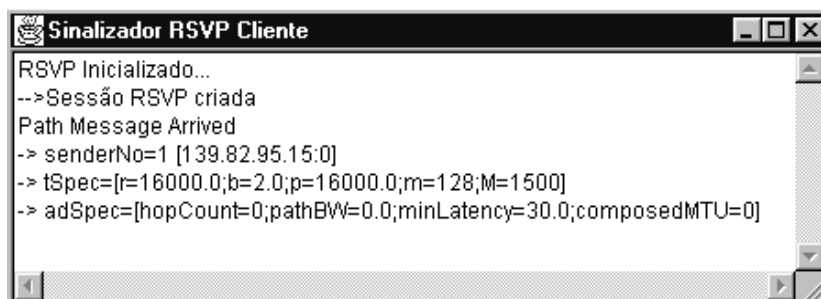


Figura 4.3.2.I – Sessão RSVP Cliente recebendo mensagem PATH

A mensagem PATH recebida possui o objeto *Adspec* que contém o parâmetro 'minimum_path_latency', responsável por determinar o retardo fim-a-fim na infra-estrutura de comunicação utilizada. Neste exemplo, o valor obtido foi de 30 ms.

Após ter recebido estas duas mensagens, o protocolo hiperâmida cliente realiza o processo de decodificação da mensagem XML, calcula o retardo total oferecido no AOS analisado, *retardo_fornecedor_serviço* (5ms) + *retardo_infra_estrutura* (30 ms) + *retardo_protocolo_hiperâmida_cliente* (2 ms), e envia a primitiva *HYP_GetQoS.confirm* (*abc@tradicao1054*, *sound.wav*, (8Khz, 8bits, Stereo, PCM), 37 ms), ao cliente hiperâmida.

Passo13: Enviando Primitiva de Serviço ao Cliente

O cliente recebe a primitiva de serviço *HYP_GetQoS.confirm* (*session_key*, *entity_id*, *trafficSpec*, *resourceSpec*), verifica que o documento solicitado (*entity_id*) existe e analisa quais são os parâmetros de descrição de tráfego (*trafficSpec*) e de alocação de recursos (*resourceSpec*) associados à transmissão do documento com garantias de qualidade. Com base nestas informações, o cliente inicia o processo de criação do *MediaPipe* para transportar às informações solicitadas.

³⁵ Retardo fim-a-fim, total de banda passante, número de roteadores, etc.

Continuando o Exemplo02, a aplicação cliente recebe a primitiva HYP_GetQoS.confirm (abc@tradicao1054, sound.wav, (8Khz, 8bits, Stereo, PCM), 37 ms), verifica que o documento solicitado existe ('sound.wav') e analisa quais são os parâmetros de descrição de tráfego (8Khz, 8bits, Stereo, PCM) e de alocação de recursos (37 ms) associados à transmissão do mesmo.

Na próxima Seção 4.4, descrevem-se as várias etapas realizadas para a criação do *MediaPipe* para a transferência da *stream* de dados solicitada, com a categoria de serviço e os parâmetros de qualidade especificados.

4.4. Cenário de Estabelecimento de Contratos de Serviço

Neste Cenário, descrevem-se todos os passos realizados pelas aplicações e pelos protocolos de comunicação hipermídia para a criação de um *MediaPipe* com a categoria de serviço e os parâmetros de QoS especificados. Criar o *MediaPipe* significa estabelecer um contrato de serviço, também conhecido como SLA - *Service Level Aggrement* [31], entre o Cliente e o Servidor.

A seguir detalhamos a seqüência de passos realizados no Cliente e no Servidor para criar um *MediaPipe* para o transporte de qualquer tipo de informação.

Passos Realizados no Cliente:

Passo1: Realizando o Controle de Admissão no Cliente

*Ao solicitar um serviço, um contrato implícito está sendo configurado entre a aplicação que faz a solicitação, entre a aplicação que irá responder à solicitação e entre os provedores de comunicação que irão assegurar o transporte das informações com a qualidade de serviço especificada. O provedor de comunicação utilizado, deverá oferecer suporte ao estabelecimento do contrato de serviço (SLA), entre o cliente (**imperio**) e o servidor (**tradicao**). Estabelecer contrato de serviço entre entidades significa criar um *MediaPipe* com a QoS especificada entre as mesmas.*

*Dando continuidade ao Exemplo02, a aplicação hipermídia cliente é responsável por iniciar o processo de **controle de admissão**, para verificar se os parâmetros de descrição*

de tráfego e de alocação de recursos especificados (8Khz, 8bits, Stereo, PCM), 37 ms) podem ser oferecidos por todos os elementos (cliente, protocolo hipermídia cliente, protocolo hipermídia servidor e servidor) e pelo provedor de serviços envolvidos na comunicação com QoS fim-a-fim.

Seguindo a abordagem descrita em Klara Nahrstedt [49], os testes de controle de admissão que devem ser realizados no *host* (cliente) são quatro:

1. **device quality test**, verificando se os dispositivos multimídia existentes no cliente suportam os requisitos de QoS especificados;
2. **local schedulability test**³⁶, verificando se o sistema operacional possui recursos necessários para suportar a *stream* (fluxo) multimídia solicitada;
3. **end-to-end delay test**, verificando se o tempo para realizar todas as tarefas está definido em um intervalo condizente com a QoS solicitada;
4. **buffer allocation test**, verificando se existe espaço suficiente na memória para alocação de buffers.

O cliente realiza os quatro testes de controle de admissão acima descritos e verifica se é possível receber o documento solicitado com os parâmetros de descrição de tráfego e de alocação de recursos especificados. Se os quatro testes forem realizados com sucesso, especifica qual a categoria de serviço e quais os parâmetros de qualidade desejados no *MediaPipe* a ser criado para receber às informações solicitadas. Em seguida, envia a primitiva *HYP_Reserve.request (session_key, entity_id, serviceQoS)* solicitando a reserva de todos os recursos necessários para a criação do *MediaPipe*.

Conforme foi descrito na Seção 3.3, as aplicações solicitam serviços de comunicação com QoS a partir de classes ou categorias de serviços padronizadas. Utilizando a modelagem do *Framework para Parametrização de Serviços*, apresentada na Seção 3.3.2.1, foram definidos seis categorias de serviços: *AudioGuaranteed*, *VideoGuaranteed*, *DataGuaranteed*, *AudioControlled-load*, *VideoControlled-load* e *DataControlled-load*.

³⁶ Antonio Tadeu [14], desenvolveu o controle de admissão do tipo (*local schedulability test*), através de uma biblioteca de escalonamento para *threads* em máquinas virtuais Java.

Cada categoria possui um conjunto de parâmetros de qualidade associados. No caso do transporte de um áudio em tempo-real, com todas as garantias de qualidade, deve-se utilizar a categoria de serviço *AudioGuaranteed*, que possui os seguintes parâmetros de QoS associados:

1. *maxDelay* (retardo máximo fim-a-fim);
2. *sampleRate* (taxa de amostragem);
3. *sampleSize* (tamanho da amostragem);
4. *Loudness* (altura do som);
5. *audioCodification* (codificação do áudio).

Dando continuidade ao Exemplo02, a aplicação hipermídia realiza os quatro teste de controle de admissão com sucesso, verificando que é possível receber o áudio solicitado com os parâmetros de descrição de tráfego e de alocação de recursos especificados. Calcula o seu retardo (3 ms) para receber e apresentar o áudio solicitado em tempo-real ao usuário final, reserva recursos correspondentes a este valor e solicita aos outros elementos a reserva de todos os recursos necessários para a criação do MediaPipe com a categoria de serviço ('AudioGuaranteed') e os parâmetros de QoS (maxDelay – 60 ms³⁷ requisitado – 3 ms reservado = 57 ms, sampleRate – 8Khz, sampleSize – 8 bits, Loudness – Stereo e audioCodification – PCM) na Sessão de Aplicação (abc@tradicao1054) criada com o servidor (tradicao.telemidia.puc-rio.br). A primitiva HYP_Reserve.request(abc@tradicao1054, sound.wav, AudioGuaranteed, 57 ms, 8Khz, 8bits, Stereo, PCM) é enviada ao protocolo hipermídia cliente.

Passo2: Realizando Controle de Admissão no Protocolo

O protocolo de comunicação hipermídia recebe do cliente a primitiva de serviço *HYP_Reserve.request (session_key, entity_id, serviceQoS)*. O protocolo modela os controles de admissão do tipo 2, 3, e 4, conforme ilustrado na Figura 3.3.2.2, e o protótipo desenvolvido implementa o controle do tipo 3. Futuras extensões do protótipo poderão implementar os demais mecanismos.

³⁷ Note que o parâmetro *maxDelay* deve ser maior ou igual ao retardo fim-a-fim anteriormente calculado, uma vez que o sistema pode, na melhor hipótese, garantir esse último valor. Este valor, entretanto, pode ser menor do que o necessário para aplicação executar a contento.

Dando continuidade ao Exemplo02, ao receber a primitiva HYP_Reserve.request(abc@tradicao1054, sound.wav, AudioGuaranteed, 57 ms, 8Khz, 8bits, Stereo, PCM), o protocolo hipermídia cliente verifica se já existe algum MediaPipe áudio criado com a QoS especificada. Se houver, verifica se o MediaPipe está sendo utilizado. Não havendo uso do MediaPipe, este será utilizado para o transporte do áudio solicitado em tempo-real com garantias de qualidade. Caso contrário, inicia-se o processo de criação de um novo MediaPipe.

*No Exemplo02, não existe nenhum MediaPipe áudio criado. Assim sendo, o protocolo de comunicação hipermídia cliente realiza o controle de admissão, verificando a necessidade de um retardo de (2 ms) para processar os pacotes de áudio em tempo-real. Reserva então esse valor e continua o processo de criação do MediaPipe, com a categoria de serviço ('AudioGuaranteed') e com os parâmetros de QoS (maxDelay – 55ms (60 ms_{requisitado} – 5 ms_{reservado}), sampleRate – 8Khz, sampleSize – 8bits, Loudness – Stereo e audioCodification - PCM), entre o cliente (**imperio**) e o servidor (**tradicao**). Para que isto seja feito, o protocolo hipermídia cliente deverá realizar o **mapeamento de parâmetros de QoS**, para permitir que todos os elementos envolvidos na comunicação fim-a-fim compreendam os parâmetros de qualidade solicitados, e, posteriormente, iniciar o processo de **reserva de recursos**.*

Passo3: Realizando o Mapeamento dos Parâmetros de Qualidade

Após ter sido realizado o controle de admissão no cliente e no protocolo de comunicação hipermídia cliente, deve ser verificado se o ambiente de comunicação possui recursos suficientes para oferecer a QoS desejada.

Seguindo o Exemplo02, o protocolo de comunicação hipermídia cliente irá mapear os parâmetros de aplicação (AudioGuaranteed, 55 ms, 8 KHz, 8 bits, Stereo) nos parâmetros de transporte (Guaranteed, 128 Kb/s, 55 ms, 16 bits/pixel, 128 Kb/s). Posteriormente, os parâmetros de transporte são mapeados em parâmetros específicos do RSVP (Guaranteed, 55 ms, 16000 B/s, 2 B, 16000 B/s, 128 B, 1500 B).

Passo4: Realizando a Reserva de Recursos

Após ter realizado com sucesso o mapeamento dos parâmetros de qualidade, o protocolo hipermídia cliente irá enviar a mensagem de reserva RESV, percorrendo o caminho inverso da mensagem PATH.

Para enviar a mensagem de reserva RESV, devem ser calculados: o tráfego a ser gerado pelo transmissor (Tspec) e os parâmetros de QoS a serem reservados pelo provedor de serviços (Rspec).

O objeto Tspec é informado pelas mensagens PATH recebidas e o objeto Rspec deve ser calculado pelo cliente, após o recebimento de mensagens PATH.

Se a categoria de serviço (*serviceCategory*) escolhida for '*Controlled-load*', o objeto Rspec é *null* [31]. Se a categoria de serviço for '*Guaranteed*', os parâmetros de Rspec devem ser calculados, utilizando-se as seguintes equações [48]:

$$(1) \text{MaxDelay} = \frac{(b - M)(p - R)}{R(p - r)} + \frac{(M + Ctot)}{R} + Dtot, \text{ caso}(p > R \geq r)$$

$$(2) \text{MaxDelay} = \frac{(M + Ctot)}{R} + Dtot, \text{ caso}(p > R \geq r), \text{ onde:}$$

p (*peak rate of flow*), b (*bucket depth*), r (*token bucket rate*), m (*minimum policed unit*), M (*maximum datagram size*) são parâmetros Tspec; e R (*bandwidth*), S (*slack term*) são parâmetros Rspec. Maiores detalhes sobre a obtenção dessas fórmulas podem ser encontrados em [31, 48].

R identifica o total de banda passante a ser alocado nos roteadores pertencentes à infraestrutura de comunicação. S representa a diferença do retardo máximo de enfileiramento que a rede pode fornecer com o retardo máximo solicitado pela aplicação do usuário. As variáveis $Ctot$ e $Dtot$ são calculadas nos roteadores RSVP pertencentes à infra-estrutura de comunicação utilizada. Essas variáveis calculam os atrasos introduzidos pelos níveis inferiores da rede, como o enquadramento a nível de enlace e o acesso ao meio. Maiores detalhes podem ser encontrados em [31].

Após o recebimento de mensagens PATH, o cliente deve calcular os parâmetros R e S para ser possível o envio da mensagem RESV. O cálculo de R pode ser obtido pelo seguinte algoritmo [48]:

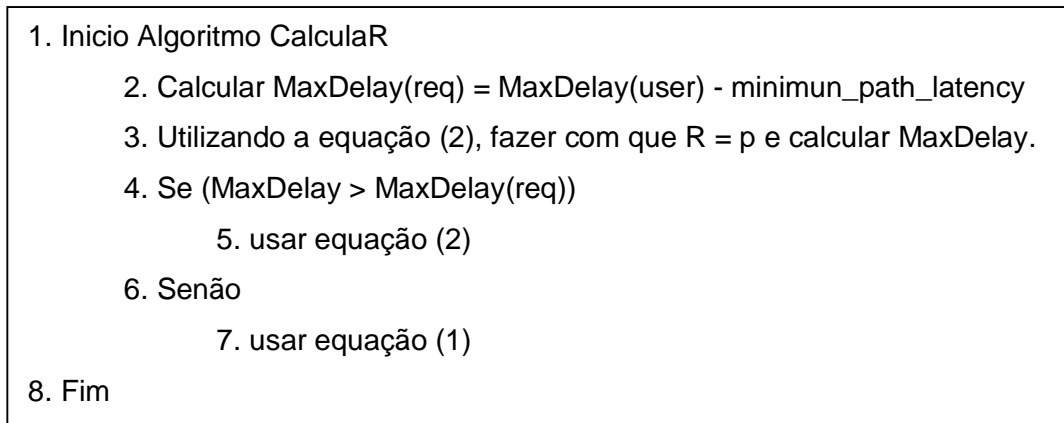


Figura 4.4.A – Algoritmo para calcular o parâmetro R do protocolo RSVP

Dando seqüência ao Exemplo02, para solicitar a transferência do áudio em tempo-real com a categoria de serviço 'Guaranteed', o protocolo cliente hipermídia deverá calcular os parâmetros R e S do objeto RSpec, utilizando o algoritmo CalculaR():

(a) $MaxDelay(req) = MaxDelay(user) - MaxDelay(infra_estrutura) = 60\ ms - 30\ ms = 30ms;$

(b) *Utilizando a equação (2), temos: $R = p = 16000\ B/s;$*

(c) $MaxDelay = \frac{(M + Ctot)}{R} + Dtot \rightarrow MaxDelay = \frac{(1500 + 0)}{16000} + 0 \rightarrow 93ms$

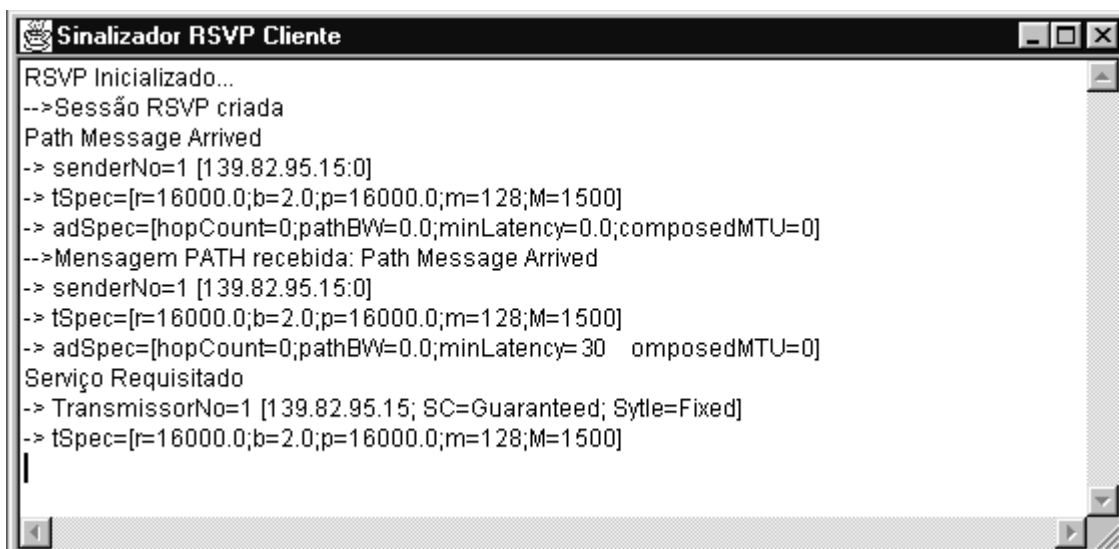
(d) $MaxDelay(93\ ms) > MaxDelay(req)(30\ ms) \rightarrow Utilizar\ a\ equação\ (2)$

(e)

$$MaxDelay = \frac{(M + Ctot)}{R} + Dtot \rightarrow R = \frac{(M + Ctot)}{MaxDelay} + Dtot \rightarrow R = \frac{(1500 + 0)}{93} = 16129\ B / s$$

Por aproximação, o valor de R utilizado foi 16000 B/s. Segundo [48], o valor de S deve ser iniciado com zero.

Utilizando-se as primitivas definidas na API IPQoS, o protocolo hiperfídia cliente envia a mensagem RESV com os parâmetros de qualidade acima especificados. A Figura 4.4.B ilustra o envio da mensagem RESV pelo cliente (*imperio*) ao servidor (*tradicao*).



```

Sinalizador RSVP Cliente
RSVP Inicializado...
-->Sessão RSVP criada
Path Message Arrived
-> senderNo=1 [139.82.95.15:0]
-> tSpec=[r=16000.0;b=2.0;p=16000.0;m=128;M=1500]
-> adSpec=[hopCount=0;pathBW=0.0;minLatency=0.0;composedMTU=0]
--> Mensagem PATH recebida: Path Message Arrived
-> senderNo=1 [139.82.95.15:0]
-> tSpec=[r=16000.0;b=2.0;p=16000.0;m=128;M=1500]
-> adSpec=[hopCount=0;pathBW=0.0;minLatency=30 composedMTU=0]
Serviço Requisitado
-> TransmissorNo=1 [139.82.95.15; SC=Guaranteed; Sytle=Fixed]
-> tSpec=[r=16000.0;b=2.0;p=16000.0;m=128;M=1500]

```

Figura 4.4.B – Sessão RSVP Cliente enviando a mensagem RESV calculada

É importante ressaltar que o retardo total solicitado pelo cliente hiperfídia para a transmissão do áudio foi de 60 ms. Deste valor, (3 ms) foram reservados pelo próprio cliente, (2 ms) reservados pelo protocolo hiperfídia cliente e (30 ms) solicitados à infraestrutura de comunicação. A diferença restante, 25 ms, deverá ser reservada pelos fornecedores de serviço (protocolo hiperfídia servidor e o servidor).

O protocolo hiperfídia cliente irá codificar os parâmetros da primitiva *HYP_Reserved.request* (*session_key*, *entity_id*, *serviceQoS*) em uma mensagem *Reserved.xml*.

Passo5: Enviando Mensagem XML ao Servidor

Utilizando qualquer protocolo de transporte, o cliente envia a mensagem XML.

Continuando o Exemplo02, o protocolo de transporte utilizado pelo cliente envia a mensagem *Reserved.xml* (ver Figura 4.4.C).

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE Reserve SYSTEM "HypProtocol.dtd">
<Reserve>
  <session_key>abc@tradicao1054</session_key>
  <entity_id>sound.wav</entity_id>
  <QoS>
    <serviceCategory>AudioGuaranteed</ServiceCategory>
    <audio>
      <maxDelay>25ms</maxDelay>
      <sampleRate>8 Khz</sampleRate>
      <sampleSize>8 bits</sampleSize>
      <loudness>Stereo</loudness>
      <audioCodification>PCM</audioCodification>
    </audio>
  </QoS>
</Reserve>
```

Figura 4.4.C – Mensagem Reserve.xml gerada pelo protocolo hipermídia

As ações realizadas no Cliente estão ilustradas no Diagrama de Seqüência da Figura 4.4.D. A seguir detalhamos a seqüência de passos realizadas no Servidor.

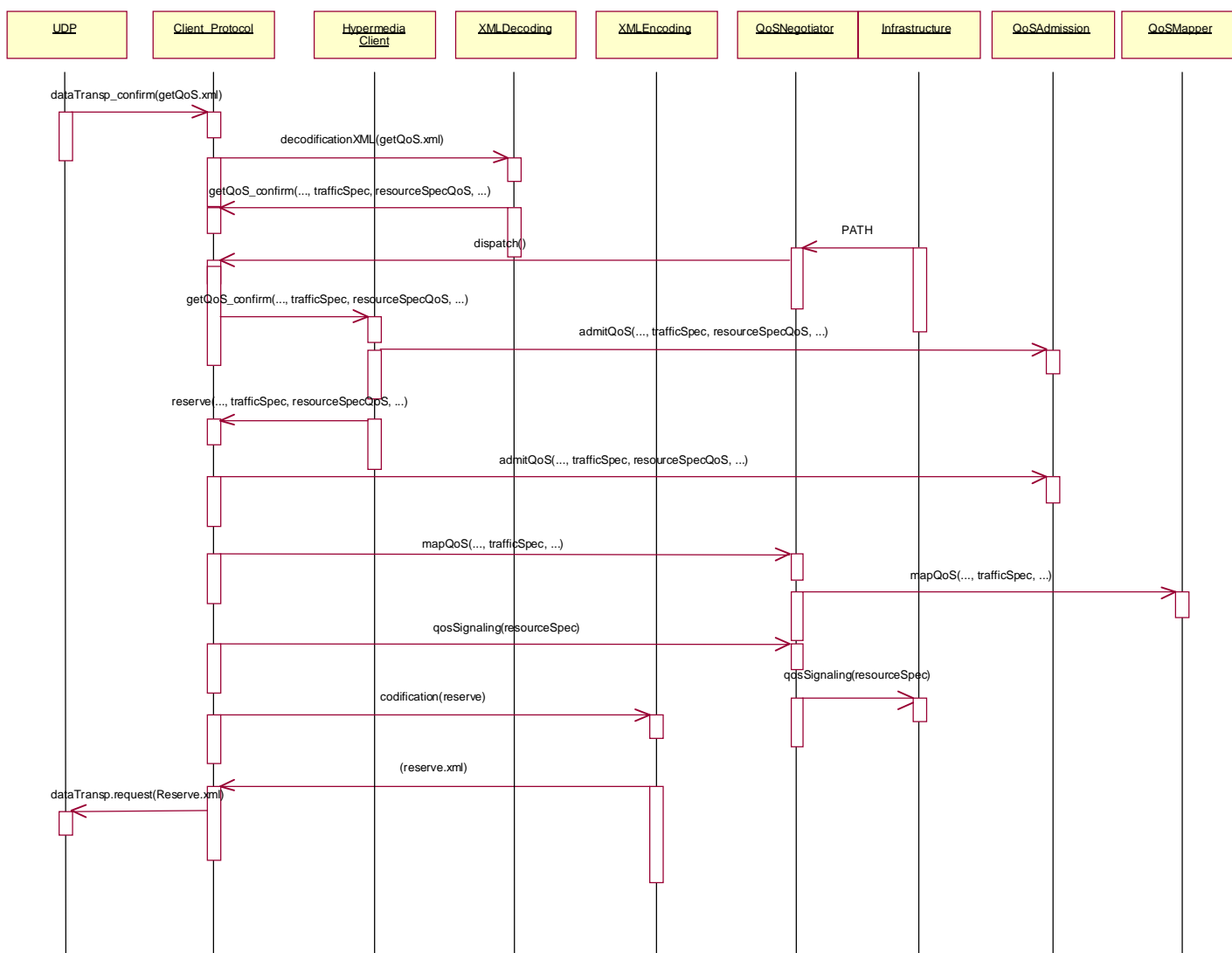


Figura 4.4.D – Diagrama de Sequência realizado pelo Cliente para criar um *MediaPipe*

Passos Realizados no Servidor:

Passo6: Recebimento da mensagem RESV

O servidor recebe uma indicação de ocorrência de evento “mensagem RESV recebida” confirmando a reserva de recursos na infra-estrutura de comunicação utilizada.

Seguindo o Exemplo02, o servidor (tradicao) recebe a primitiva `IPQoS.dispatch(ResvMessageArrived)`, indicando o recebimento da mensagem RESV. Ao receber esta mensagem, garante-se que o retardo de 30 ms solicitado foi reservado pela infra-estrutura de comunicação para a transmissão do áudio em tempo-real com a

categoria de serviço e os parâmetros de QoS especificados. A Figura 4.4.E ilustra o recebimento da mensagem RESV pelo servidor (*tradicao*).

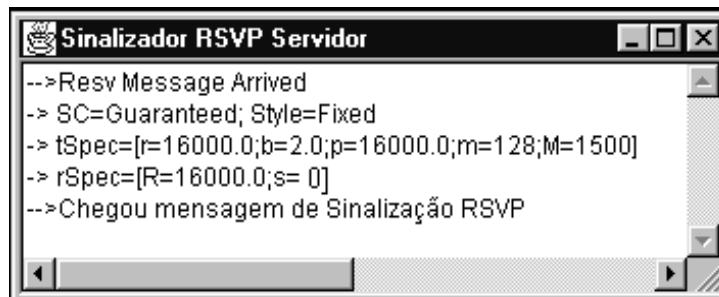


Figura 4.4.E – Sessão RSVP Servidor recebendo mensagem RESV

Passo7: Recebendo mensagem XML do Cliente

Fazendo uso de qualquer protocolo de transporte, o servidor recebe a mensagem XML.

Seguindo o Exemplo02, o protocolo de transporte utilizado pelo servidor recebe a mensagem Reserve.xml.

Passo8: Decodificando mensagem XML recebida

O protocolo hipermídia servidor recebe do protocolo de transporte a primitiva (*dataTransp.confirm(message)*), indicando o recebimento de uma mensagem XML, decodifica-a e gera outra primitiva de serviço.

Dando sequência ao Exemplo02, o protocolo hipermídia servidor recebe a primitiva dataTransp.confirm(reserve.xml), indicando o recebimento da mensagem Reserve.xml³⁸, decodifica-a e gera a primitiva de serviço HYP_Reserve.indication(abc@tradicao1054, sound.wav, (AudioGuaranteed, 25 ms (60 ms_{requisitado} – 35 ms_{reservado}), 8Khz, 8bits, Stereo, PCM).

³⁸ Esta mensagem identifica os requisitos de QoS que deverão ser atendidos durante a transmissão do áudio em tempo-real e qual a quantidade de recursos que deverão ser reservados pelos fornecedores de serviço (protocolo hipermídia servidor e o servidor - maxDelay – 25 ms).

Passo9: Realizando Controle de Admissão no Protocolo

O protocolo de comunicação hipermídia servidor recebe a primitiva *HYP_Reserve.indication* (*session_key*, *entity_id*, *serviceQoS*) e realiza os mecanismos de controle de admissão para verificar qual a quantidade de recursos que pode alocar para a criação do *MediaPipe* com os requisitos de QoS especificados (*serviceQoS*) pelo cliente hipermídia.

Dando continuidade ao Exemplo02, ao receber a primitiva HYP_Reserve.indication(abc@tradicao1054, sound.wav, (AudioGuaranteed, 25 ms, 8Khz, 8bits, Stereo, PCM), o protocolo hipermídia servidor realiza com sucesso os testes de controle de admissão, levando um retardo de (1 ms) para processar e transmitir o áudio em tempo-real, reserva então os recursos, e envia ao servidor a primitiva HYP_Reserve.indication(abc@tradicao1054, sound.wav, (AudioGuaranteed, 24 ms (60 ms_{requisitado} – 36 ms_{reservado}), 8Khz, 8bits, Stereo, PCM).

Passo10: Realizando Controle de Admissão no Servidor

O servidor, ao receber a primitiva de serviço primitiva *HYP_Reserve.indication* (*session_key*, *entity_id*, *serviceQoS*), irá realizar todos os quatro testes de controle de admissão para verificar se suporta os requisitos de QoS especificados pelo cliente. Se os quatro testes forem realizados com sucesso, o *MediaPipe* é criado e o servidor inicia a transmissão do documento com os requisitos de QoS especificados.

Continuando o Exemplo02, a aplicação servidor (tradicao) recebe a primitiva HYP_Reserve.indication(abc@tradicao1054, sound.wav, (AudioGuaranteed, 24 ms, 8Khz, 8bits, Stereo, PCM), realiza os testes de controle de admissão com sucesso, reserva o retardo de 4 ms para a transmissão do áudio solicitado e inicia a transmissão deste documento com os parâmetros de QoS especificados.

Passo11: Iniciando a Transmissão do Áudio Solicitado em tempo-real

O servidor utiliza o protocolo de transporte RTP para enviar os pacotes de áudio no *MediaPipe* criado. A escolha do RTP deve-se ao fato deste ser o protocolo de transporte padrão da Internet para o envio e recebimento de dados de mídia contínua, como áudio e

vídeo. Foi definido pelo grupo de trabalho AVT(*Audio Video Transport*) da IETF e está especificado na RFC 1889 [7].

As ações realizadas no Servidor estão ilustradas no Diagrama de Sequência da Figura 4.4.F. A seguir detalhamos a seqüência de passos realizadas no Cliente.

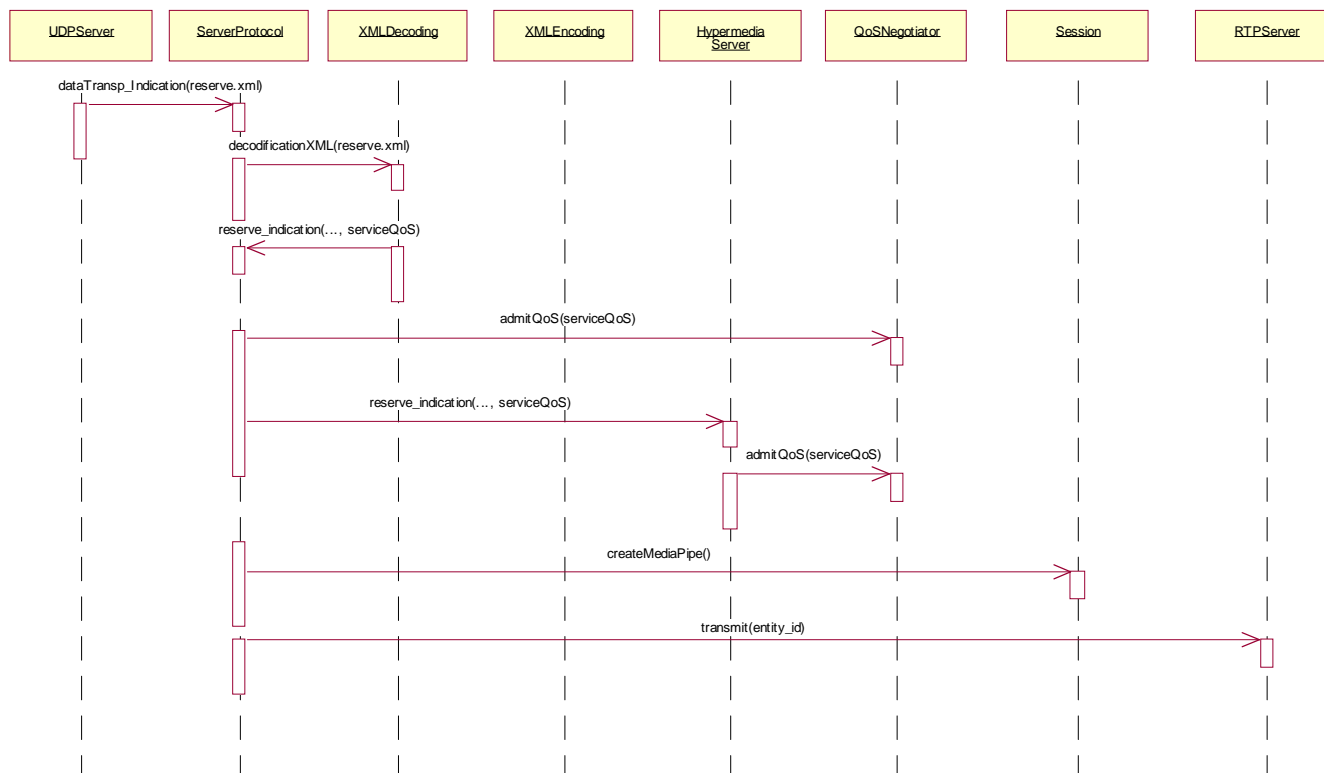


Figura 4.4.F – Diagrama de Sequência realizado pelo Servidor para criar um *MediaPipe*

Passos Realizados no Cliente:

Passo12: Recebimento da *Stream* de Áudio em tempo-real

O cliente começa a receber a *stream* de dados enviada pelo servidor.

O cliente hipermídia cria um player RTP para receber os pacotes de áudio enviados pelo servidor e apresentar a informação ao usuário final. A Figura 4.4.G ilustra o AOS com o MediaPipe áudio criado. A Figura 4.4.H ilustra o player RTP criado pelo cliente para receber o áudio (sound.wav).

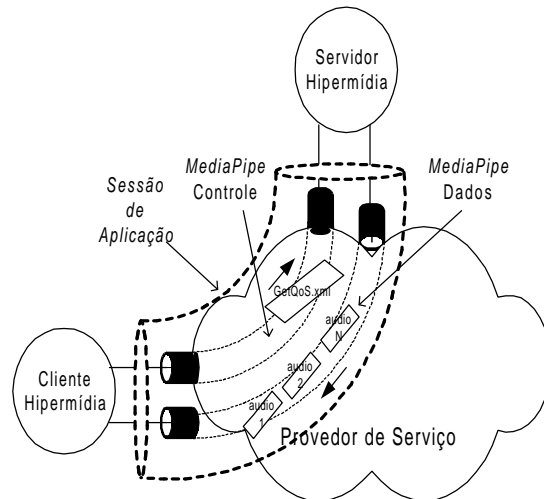


Figura 4.4.G – AOS com MediaPipe Controle e MediaPipe Dados

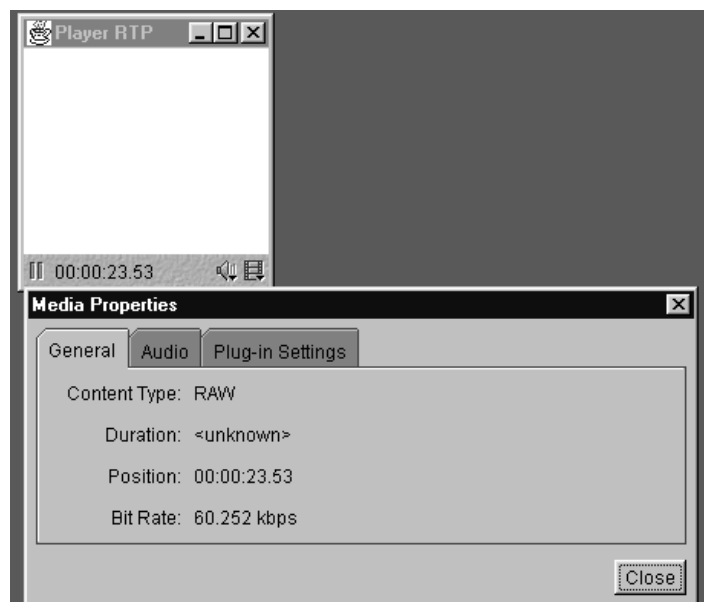


Figura 4.4.H – Player RTP para receber a transmissão do áudio (sound.wav) em tempo-real

O *player* RTP foi construído no protótipo utilizando-se as classes e interfaces definidas no pacote JMF 2.1[4].

Utilizando a expressão matemática apresentada na Seção 3.2.1, o *perfil de QoS da aplicação hiperídia* que solicita o áudio é representada pela seguinte expressão:

$$Q(a) = (q_{\text{controle}}, q_{\text{áudio}}),$$

onde:

$$Q_{\text{áudio}} = (r_{\text{memoria}}, r_{\text{cpu}}, r_{\text{banda_passante}}) \in \mathbf{R} = R_{\text{host}} \times R_{\text{roteadores}}$$

4.5. Cenário de Manutenção (Controle e Gerência) do Contrato de Serviço

Após a criação do *MediaPipe*, o provedor deve garantir que a categoria de serviço e a QoS negociada sejam mantidos durante todo o tempo de uso do *MediaPipe*. Dentre os mecanismos responsáveis pelo controle e gerência da QoS, estão incluídos os de (a) monitorização e a (b) ressintonização da QoS [12].

(a) Os mecanismos de monitorização permitem o registro da carga efetivamente gerada pelos usuários e da QoS realmente oferecida pelo provedor [12]. Em caso de violação de um contrato de serviço (SLA), estes mecanismos podem simplesmente ter seus registros repassados aos usuários de interesse sob a forma de alertas, utilizando-se para isto a primitiva de serviço *HYP_AlertQoS()* definida na API do protocolo, indicando que a QoS especificada para a transferência de um documento não está sendo oferecida pelo provedor de serviços.

(b) Os mecanismos de ressintonização da QoS são os responsáveis pela manutenção da orquestração dos recursos definida durante o estabelecimento do contrato sem que haja a necessidade de interrupção do serviço [12].

O protótipo desenvolvido implementa mecanismos monitoração, porém os parâmetros obtidos durante este processo, (1. número de pacotes perdidos, 2. número de pacotes fora de ordem e 3. número de pacotes recebidos), só retornam a percentagem de pacotes perdidos, não possuindo relação com o parâmetro retardo especificado no *MediaPipe*. Futuras atualizações no pacote JMF, deverão permitir a aferição de outros parâmetros de QoS que são diretamente aplicáveis, tais como: máximo retardo fim-a-fim, taxa máxima e taxa média recebida, permitindo a utilização de mecanismos de policiamento, para analisar se a QoS recebida está de acordo com a QoS especificada pela aplicação.

4.6. Considerações Finais

O presente capítulo apresentou exemplos de etapas que devem ser realizadas no protocolo de comunicação hipermídia para oferecer um serviço com garantias de qualidade. Para garantir os requisitos de qualidade solicitados, todos os elementos participantes da comunicação fim-a-fim deverão prover QoS. Dessa forma, além de utilizar o protocolo de comunicação hipermídia com QoS especificado neste trabalho, seria necessário fazer uso de: 1- aplicações do usuário (um *browser*, por exemplo) que possuísem funcionalidades de QoS, permitindo neste caso, a solicitação de ajustes nos parâmetros de qualidade recebidos durante a comunicação; 2- sistemas operacionais que oferecessem QoS, para que fosse possível a alocação, de forma eficiente, de todos os recursos necessários na estação do usuário (memória, tempo de processamento e recursos de comunicação) para a comunicação com QoS; e 3- provedores de comunicação que implementassem algum dos modelos de serviços com QoS propostos para a Internet (serviços integrados e diferenciados).

5

CONCLUSÕES & TRABALHOS FUTUROS

Este Capítulo realiza uma análise comparativa entre os protocolos de comunicação e as interfaces com QoS apresentadas no Capítulo 2 e o protocolo de comunicação hipermídia proposto. Em seguida, apresenta as contribuições da dissertação e as propostas para trabalhos futuros.

5.1- Comparação com Trabalhos Relacionados

A comparação entre os protocolos baseia-se em algumas características consideradas fundamentais na especificação de um protocolo de comunicação hipermídia, tais como: 1- QoS; 2- modularidade funcional; 3- independência de protocolos de transporte e de reserva de recursos; e 4- funcionalidades de trabalho cooperativo.

5.1.1- Comparação das Características dos Protocolos de Comunicação

- *QoS*

Os protocolos analisados no Capítulo 2 não foram projetados para oferecer um serviço com garantias de qualidade. Nas suas especificações, não existe a definição de como negociar os parâmetros de qualidade desejados, de como mapear estes parâmetros entre os diversos subsistemas envolvidos e de como monitorar os parâmetros de QoS negociados. Contudo, é possível utilizar um protocolo de reserva de recursos, como o RSVP, e solicitar o transporte de um documento com garantias de QoS, sendo necessário neste caso, ter conhecimento dos parâmetros nível de transporte especificados pelo RSVP. Percebe-se que nestes protocolos, mesmo utilizando o RSVP, não existe a definição de QoS de alto-nível.

No padrão H.323 v4, é possível que terminais H.323 solicitem o transporte de um documento com garantias de qualidade. Para que isto seja possível, utiliza-se o protocolo

H.245 em associação com o RSVP. A mensagem Open Logical Channel do H.245 possui o campo *qOSCapability* que determina quais os parâmetros de QoS desejados para o estabelecimento da conexão. A única desvantagem observada é a necessidade dos terminais H.323 terem conhecimento dos parâmetros de QoS nível de transporte especificados pelo protocolo RSVP.

O protocolo de comunicação hipermídia deste trabalho oferece tratamento de QoS, permitindo que as aplicações possam solicitar serviços com garantias de qualidade. Neste protocolo, o transporte de informações só é possível após a criação de uma Sessão de Aplicação que possua *MediaPipes* com os requisitos de qualidade solicitados. Para ser possível oferecer funcionalidades de *MediaPipes*, o protocolo especificado instanciou o *framework* para provisão de QoS, desenvolvido em [14], e implementou mecanismos de parametrização, negociação e sintonização de QoS, e mecanismos de alocação de recursos, fazendo uso da API IPQoS, desenvolvido em [31].

- ***Modularidade Funcional***

Os protocolos de comunicação HTTP, WEBDAV, RTSP e SIP não possuem o conceito de unidade funcional, não sendo possível a aplicação WWW, por exemplo um *browser*, escolher quais funcionalidades do protocolo deseja utilizar. Neste caso, toda API do protocolo deverá ser implementada pelas aplicações.

Os protocolos analisados foram projetados para dar suporte a um determinado tipo de serviço. O HTTP permite a troca de informações entre as aplicações cliente e servidor WWW de maneira simples e eficiente. Devido a constante evolução dos serviços oferecidos na WWW, várias extensões foram propostas ao HTTP, porém sem oferecer os conceitos de unidade funcional e composição de unidades funcionais.

O WEBDAV é um protocolo que estende o funcionamento do HTTP, definindo novos métodos para trabalho cooperativo. Estes novos métodos poderiam ter sido definidos como sendo uma nova unidade funcional do HTTP, caso o protocolo possuísse este conceito.

O RTSP é o protocolo utilizado para controlar o envio de mídias contínuas no tempo na Internet. Possui funcionamento semelhante ao HTTP, sendo que define novos métodos que poderiam estar definidos em uma unidade funcional do HTTP.

O SIP é o protocolo de controle utilizado para criar sessões multimídia na Internet. Pode ser utilizado com qualquer outro protocolo de comunicação. Define métodos para inicializar e finalizar chamadas, permitindo que qualquer participante de uma sessão possa convidar outros participantes a entrar ou sair de uma sessão multimídia.

A principal desvantagem observada é que uma aplicação WWW, por exemplo um *browser*, para poder explorar as principais funcionalidades oferecidas em um ambiente hipermídia, tais como: troca de hiper-documentos, transmissão de áudio e vídeo em tempo-real, autoria cooperativa em documentos, criação de sessões multimídia, entre outros, precisa utilizar todos os protocolos acima citados. O problema observado é que estes protocolos não foram desenvolvidos de maneira integrada, sendo necessário implementá-los ou utilizar aplicativos que os implementam, aumentando a complexidade na construção da aplicação. *Por exemplo, para receber um áudio em tempo-real na WWW, o browser deve possuir o plug-in de um aplicativo que implemente o protocolo RTSP, como por exemplo, o RealPlayer G2 .*

O protocolo WSP define dois tipos de serviços: orientado à conexão e não-orientado à conexão. Para cada tipo de serviço existe uma API específica. Desta forma, pode-se analisar o protocolo WSP como sendo composto por duas unidades funcionais. A limitação observada reside no fato de que a aplicação WAP só pode utilizar um tipo de serviço por vez, não sendo possível combinar o uso das duas unidades funcionais, o que não faria sentido no caso do WSP, visto que os dois tipos de serviços oferecidos são excludentes.

Os protocolos definidos no H.323 não possuem o conceito de modularidade funcional. Neste caso, quem desejar utilizar este padrão deverá implementar todas as funcionalidades oferecidas pelos protocolos de comunicação utilizados, aumentando a complexidade no uso deste padrão, ou implementar somente sub-conjuntos H.323,

possibilitando o desenvolvimento de diversas implementações do padrão que são incompatíveis entre si.

O protocolo de comunicação hipermídia especificado define quatro unidades funcionais: genérico, controle de acesso, conteúdo e QoS. Cada unidade funcional especifica um conjunto de primitivas de serviço, oferecendo as mais diversas funcionalidades. No protocolo, foi utilizada uma especificação orientada a objetos que permite utilizar as mais variadas combinações de unidades funcionais.

- ***Independência de Protocolos de Transporte e de Reserva de Recursos***

Todos os protocolos de comunicação analisados nesta seção são, a princípio, independentes dos protocolos de transporte e reserva de recursos utilizados. O problema observado é que a implementação destes protocolos de comunicação utiliza protocolos de transporte específicos e que não tem relação com os requisitos de qualidade das informações a serem transportadas.

As diversas implementações do HTTP utilizam o protocolo de transporte TCP para transportar dados multimídia. O WEBDAV, sendo uma extensão do HTTP, também utiliza o TCP para o transporte das informações. Conforme foi descrito na Seção 1.3, o TCP não é um protocolo adequado para transportar mídias contínuas no tempo. Esta é uma das principais limitações das diversas implementações do HTTP.

O RTSP pode utilizar dois protocolos de transporte (UDP e TCP) para enviar mensagens e um protocolo (RTP) para transportar as informações. O protocolo SIP pode utilizar qualquer protocolo de transporte, sendo que na sua especificação, são utilizados os protocolos UDP e TCP.

O WSP pode utilizar dois protocolos de transporte: UDP e WDP (*Wireless Datagram Protocol*). O uso dos mesmos depende do tipo de serviço (orientado ou não à conexão) escolhido pela aplicação WAP.

O H.323 define um Modelo de Referência que utiliza quatro protocolos de transporte (UDP, TCP, RTP, T.120) e três protocolos de sinalização e reserva de recursos (H.225 RAS, H.225 e H.245). Existem diversos estudos em andamento (*Annex C - H.323 on ATM* e *APPENDIX II - Transport Level Resource Reservation Procedures*) para adicionar no seu Modelo de Referência o protocolo de transporte ATM (AAL/5) e o de reserva de recursos (RSVP), respectivamente.

O protocolo de comunicação hipermídia especificado utiliza uma camada de adaptação (ver Seção 3.1) que é responsável por definir o protocolo de transporte mais adequado para o tipo de informação a ser transportada. Os dois arquivos de configuração utilizados (ver Seção 4.2), *transport.ini* e *mediaPipeTransport.ini*, indicam que o protocolo de comunicação hipermídia deve utilizar: o RTP para transportar mídias contínuas no tempo (áudio e vídeo); o TCP para transportar documentos com texto e imagem; o UDP para transportar mensagens do protocolo; o RSVP para reservar todos os recursos necessários para o transporte de dados com garantias de QoS; e o RTCP para monitorar a qualidade da transmissão.

É importante ressaltar que devido a existência de uma camada de adaptação no Modelo de Referência do protocolo, facilmente novos protocolos de transporte e reserva de recursos podem ser adicionados no mesmo. Por não definir uma camada de adaptação, o Modelo de Referência especificado pelo padrão H.323 não possui esta flexibilidade, sendo necessário desenvolver novos estudos, criar anexos/apêndices, para ser possível adicionar novos protocolos a este padrão.

- ***Trabalho Cooperativo***

De todos os protocolos analisados, somente o WEBDAV define mensagens para trabalho cooperativo na WWW.

O protocolo de comunicação especificado define no Capítulo 3, Seção 3.2.2, uma unidade funcional para trabalho cooperativo (controle de acesso). Desta forma, este protocolo pode ser utilizado por aplicações de autoria cooperativa no ambiente Internet.

5.1.2- API com QoS

No caso de oferecimento de serviços com QoS, as principais características que devem ser observadas são: abstrações definidas na API e primitivas de serviço.

- ***Abstrações Definidas na API***

Para oferecer simplicidade de uso, a API com QoS especificada no protocolo de comunicação hipermídia e as demais interfaces de comunicação com QoS definem abstrações do tipo: *sessão de aplicação*, *MediaPipe*, *sessão*, *fluxo*, *sessão de cooperação*, *sessão de transferência de informações*, *sessão de transporte* e *sessão de sinalização*.

A API com QoS do *middleware* Da CaPo++ define duas abstrações: *fluxo* e *sessão*. O conceito de *fluxo* representa uma *stream* de dados ponto-a-ponto (*unicast*) ou ponto-a-multiponto (*multicast*) que possui parâmetros de qualidade e categoria de serviço associados. Todo *fluxo* pode utilizar três protocolos de transporte: UDP, TCP e AAL/5 ATM. Além disso, o *fluxo* no *middleware* Da CaPo++ pode agregar novas funcionalidades de comunicação. *Por exemplo, pode utilizar um módulo de encriptação (DES, RC4, etc.) para transmitir as informações com segurança.* O conceito de *sessão* é utilizado para agrupar e gerenciar diversos fluxos.

A API com QoS para Aplicações de Trabalho Cooperativo (API com QoS para CSCW) define quatro abstrações: *sessão de cooperação*, *sessão de transferência de informações*, *sessão de transporte* e *sessão de sinalização*. O conceito de *sessão de cooperação* é idêntico ao conceito de *sessão* apresentado no Da CaPo++. O conceito de *sessão de transferência de informações* é semelhante ao conceito de *fluxo* Da CaPo++, porém com algumas particularidades: não utiliza diversos protocolos de transporte, somente o RTP; e não utiliza um conjunto de módulos com diversas funcionalidades. A *sessão de transporte* possui o mesmo conceito que a *sessão de transferência*, exceto pelo fato de ser um conceito do nível de sistemas de comunicação, específico para o protocolo de transporte RTP, enquanto que a *sessão de transferência* é um conceito do nível de aplicação, independente dos protocolos de transporte utilizados. A *sessão de sinalização* utiliza o protocolo RSVP para reservar todos os recursos necessários para a comunicação.

A API com QoS especificada no protocolo de comunicação hipermídia desenvolvido define duas abstrações: *MediaPipe* e *Sessão de Aplicação*. *MediaPipes* são criados pelo protocolo hipermídia quando a aplicação do usuário solicita o transporte de documentos. Todo *MediaPipe* possui um requisito de QoS associado. Caso a aplicação não especifique os parâmetros de qualidade desejados, a QoS *best-effort* é utilizada. O conceito de *MediaPipe* é semelhante aos conceitos de *fluxo* e *sessão de transferência de informações*, acima apresentados. Entretanto, o uso do termo *MediaPipe* indica que é um canal de comunicação específico para um determinado tipo de mídia, diferentemente dos conceitos de *sessão de transferência de informações* e *fluxo*.

Para facilitar o gerenciamento de diversos *MediaPipes*, utilizou-se o conceito de *Sessão de Aplicação*. O conceito de *Sessão de Aplicação* definido no protocolo é semelhante ao conceito de *sessão* do Da CaPo++ e *sessão de cooperação* do CSCW. A principal diferença é que a *Sessão* definida no protocolo de comunicação hipermídia só foi implementada ponto-a-ponto (*unicast*), enquanto que o conceito de *sessão* Da CaPo++ foi implementado ponto-a-multiponto (*multicast*).

- ***Primitivas de Serviço***

Os trabalhos analisados nesta seção definiram diversas primitivas de serviço para poder oferecer as mais diversas funcionalidades de comunicação.

Na API com QoS do Da CaPo++ existem algumas inconsistências na sintaxe das primitivas definidas: 1- a utilização dos prefixos Get / Recv e Set / Send nas primitivas de serviço. O ideal seria a utilização de uma única notação para enviar e receber informações, evitando qualquer tipo de confusão; e 2- a não especificação de primitiva para alertar qualquer violação no contrato de serviço estabelecido entre o cliente e o servidor.

Na API com QoS CSCW as *sessões de transferência de informações* são criadas quando a aplicação do usuário desejar enviar ou receber dados especificando os parâmetros de qualidade utilizados. A principal limitação observada é que não especifica primitiva para indicar qualquer violação no contrato de serviço estabelecido entre o cliente e o servidor.

Na API com QoS do protocolo de comunicação hipermídia, foram definidas primitivas para solicitar o transporte de documentos com garantias de QoS, criar *MediaPipes* com os parâmetros de QoS especificados, renegociar os parâmetros de QoS inicialmente especificados em um *MediaPipe* e indicar qualquer violação no contrato de serviço estabelecido entre o cliente e o servidor.

Na API especificada, a aplicação do usuário não fica responsável por solicitar a destruição do *MediaPipe*, como ocorre no Da CaPo++, ou seja, isto é uma tarefa interna do protocolo, reduzindo a complexidade de uso do mesmo pelas aplicações. Também não define primitivas para enviar dados de controle, como no Da CaPo++, pois possui um *MediaPipe* (controle) específico para isto.

Após o estudo e comparação de alguns protocolos e interfaces de comunicação com QoS, percebeu-se que o protocolo de comunicação hipermídia é bastante abrangente, oferecendo suporte adequado para o transporte de documentos hipermídia com garantias de QoS.

5.2- Contribuições da Dissertação

A contribuição e objetivo desta dissertação foi a especificação de um protocolo de comunicação hipermídia com QoS e Modularidade Funcional, que tem como características: 1- oferecer diversas unidades funcionais, permitindo o seu uso de forma individual ou combinada, reduzindo a complexidade de implementação do protocolo por parte da aplicação do usuário; 2- ser conceitualmente independente de protocolo de transporte e reserva de recursos, podendo ser adaptado para qualquer infra-estrutura de comunicação; e 3- utilizar a linguagem XML para codificar todas as mensagens do protocolo.

O novo protocolo especificado foi testado através de um protótipo [53] implementado no Laboratório Telemídia da Puc-Rio. Essa implementação utiliza: 1- os sistemas de *software* (*ServiceCategory* e *Quality*), desenvolvidos a partir das instanciações realizadas no *framework* de provisão de QoS [14], o que constitui outra contribuição desta dissertação; e 2- as classes e interfaces Java definidas no pacote JDK (*Java Development*

Kit), versão 1.2.2 [40]. Além do uso do JDK, foi necessária a utilização de outros três pacotes Java: a- JMF (*Java Media Framework*) [4], API utilizada para incorporar mídias contínuas no tempo em aplicações Java; b- XML4J (*IBM's XML Parser for Java*) [30], processador XML escrito em Java, utilizado para gerar documentos XML válidos e bem-formatados [26, 27]; e c- IPQoS [54], interface em Java para solicitação de serviços com QoS na Internet.

5.3- Trabalhos Futuros

Em decorrência do estudo feito nesta dissertação, observou-se que alguns aspectos devem ser considerados nas futuras atualizações deste protocolo:

- 1- **Instanciar e implementar o *framework* de escalonamento de recursos.** Apesar de não ter sido um questão de principal importância durante o desenvolvimento do protocolo, instanciar este *framework* significa alocar, de maneira eficiente, todos os recursos utilizados pelo sistema operacional para transportar as informações multimídia com garantias de qualidade.
- 2- **Especificar e implementar os conceitos de Sessão de Aplicação e *MediaPipes* do protocolo para oferecer comunicação *multicast*.** O protocolo de comunicação especificou e implementou o conceito de Sessão de Aplicação e *MediaPipes* para uma comunicação ponto-a-ponto (*unicast*). As futuras atualizações deste protocolo devem oferecer estes conceitos para uma comunicação ponto-a-multiponto (*multicast*), permitindo que estações do usuário possam ser dinamicamente adicionados ou removidos de um *MediaPipe*.
- 3- **Adaptar o funcionamento do protocolo para outros sistemas de transporte, como por exemplo as redes ATM. O Modelo de Referência do protocolo utiliza somente protocolos da arquitetura Internet.** Possíveis extensões a este modelo de referência irá permitir o uso dos protocolos de transporte e sinalização utilizados nas redes ATM. Neste caso, deverão ser utilizados novos mecanismos de mapeamento de parâmetros.

- 4- **Especificar e implementar mecanismos de segurança para a troca de informações no protocolo.** O único mecanismo de segurança implementado no protocolo é a autenticação dos usuários durante a criação de uma Sessão de Aplicação. Entretanto, as informações transportadas não utilizam nenhum algoritmo de segurança, como criptografia. Neste caso, futuras atualizações do protocolo deverão considerar privacidade e autenticidade como sendo parâmetros de QoS, permitindo oferecer à aplicação do usuário o uso de mecanismos de segurança durante o transporte de informações.

- 5- **Implementar a Unidade Funcional Controle de Acesso.** O protocolo de comunicação hipermídia especificou o conjunto de primitivas para realizar operações de trabalho cooperativo em um sistema hipermídia. Contudo, este serviço de controle de acesso não foi implementado no protocolo.

- 6- **Especificar e Implementar a Unidade Funcional Controle de Versão.** O protocolo de comunicação hipermídia não define uma unidade funcional para oferecer um serviço de controle de versão em documentos hipermídia. As futuras atualizações deste protocolo devem especificar o conjunto de primitivas para realizar operações de versão e implementar este serviço.

Apêndice A - API do Protocolo de Comunicação Hiperfídia

1. Notação

A notação utilizada para representar todas as primitivas de serviço do protocolo está descrita a seguir:

HYP_Nome_da_Primitiva (argumento1, argumento2, ..., argumentoN)

Todas as primitivas de serviço estão organizadas em unidades funcionais. Nas próximas seções, serão detalhadas as primitivas utilizadas na API cliente e na API servidor.

2. API do Protocolo de Comunicação Hiperfídia Cliente

2.1. Primitivas de Serviço (Unidade Funcional Genérico)

Unidade Funcional	Primitivas de Serviço
Genérico (<i>Kernel</i>)	<ul style="list-style-type: none"> - Abertura de uma Sessão de Aplicação (a) - Fechamento de uma Sessão de Aplicação (b) - Consulta ao status da Sessão de Aplicação (c) - Criação de Entidade (qualquer objeto) (d) - Criação de propriedades de uma Entidade (e) - Consulta as propriedades de uma determinada Entidade (f) (g) (h)

a) HYP_OpenSession

Descrição: Utilizado para abrir uma Sessão de Aplicação entre um cliente hipermídia em um servidor hipermídia.

```
HYP_OpenSession.request (
    String      user_name,
    String      password,
    String      server_id,
    String      profile
);
HYP_OpenSession.confirm (
    String      session_key
);
```

user_name Identifica o login do usuário;
password Identifica a senha do usuário;
server_id Identifica o nome do servidor hipermídia;
profile Identifica o perfil funcional desejado pelo usuário;
session_key Identifica a chave de autenticação do usuário com um determinado servidor;

b) HYP_CloseSession

Descrição: Utilizado para fechar uma Sessão de Aplicação entre um cliente hipermídia em um servidor hipermídia.

```
HYP_CloseSession.request (
    String      session_key,
);
HYP_CloseSession.confirm (
    String      session_key,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

c) HYP_SessionStatus

Descrição: Utilizado para mostrar o status de uma determinada sessão de aplicação entre um cliente hiperfície em um servidor hiperfície.

```
HYP_SessionStatus.request (
    String    session_key,
);
session_key  Identifica a chave de autenticação do usuário com um
              determinado servidor;
```

d) HYP_CreateEntity

Descrição: Utilizado para criar uma nova entidade.

```
HYP_CreateEntity.request (
    String    session_key,
    Object    entity_description,
    String    entity_id
);
```

```
HYP_CreateEntity.confirm (
    String    session_key,
    String    mediaPipeId,
);
```

```
session_key  Identifica a chave de autenticação do usuário com um
              determinado servidor;
mediaPipeId  Identifica o MediaPipe utilizado;
entity_description  Nova entidade criada.
entity_id    Identifica a nova entidade criada.
```

e) HYP_CreateEntityProperty

Descrição: Utilizado para criar uma propriedade de uma determinada entidade.

```
HYP_CreateEntityProperty.request (
    String      session_key,
    String      entity_id,
    Object      property_label,
    Object      property_value
);
```

```
HYP_CreateEntityProperty.confirm (
    String      session_key,
    String      mediaPipeId,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipeId Identifica o MediaPipe utilizado;

entity_id Identifica a entidade da qual se deseja criar uma determinada propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja modificar;

property_value Identifica o novo valor de uma determinada propriedade de um nó.

f) HYP_GetEntityProperties

Descrição: Utilizado para consultar as propriedades de uma determinada entidade (qualquer objeto).

```
HYP_GetEntityProperties.request (
    String      session_key,
    String      entity_id,
    String      level,
);
```

```
HYP_GetEntityProperties.confirm (
    String      session_key,
    String      mediaPipeId,
```

```

        Object      entity_properties,
        Object      entity_type
    );

```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;
mediaPipeId Identifica o MediaPipe utilizado;
entity_id Identifica a entidade da qual se deseja obter as propriedades;
level Identifica o nível da pesquisa a ser realizada. Pode assumir os valores:

- Nível 0: retorna apenas os atributos da entidade, sem conteúdo algum, mesmo se for nó de composição. Não traz estrutura;
- Nível n: retorna tudo recursivamente até o nível n-1 e apenas as descrições do nível n; - Nível -1: Retorna tudo, toda a estrutura recursiva de uma entidade;

entity_properties Identifica as propriedades de uma entidade a serem retornadas pelo servidor;
entity_type Identifica o tipo da entidade (nó terminal, nó composição, elo, descritor, etc).

g) HYP_GetEntityProperty

Descrição: Utilizado para consultar uma determinada propriedade de uma entidade.

```

HYP_GetEntityProperty.request (
    String      session_key,
    String      entity_id,
    Object      property_label,
);

```

```

HYP_GetEntityProperty.confirm (
    String      session_key,
    Object      property_value
);

```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;
entity_id Identifica a entidade da qual se deseja obter a propriedade;
property_label Identifica o tipo de propriedade de um nó que o cliente deseja consultar;

property_value Identifica o valor de uma determinada propriedade de um nó.

h) HYP_SetEntityProperty

Descrição: Utilizado para modificar a propriedade de uma determinada entidade.

```
HYP_SetEntityProperty.request (
    String      session_key,
    String      entity_id,
    Object      property_label,
    Object      property_value
);
```

```
HYP_SetEntityProperty.confirm (
    String      session_key,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja modificar uma determinada propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja modificar;

property_value Identifica o novo valor de uma determinada propriedade de um nó.

i) HYP_Error

Descrição: Indica a ocorrência de algum erro.

```
HYP_Error.confirm (
    String      message
);
message      Mensagem de erro.
```

2.2. Primitivas de Serviço (Unidade Funcional Controle de Acesso)

Unidade Funcional	Primitivas de Comunicação
Controle de Acesso (Access Control)	- Controle de acesso as propriedades de uma determinada Entidade (a) (b) (c) (d) (e) (f) (g)

a) LockEntityProperty

Descrição: Utilizado para controlar o acesso às propriedades de uma determinada Entidade. Um cliente hipermídia só pode alterar as propriedades de uma determinada entidade se, inicialmente, "bloquear" o acesso às propriedades desta entidade. Isto é feito utilizando-se a primitiva **LockEntityProperty**.

```
HYP_LockEntityProperty.request (
    String      session_key,
    String      entity_id,
    Property    property_label,
    Principal   user_id
);
```

```
HYP_LockEntityProperty.confirm (
    String      session_key,
    Boolean     isLocked
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja restringir o acesso a alguma propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja restringir o acesso;

user_id Identifica o usuário que solicita lock;

IsLocked Identifica se a propriedade foi bloqueada.

b) UnLockEntityProperty

Descrição: Utilizado para "desbloquear" o acesso às propriedades de uma determinada entidade. Desta forma, qualquer outro cliente poderá ter acesso a uma determinada propriedade de um nó.

```
HYP_LockEntityProperty.request (
```

```

        String      session_key,
        String      entity_id,
        Property    property_label,
        Principal   user_id
    );

```

```

HYP_UnLockEntityProperty.confirm (
    String      session_key,
    Boolean     isLocked
);

```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja liberar o acesso a alguma propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja liberar o acesso.

c) IsLocked

Descrição: Utilizado para verificar se o acesso a uma propriedade de uma entidade está "bloqueado" ou "desbloqueado".

```

HYP_IsLocked.request (
    String      session_key,
    String      entity_id,
    Property    property_label,
);

```

```

HYP_IsLocked.confirm (
    String      session_key,
    Boolean     isLocked
);

```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja consultar o tipo de acesso a suas propriedades;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja consultar o tipo acesso;

is_locket Identifica se a propriedade está bloqueada ou desbloqueada.

d) GetEntityPermission

Descrição: Utilizado para consultar o tipo de permissão de uma Entidade.

```
HYP_GetEntityPermission.request (
    String session_key,
    Entity entity,
    Principal subject
);
```

```
HYP_GetEntityPermission.confirm (
    String session_key,
    ObjectPermission permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity Identifica a Entidade;

subject Identifica o sujeito que quer realizar uma ação;

permission Identifica a permissão do sujeito sobre a Entidade.

e) SetEntityPermission

Descrição: Utilizado para modificar o tipo de permissão de uma Entidade.

```
HYP_SetEntityPermission.request (
    String session_key,
    Entity entity,
    Principal subject,
    ObjectPermission permission
);
```

```
HYP_GetEntityPermission.confirm (
    String session_key,
    ObjectPermission old_permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity Identifica a Entidade;

subject Identifica o sujeito que quer realizar uma ação;

permission Identifica a permissão do sujeito sobre a Entidade;

old_permission Identifica a antiga permissão de uma Entidade.

f) GetPropertyPermission

Descrição: Utilizado para consultar o tipo de permissão de uma Propriedade de uma Entidade.

```
HYP_GetPropertyPermission.request (
    String session_key,
    Entity entity,
    Property property,
    Principal subject
);
```

```
HYP_GetPropertyPermission.confirm (
    String session_key,
    ObjectPermission permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity Identifica a Entidade;

property Identifica a Propriedade de uma Entidade;

subject Identifica o sujeito que quer realizar uma ação;

permission Identifica a permissão do sujeito sobre a Entidade.

g) SetPropertyPermission

Descrição: Utilizado para modificar o tipo de permissão de uma Propriedade de uma Entidade.

```
HYP_SetEntityPermission.request (
    String session_key,
    Entity entity,
    Principal subject,
    Property property,
    ObjectPermission permission
);
```

```
HYP_SetEntityPermission.confirm (
    String session_key,
```

```

    ObjectPermission    old_permission
);

session_key    Identifica a chave de autenticação do usuário com um
               determinado servidor;

entity         Identifica a Entidade;

subject        Identifica o sujeito que quer realizar uma ação;

property       Identifica a Propriedade de uma Entidade;

permission     Identifica a permissão do sujeito sobre a Entidade;

old_permission Identifica a antiga permissão de uma Entidade.

```

2.3. Primitivas de Serviço (Unidade Funcional Conteúdo)

Unidade Funcional	Primitivas de Serviços
Conteúdo (<i>Content</i>)	<ul style="list-style-type: none"> - Solicitação para a transferência de dados (a) (c) - Finalização da transferência de dados (b) (d)

a) HYP_Get

Descrição: Utilizado para buscar um documento.

```

HYP_Get.request (
    String session_key,
    String entity_id
);

HYP_Get.confirm (
    String session_key,
    String mediaPipe_id
);

session_key    Identifica a chave de autenticação do usuário com um
               determinado servidor;

entity_id      Identifica a entidade da qual se deseja obter o conteúdo;

mediaPipe_id   Identifica o fluxo de transporte utilizado.

```

b) HYP_Pause

Descrição: Utilizado para finalizar, momentaneamente, a transmissão em um *MediaPipe*.

```
HYP_Pause.request (
```

```
    String session_key,
    String mediaPipe_id,
);
```

```
HYP_Pause.confirm (
    String session_key,
    Boolean isOK
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipe_id Identifica o fluxo utilizado para o transporte dos dados.

IsOK Identifica se a transmissão foi parada, momentaneamente, no *MediaPipe* identificado.

c) HYP_Resume

Descrição: Utilizado para reiniciar a transmissão em um *MediaPipe*.

```
HYP_Resume.request (
```

```
    String session_key,
    String mediaPipe_id,
);
```

```
HYP_Resume.confirm (
    String session_key,
    Boolean isOK
);
```

user_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipe_id Identifica o fluxo utilizado para o transporte dos dados.

IsOK Identifica se a transmissão foi reinicializada

d) HYP_Stop

Descrição: Utilizado para finalizar a transmissão em um *MediaPipe*.

```
HYP_Stop.request (
```

```
    String session_key,
    String mediaPipe_id,
);
```

```
HYP_Stop.confirm (
    String session_key,
```

```

        Boolean    isOK
    );

    session_key    Identifica a chave de autenticação do usuário com um
                    determinado servidor;

    mediaPipe_id   Identifica o fluxo utilizado para o transporte dos dados.

    IsOK           Identifica se a transmissão foi parada

```

2.4. Primitivas de Serviço (Unidade Funcional Qualidade de Serviço)

Unidade Funcional	Primitivas de Serviços
Qualidade de Serviço (<i>Quality of Service</i>)	<ul style="list-style-type: none"> - Negociação de QoS para o transporte de dados (a) - Criação do <i>MediaPipe</i> com a QoS especificada (b) - Renegociação de QoS (c) - Monitoração da QoS estabelecida (d)

a) HYP_GetQoS

Descrição: Utilizado para solicitar um documento com garantias de QoS.

```

HYP_GetQoS.request (
    String session_key,
    String entity_id
);

HYP_GetQoS.confirm (
    String session_key,
    String entity_id,
    String trafficSpec,
    String resourceSpec,
);

session_key    Identifica a chave de autenticação do usuário com um
                determinado servidor;

entity_id      Identifica a entidade da qual se deseja obter o conteúdo;

trafficSpec    Identifica quais são os parâmetros de descrição de tráfego.

resourceSpec   Identifica quais são os parâmetros de alocação de recursos.

```

b) HYP_Reserve

Descrição: Utilizado para solicitar a criação de um *MediaPipe* com a QoS especificada.

```
HYP_ReserverQoS.request (
    String session_key,
    String entity_id,
    ServiceCategory serviceQoS
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja obter o conteúdo;

serviceQoS Identifica a QoS especificada pelo usuário.

c) HYP_RenegotiateQoS

Descrição: Utilizado para renegociar os parâmetros de QoS definidos em um MediaPipe existente.

```
HYP_RenegotiateQoS.request (
    String session_key,
    String MediaPipe_id,
    ServiceCategory serviceQoS
);
```

```
HYP_RenegotiateQoS.confirm (
    String session_key,
    Boolean isRenegotiated,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

MediaPipe_id Identifica o *MediaPipe* que terá os requisitos de QoS modificados;

serviceQoS Identifica quais são os novos parâmetros de qualidade de serviço desejados pelo usuário.

IsRenegotiated Identifica se os requisitos de QoS de um MediaPipe foram renegociados.

d) HYP_AlertQoS

Descrição: Utilizado para indicar que a QoS desejada pelo usuário não está sendo oferecida pelo provedor de serviços.

```
HYP_AlertQoS.confirm(
    String session_key,
```

```

        String  MediaPipe_id,
        ServiceCategory  serviceQoS
    );

```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

MediaPipe_id Identifica MediaPipe cuja QoS solicitada não está sendo oferecida.

serviceQoS Identifica quais são os parâmetros de qualidade de serviço oferecidos pela provedor.

3. API do Protocolo de Comunicação Hiperfídia Servidor

3.1. Primitivas de Serviço (Unidade Funcional Genérico)

Unidade Funcional	Primitivas de Serviço
Genérico (<i>Kernel</i>)	<ul style="list-style-type: none"> - Abertura de uma Sessão de Aplicação (a) - Fechamento de uma Sessão de Aplicação (b) - Consulta ao status da Sessão de Aplicação (c) - Criação de Entidade (qualquer objeto) (d) - Criação de propriedades de uma Entidade (e) - Consulta as propriedades de uma determinada Entidade (f) (g) (h)

a) HYP_OpenSession

Descrição: Utilizado para abrir uma sessão de aplicação entre um cliente hiperfídia em um servidor hiperfídia.

```

HYP_OpenSession.indication (

```

```

    String  user_name,
    String  password,
    String  profile,
    String  client_id,
    String  port_client,
    String  session_key

```

```

);

```

```

HYP_OpenSession.response (

```

```

    String  session_key
    String  profile,
    String  client_id,
    String  port_client,

```

```

);

```

user_name	Identifica o login do usuário;
password	Identifica a senha do usuário;
client_id	Identifica o cliente hipermídia;
port_client	Identifica a porta do cliente hipermídia;
profile	Identifica o perfil funcional desejado pelo usuário;
session_key	Identifica a chave de autenticação do usuário com um determinado servidor;

b) HYP_CloseSession

Descrição: Utilizado para fechar uma sessão de aplicação entre um cliente hipermídia em um servidor hipermídia.

```
HYP_CloseSession.indication (
    String session_key,
);
```

```
HYP_CloseSession.response (
    String session_key,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

c) HYP_SessionStatus

Descrição: Utilizado para mostrar o status de uma determinada sessão de aplicação entre um cliente hipermídia em um servidor hipermídia.

```
HYP_SessionStatus.response (
    String session_key,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

d) HYP_CreateEntity

Descrição: Utilizado para criar uma nova entidade.

```
HYP_CreateEntity.indication (
```



```

        String      session_key,
        Object      entity_description,
        String      entity_id
    );

```

```

HYP_CreateEntity.response (
    String      session_key,
    String      mediaPipeId,
);

```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
MediaPipe_id	Identifica MediaPipe cuja QoS solicitada não está sendo oferecida.
entity_description	Nova entidade criada.
entity_id	Identifica a nova entidade criada.

e) HYP_CreateEntityProperty

Descrição: Utilizado para criar uma propriedade de uma determinada entidade.

```

HYP_CreateEntityProperty.indication (
    String      session_key,
    String      entity_id,
    Object      property_label,
    Object      property_value
);
HYP_CreateEntityProperty.response (
    String      session_key,
);

```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
entity_id	Identifica a entidade da qual se deseja criar uma determinada propriedade;
property_label	Identifica o tipo de propriedade de um nó que o cliente deseja modificar;
property_value	Identifica o novo valor de uma determinada propriedade de um nó.

f) HYP_GetEntityProperties

Descrição: Utilizado para consultar as propriedades de uma determinada entidade (qualquer objeto).

```
HYP_GetEntityProperties.indication (
    String      session_key,
    String      entity_id,
    String      level,
);
```

```
HYP_GetEntityProperties.response (
    String      session_key,
    String      mediaPipeId,
    Object      entity_properties,
    Object      entity_type
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

MediaPipe_id Identifica MediaPipe cuja QoS solicitada não está sendo oferecida.

entity_id Identifica a entidade da qual se deseja obter as propriedades;

level Identifica o nível da pesquisa a ser realizada. Pode assumir os valores:

- Nível 0: retorna apenas os atributos da entidade, sem conteúdo algum, mesmo se for nó de composição. Não traz estrutura;

- Nível n: retorna tudo recursivamente até o nível n-1 e apenas as descrições do nível n; - Nível -1: Retorna tudo, toda a estrutura recursiva de uma entidade;

entity_properties Identifica as propriedades de uma entidade a serem retornadas pelo servidor;

entity_type Identifica o tipo da entidade (nó terminal, nó composição, elo, descritor, etc).

g) HYP_GetEntityProperty

Descrição: Utilizado para consultar uma determinada propriedade de uma entidade.

```
HYP_GetEntityProperty.indication (
    String      session_key,
    String      entity_id,
    Object      property_label,
);
```

```
);
HYP_GetEntityProperty.response (
    String      session_key,
    Object      property_value
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja obter a propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja consultar;

property_value Identifica o valor de uma determinada propriedade de um nó.

h) HYP_SetEntityProperty

Descrição: Utilizado para modificar a propriedade de uma determinada entidade.

```
HYP_SetEntityProperty.indication (
    String      session_key,
    String      entity_id,
    Object      property_label,
    Object      property_value
);
```

```
HYP_SetEntityProperty.response (
    String      session_key,
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja modificar uma determinada propriedade;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja modificar;

property_value Identifica o novo valor de uma determinada propriedade de um nó.

i) HYP_Error

Descrição: Indica a ocorrência de algum erro.

```
HYP_Error.indication (
    String      message
);
HYP_Error.response (
    String      message,
);

client_id      Identifica o cliente hipermedia;
```

3.2. Primitivas de Serviço (Unidade Funcional Controle de Acesso)

Unidade Funcional	Primitivas de Comunicação
Controle de Acesso (<i>Access Control</i>)	- Controle de acesso as propriedades de uma determinada Entidade (a) (b) (c) (d) (e) (f) (g)

a) LockEntityProperty

Descrição: Utilizado para controlar o acesso às propriedades de uma determinada Entidade.

```
HYP_LockEntityProperty.indication (
    String      session_key,
    String      entity_id,
    Property    property_label,
    Principal   user_id
);

HYP_LockEntityProperty.response (
    String      session_key,
    Boolean     isLocked
);

session_key    Identifica a chave de autenticação do usuário com um
                determinado servidor;
```

entity_id	Identifica a entidade da qual se deseja restringir o acesso a alguma propriedade;
property_label	Identifica o tipo de propriedade de um nó que o cliente deseja restringir o acesso;
user_id	Identifica o usuário que solicita lock;
IsLocked	Identifica se a propriedade foi bloqueada.

b) UnlockEntityProperty

Descrição: Utilizado para "desbloquear" o acesso às propriedades de uma determinada entidade.

```
HYP_LockEntityProperty.indication (
    String      session_key,
    String      entity_id,
    Property    property_label,
    Principal   user_id
);
```

```
HYP_UnlockEntityProperty.response (
    String      session_key,
    Boolean     isLocked
);
```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
entity_id	Identifica a entidade da qual se deseja liberar o acesso a alguma propriedade;
property_label	Identifica o tipo de propriedade de um nó que o cliente deseja liberar o acesso.

c) IsLocked

Descrição: Utilizado para verificar se o acesso a uma propriedade de uma entidade está "bloqueado" ou "desbloqueado".

```
HYP_IsLocked.indication (
    String      session_key,
    String      entity_id,
    Property    property_label,
);
```

```
HYP_IsLocked.response (
    String      session_key,
    Boolean     isLocked
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja consultar o tipo de acesso a suas propriedades;

property_label Identifica o tipo de propriedade de um nó que o cliente deseja consultar o tipo acesso;

is_locket Identifica se a propriedade está bloqueada ou desbloqueada.

d) GetEntityPermission

Descrição: Utilizado para consultar o tipo de permissão de uma Entidade.

```
HYP_GetEntityPermission.indication (
    String      session_key,
    Entity      entity_id,
    Principal    subject
);
```

```
HYP_GetEntityPermission.response (
    String      session_key,
    ObjectPermission permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a Entidade;

subject Identifica o sujeito que quer realizar uma ação;

permission Identifica a permissão do sujeito sobre a Entidade.

e) SetEntityPermission

Descrição: Utilizado para modificar o tipo de permissão de uma Entidade.

```
HYP_SetEntityPermission.indication (
    String      session_key,
    Entity      entity_id,
    Principal    subject,
    ObjectPermission permission
);
```

```
HYP_SetEntityPermission.response (
    String      session_key,
    ObjectPermission old_permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a Entidade;

subject Identifica o sujeito que quer realizar uma ação;

permission Identifica a permissão do sujeito sobre a Entidade;

old_permission Identifica a antiga permissão de uma Entidade.

f) GetPropertyPermission

Descrição: Utilizado para consultar o tipo de permissão de uma Propriedade de uma Entidade.

```
HYP_GetPropertyPermission.indication (
    String      session_key,
    Entity      entity_id,
    Property    property,
    Principal    subject
);
```

```
HYP_GetPropertyPermission.response (
    String      session_key,
    ObjectPermission permission
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id	Identifica a Entidade;
property	Identifica a Propriedade de uma Entidade;
subject	Identifica o sujeito que quer realizar uma ação;
permission	Identifica a permissão do sujeito sobre a Entidade.

g) SetPropertyPermission

Descrição: Utilizado para modificar o tipo de permissão de uma Propriedade de uma Entidade.

```
HYP_SetEntityPermission.indication (
    String          session_key,
    Entity          entity_id,
    Principal       subject,
    Property        property,
    ObjectPermission permission
);
```

```
HYP_SetEntityPermission.response (
    String          session_key,
    ObjectPermission old_permission
);
```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
entity	Identifica a Entidade;
subject	Identifica o sujeito que quer realizar uma ação;
property	Identifica a Propriedade de uma Entidade;
permission	Identifica a permissão do sujeito sobre a Entidade;
old_permission	Identifica a antiga permissão de uma Entidade.

3.3. Primitivas de Serviço (Unidade Funcional Conteúdo)

Unidade Funcional	Primitivas de Serviços
Conteúdo (<i>Content</i>)	- Solicitação para a transferência de dados (a) (c) - Finalização da transferência de dados (b) (d)

a) HYP_Get

Descrição: Utilizado para buscar um documento.

```
HYP_Get.indication (
    String session_key,
    String entity_id
);

HYP_Get.response (
    String session_key,
    String mediaPipe_id
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja obter o conteúdo;

mediaPipe_id Identifica o fluxo de transporte utilizado.

b) HYP_Pause

Descrição: Utilizado para finalizar, momentaneamente, a transmissão em um *MediaPipe*.

```
HYP_Pause.indication (
    String session_key,
    String mediaPipe_id,
);

HYP_Pause.response (
    String session_key,
    Boolean isOK
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipe_id Identifica o fluxo utilizado para o transporte dos dados.

isOK Identifica se a transmissão foi parada, momentaneamente, no *MediaPipe* identificado.

c) HYP_Resume

Descrição: Utilizado para reiniciar a transmissão em um *MediaPipe*.

```
HYP_Resume.indication (
    String session_key,
    String mediaPipe_id,
);
```

```
HYP_Resume.response (
    String session_key,
    Boolean isOK
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipe_id Identifica o fluxo utilizado para o transporte dos dados.

IsResumed Identifica se a transmissão foi reinicializada

d) HYP_Stop

Descrição: Utilizado para finalizar a transmissão em um *MediaPipe*.

```
HYP_Stop.indication (
    String session_key,
    String mediaPipe_id,
);
```

```
HYP_Stop.response (
    String session_key,
    Boolean isOK
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

mediaPipe_id Identifica o fluxo utilizado para o transporte dos dados.

IsStoped Identifica se a transmissão foi parada

3.4. Primitivas de Serviço (Unidade Funcional Qualidade de Serviço)

Unidade Funcional	Primitivas de Serviços
Qualidade de Serviço	- Negociação de QoS para o transporte de dados (a)

(Quality of Service)	<ul style="list-style-type: none"> - Criação do <i>MediaPipe</i> com a QoS especificada (b) - Renegociação de QoS (c)
----------------------	---

a) HYP_GetQoS

Descrição: Utilizado para buscar um documento especificando os parâmetros de QoS e a categoria de serviço desejados.

```
HYP_GetQoS.indication (
```

```
    String session_key,
    String entity_id,
```

```
);
```

```
HYP_GetQoS.response (
```

```
    String session_key,
    String entity_id,
    String trafficSpec,
    String resourceSpec,
```

```
);
```

session_key Identifica a chave de autenticação do usuário com um determinado servidor;

entity_id Identifica a entidade da qual se deseja obter o conteúdo;

trafficSpec Identifica quais são os parâmetros de descrição de tráfego.

resourceSpec Identifica quais são os parâmetros de alocação de recursos.

b) HYP_Reserve

Descrição: Utilizado para solicitar a criação de um *MediaPipe* com a QoS especificada.

```
HYP_ReserverQoS.indication (
```

```
    String session_key,
    String entity_id,
    ServiceCategory serviceQoS
```

```
);
```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
entity_id	Identifica a entidade da qual se deseja obter o conteúdo;
serviceQoS	Identifica a QoS especificada pelo usuário.

c) HYP_RenegotiateQoS

Descrição: Utilizado para renegociar os parâmetros de QoS definidos em um MediaPipe existente.

```
HYP_RenegotiateQoS.indication (
    String session_key,
    String MediaPipe_id,
    ServiceCategory serviceQoS
);
```

```
HYP_RenegotiateQoS.response (
    String session_key,
    Boolean isRenegotiated,
);
```

session_key	Identifica a chave de autenticação do usuário com um determinado servidor;
MediaPipe_id	Identifica o <i>MediaPipe</i> que terá os requisitos de QoS modificados;
serviceQoS	Identifica quais são os novos parâmetros de qualidade de serviço desejados pelo usuário.
IsRenegotiated	Identifica se os requisitos de QoS de um MediaPipe foram renegociados.

Apêndice B - Primitivas da Interface da Camada de Adaptação

1. Primitivas da Interface da Camada de Adaptação (Cliente)

1.1. Primitivas para o Estabelecimento e Encerramento de Conexão

a) HYP_OpenConnexion

Descrição: Utilizado para estabelecer uma conexão com o servidor.

```
HYP_OpenConnexion.request (  
    String server,  
    Int port  
);
```

```
HYP_OpenConnexion.confirm (  
    Boolean isConnected,  
);
```

server Identifica o endereço da estação com o qual se deseja estabelecer conexão.

port Identifica a porta da estação.

IsConnected Identifica se a conexão foi criada corretamente.

b) HYP_CloseConnexion

Descrição: Utilizado para finalizar uma conexão com o servidor.

```
HYP_CloseConnexion.request (  
    String server,  
    Int port  
);
```

```
HYP_CloseConnexion.confirm (
    Boolean isDisConected,
);
```

server Identifica o endereço da estação com o qual se deseja finalizar conexão.

port Identifica a porta da estação.

IsConnected Identifica se a conexão foi finalizada corretamente

1.2. Primitiva para o Transferência de Dados

a) HYP_DataTransp

Descrição: Utilizado para enviar ou receber dados da camada de transporte.

```
HYP_DataTransp.request (
    Object data
    String server,
    Int port
);
```

```
HYP_DataTransp.confirm (
    Object data
    String server,
    Int port
);
```

data Identifica os dados enviados ou recebidos à camada de transporte.

server Identifica o endereço da estação com o qual se deseja estabelecer conexão.

port Identifica a porta da estação.

1.3. Primitivas para a Reserva de Recursos

a) HYP_CreateQoSNegotiator

Descrição: Utilizado para criar o negociador de QoS em redes IP.

```
HYP_CreateQoSNegotiator.request (
    String QoSNegotiator_type
    String negotiatorId
);
```

```
HYP_CreateQoSNegotiator.confirm (
```

```

        Boolean isQoSNegotiator
    );

    QoSNegotiator_type Indica o tipo de negociador de QoS.

    isQoSNegotiator Indica se o negociador de QoS foi criado corretamente.

    negotiatorId Identifica o negociador de QoS.

```

b) HYP_Dispatch

Descrição: Utilizado para receber mensagens de sinalização enviadas pelo servidor.

```

    HYP_Dispatch.confirm ( );

```

c) HYP_Request

Descrição: Utilizado para enviar mensagens de sinalização para o servidor.

```

    HYP_Request.request (

        String session
        String client
        FilterSpec filterSpecList
        TrafficSpec scList
    );

    Session Sessão de sinalização criada.

    Client Cliente que envia a solicitação

    FilterSpecList Identifica o tipo de filterSpec utilizado

    ScList Possui os valores de Tspec e RSpec

```

d) HYP_DestroyQoSNegotiator

Descrição: Utilizado para destruir o negociador de QoS em redes IP.

```

    HYP_DestroyQoSNegotiator.request (
        String negotiatorId
    );

    HYP_CreateQoSNegotiator.confirm (
        Boolean isDestroyed
    );

    negotiatorId Identifica o negociador de QoS.
    IsDestroyed Identifica se a sessão de sinalização foi destruída

```

1.4. Primitiva para Monitoração de QoS

a) HYP_ReceiverReport

Descrição: Utilizado para indicar ao servidor os parâmetros de QoS utilizados durante uma transmissão.

```
HYP_ReceiverReport.request (
    String      session_id,
    ServiceCategory  QoS
);
```

session_id Identifica a sessão de sinalização criada entre o cliente e o servidor hiperfórmula.

QoS Identifica quais são os parâmetros de qualidade de serviço, a nível de transporte, recebidos pelo cliente hiperfórmula.

2. Primitivas da Interface da Camada de Adaptação (Servidor)

2.1. Primitivas para o Estabelecimento e Encerramento de Conexão

a) HYP_OpenConnexion

Descrição: Utilizado para estabelecer uma conexão com o cliente.

```
HYP_OpenConnexion.indication (
    String  client,
    Int     port
);
HYP_OpenConnexion.response (
    Boolean isConnected,
);
```

client Identifica o endereço da estação com o qual se deseja estabelecer conexão.

port Identifica a porta da estação.

IsConnected Identifica se a conexão foi criada corretamente

b) HYP_CloseConnexion

Descrição: Utilizado para finalizar uma conexão com o cliente.

```
HYP_CloseConnexion.indication (
    String client,
    Int port
);
```

```
HYP_CloseConnexion.response (
    Boolean isDisConected,
);
```

client Identifica o endereço da estação com o qual se deseja finalizar conexão.

port Identifica a porta da estação.

IsConnected Identifica se a conexão foi finalizada corretamente

2.2. Primitiva para o Transferência de Dados**a) HYP_DataTransp**

Descrição: Utilizado para enviar ou receber dados à camada de transporte.

```
HYP_DataTransp.indication (
    Object data
    String client,
    Int port
);
```

```
HYP_DataTransp.response (
    Object data
    String client,
    Int port
);
```

data Identifica os dados enviados ou recebidos à camada de transporte.

client Identifica o endereço da estação com o qual se deseja estabelecer conexão.

port Identifica a porta da estação.

2.3. Primitivas para a Reserva de Recursos

a) HYP_CreateQoSNegotiator

Descrição: Utilizado para criar o negociador de QoS em redes IP.

```
HYP_CreateQoSNegotiator.indication (
    String QoSNegotiator_type
);
```

```
HYP_CreateQoSNegotiator.response (
    Boolean isQoSNegotiator
);
```

QoSNegotiator_type Indica o tipo de negociador de QoS.
isQoSNegotiator Indica se o negociador de QoS foi criado corretamente.

b) HYP_Dispatch

Descrição: Utilizado para receber mensagens de sinalização enviadas pelo cliente.

```
HYP_Dispatch.indication ( );
```

c) HYP_Request

Descrição: Utilizado para enviar mensagens de sinalização ao cliente.

```
HYP_Request.response (
    String session
    String client
    FilterSpec filterSpecList
    Trafficspec scList
);
```

session Sessão de sinalização criada.
client Cliente que envia a solicitação
FilterSpecList Identifica o tipo de filterSpec utilizado
ScList Possui os valores de Tspec e RSpec

d) HYP_DestroyQoSNegotiator

Descrição: Utilizado para destruir o negociador de QoS em redes IP.

```
HYP_DestroyQoSNegotiator.request (
    String negotiatorId
);
HYP_CreateQoSNegotiator.confirm (
```

```

        Boolean isDestroyed
    );

```

2.4. Primitivas para Monitoração de QoS

a) HYP_SenderReport

Descrição: Utilizado para indicar ao cliente os parâmetros de QoS utilizados durante uma transmissão.

```

HYP_SenderReport (
    int      session_id,
    ServiceCategory    QoS
);

```

session_id Identifica a sessão de sinalização criada entre o cliente e o servidor hipermídia.

QoS Identifica quais são os parâmetros de qualidade de serviço, a nível de transporte, enviados pelo servidor.

Apêndice C - Mensagens do Protocolo de Comunicação Hipermídia

1. Classe de Mensagens Genérico (*Kernel*)

1.1. OpenSession

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE OpenSession SYSTEM "HypProtocol.dtd">
<OpenSession>
  <user>abc</user>
  <password>sampaio</password>
  <client_id>127.0.0.1</client_id>
  <port_client>1234</port_client>
  <server_id>127.0.0.1</server_id>
  <port_server>1234</port_server>
  <profile>generic</profile>
  <session_key></session_key>
</OpenSession>
```

1.2. CloseSession

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE CloseSession SYSTEM "HypProtocol.dtd">
<CloseSession>
  <session_key>1</session_key>
</CloseSession>
```

1.3. CreateEntity

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE CreateEntity SYSTEM "HypProtocol.dtd">
<CreateEntity>
  <session_key>1</session_key>
  <mediaPipe_id>5</mediaPipe_id>
  <entity_description>audio_node</entity_description>
  <entity_id>10</entity_id>
</CreateEntity>
```

1.4. GetEntityProperties

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE GetEntityProperties SYSTEM "HypProtocol.dtd">
<GetEntityProperties>
  <session_key>1</session_key>
  <mediaPipe_id>5</mediaPipe_id>
  <entity_id>10</entity_id>
  <level>N</level>
  <entity_properties>content</entity_properties>
```

```

    <entity_type>node</entity_type>
</GetEntityProperties>

```

1.5. EntityProperty

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE EntityProperty SYSTEM "HypProtocol.dtd">
<EntityProperty>
  <type>Get</type>
  <session_key>1</session_key>
  <entity_id>10</entity_id>
  <property_label>content</property_label>
  <property_value>sound.wav</property_value>
</EntityProperty>

```

1.6. Error

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Error SYSTEM "HypProtocol.dtd">
<Error>
  <error_message>QoS Incompativel</error_message>
</Error>

```

2. Classe Controle de Acesso (Access Control)

2.1. EntityPropertyAccess

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE EntityPropertyAccess SYSTEM "HypProtocol.dtd">
<EntityPropertyAccess>
  <type>Lock</type>
  <session_key>2</session_key>
  <entity_id>3</entity_id>
  <property_label>content</property_label>
  <user_id>5</user_id>
  <isLocked>No</isLocked>
</EntityPropertyAccess>

```

2.2. EntityPermission

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE EntityPermission SYSTEM "HypProtocol.dtd">
<EntityPermission>
  <type>Get</type>
  <session_key>2</session_key>
  <entity_id>3</entity_id>
  <subject>abc</subject>
  <permission>read,write</permission>

```

```

    <old_permission>read</old_permission>
</EntityPermission>

```

2.3. PropertyPermission

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE PropertyPermission SYSTEM "HypProtocol.dtd">
<PropertyPermission>
  <type>Get</type>
  <session_key>2</session_key>
  <entity_id>3</entity_id>
  <subject>abc</subject>
  <property>content</property>
  <permission>read,write</permission>
  <old_permission>read</old_permission>
</PropertyPermission>

```

3. Classe Conteúdo (Content)

3.1. Get

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Get SYSTEM "HypProtocol.dtd">
<Get>
  <session_key>2</session_key>
  <entity_id>3</entity_id>
  <mediaPipe_id>5</mediaPipe_id>
</Get>

```

3.2. ThreatPlay

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ThreatPlay SYSTEM "HypProtocol.dtd">
<ThreatPlay>
  <type>Pause</type>
  <session_key>2</session_key>
  <mediaPipe_id>3</mediaPipe_id>
  <isOK>No</isOK>
</ThreatPlay>

```

4. Classe Qualidade de Serviço (*QualityofService*)

4.1. GetQoS

```

<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE GetQoS SYSTEM "HypProtocol.dtd">
<GetQoS>
  <session_key>abc@tradicao1054</session_key>
  <entity_id>sound.wav</entity_id>
  <TrafficSpec>

```

```

        <sampleRate>8KHz</sampleRate>
        <sampleSize>8bits</sampleSize>
        <loudness>Stereo</loudness>
        <audioCodification>PCM</audioCodification>
    </TrafficSpec>
    <ResourceSpec>5ms</ResourceSpec>
</GetQoS>

```

4.2. Reserve

```

<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE Reserve SYSTEM "HypProtocol.dtd">
<Reserve>
    <session_key>abc@tradicao1054</session_key>
    <entity_id>sound.wav</entity_id>
    <QoS>
        <serviceCategory>AudioGuaranteed</ServiceCategory>
        <audio>
            <maxDelay>5ms</maxDelay>
            <sampleRate>8 Khz</sampleRate>
            <sampleSize>8 bits</sampleSize>
            <loudness>Stereo</loudness>
            <audioCodification>PCM</audioCodification>
        </audio>
    </QoS>
</Reserve>

```

4.3. RenegotiateQoS

```

<?xml version='1.0' encoding='UTF-8'?>
<!-- Chamando uma DTD externa -->
<!DOCTYPE RenegotiateQoS SYSTEM "HypProtocol.dtd">
<RenegotiateQoS>
    <session_key>2</session_key>
    <mediaPipe_id>1</mediaPipe_id>
    <QoS>
        <serviceCategory>AudioControlledLoad</serviceCategory>

```

```
<audio>  
  <maxDelay>50</maxDelay>  
  <sampleRate>4</sampleRate>  
  <sampleSize>4</sampleSize>  
  <loudness>Mono</loudness>  
  <audioCodification>PCM</audioCodification>  
</audio>  
</QoS>  
<isOK>Yes</isOK>  
</RenegotiateQoS>
```

Apêndice D (HypProtocol.dtd) - A DTD do Protocolo Hipermídia

```
<!-- DTD do protocolo de comunicação hipermídia com QoS e Modularidade Funcional -  
-->
```



```

<!-- Mensagens do protocolo -->

<ELEMENT HypProtocol (Kernel, AccessControl, Content, QualityofService)>

<!--***** Classe de Mensagem Kernel***** -->

<ELEMENT Kernel (OpenSession, CloseSession, CreateEntity,
GetEntityProperties, EntityProperty, Error)>

<!-- Mensagem OpenSession -->
<ELEMENT OpenSession (user, password, client_id, port_client,
server_id, port_server, profile, session_key)>
<ELEMENT user (#PCDATA)>
<ELEMENT password (#PCDATA)>
<ELEMENT client_id (#PCDATA)>
<ELEMENT port_client (#PCDATA)>
<ELEMENT server_id (#PCDATA)>
<ELEMENT port_server (#PCDATA)>
<ELEMENT profile (#PCDATA)>
<ELEMENT session_key (#PCDATA)>

<!-- Mensagem CloseSession -->
<ELEMENT CloseSession (session_key)>

<!-- Mensagem CreateEntity -->
<ELEMENT CreateEntity (session_key, entity_description,
entity_id, mediaPipeId)>
<ELEMENT entity_description (#PCDATA) >
<ELEMENT entity_id (#PCDATA) >
<ELEMENT mediaPipe_id (#PCDATA)>

<!-- Mensagem EntityProperty -->
<ELEMENT EntityProperty (type, session_key, entity_id, property_label,
property_value)>
<ELEMENT type (#PCDATA) >
<ELEMENT property_label (#PCDATA) >
<ELEMENT property_value (#PCDATA) >

<!-- Mensagem GetEntityProperties -->
<ELEMENT GetEntityProperties (session_key, mediaPipeId, entity_id, level,
entity_properties, entity_type)>
<ELEMENT level (#PCDATA) >

<!-- Mensagem Error -->
<ELEMENT Erro (error_message)>
<!-- Existem diversos tipos de mensagens de erro -->
<ELEMENT error_message (#PCDATA) >

```

```

<!--***** Classe de Mensagem CooperativeWork *****-->

<ELEMENT CooperativeWork (EntityPropertyAccess, EntityPermission,
PropertyPermission)>

<!-- Mensagem EntityPropertyAccess -->
<ELEMENT EntityPropertyAccess (type, session_key, entity_id,
property_label, user_id, isLocked)>
<!-- Existem tres tipos de mensagens: IsLocked, Lock e UnLock -->
<ELEMENT user_id (#PCDATA) >
<ELEMENT isLocked (#PCDATA) >

<!-- Mensagem EntityPermission -->
<ELEMENT EntityPermission (type, session_key, entity_id,
subject, permission, old_permission)>
<ELEMENT subject (#PCDATA) >
<ELEMENT permission (#PCDATA) >
<ELEMENT old_permission (#PCDATA) >

<!-- Mensagem PropertyPermission -->
<ELEMENT PropertyPermission (type, session_key, entity_id,
subject, property, permission, old_permission)>
<ELEMENT property (#PCDATA) >

<!--***** Classe de Mensagem Content *****-->

<!-- Mensagem Get -->
<ELEMENT Get (session_key, entity_id, mediaPipe_id)>

<!-- Mensagem ThreatPlay -->
<ELEMENT ThreatPlay (type, session_key, mediaPipe_id, isOK)>
<!-- Existem tres tipos de mensagens: Pause, Resume e Stop -->
<ELEMENT isOK (#PCDATA)>

<!--***** Classe de Mensagem QualityofService *****-->

<ELEMENT QualityofService (GetQoS, Reserve, RenegotiateFlowQoS)>

<!-- Mensagem GetQoS -->
<ELEMENT GetQoS (session_key, entity_id, trafficSpec, resourceSpec)>
<ELEMENT trafficSpec (#PCDATA)>
<ELEMENT resourceSpec (#PCDATA)>

<ELEMENT Reserve(session_key, entity_id, serviceQoS)>

```

```
<!-- Mensagem Reserve -->  
  
<!ELEMENT serviceQoS (serviceCategory, audio, video, data)>  
<!ELEMENT serviceCategory (#PCDATA)>  
<!ELEMENT audio (maxDelay, sampleRate, sampleSize, loudness, audioCodification)>  
<!ELEMENT maxDelay (#PCDATA)>  
<!ELEMENT sampleRate (#PCDATA)>  
<!ELEMENT sampleSize (#PCDATA)>  
<!ELEMENT loudness (#PCDATA)>  
<!ELEMENT audioCodification (#PCDATA)>  
  
<!ELEMENT video (maxDelay, frameSize, frameRate, resolution, videoCodification)>  
<!ELEMENT frameSize (#PCDATA)>  
<!ELEMENT frameRate (#PCDATA)>  
<!ELEMENT resolution (#PCDATA)>  
<!ELEMENT videoCodification (#PCDATA)>  
  
<!ELEMENT data (dataCodification)>  
<!ELEMENT dataCodification (#PCDATA) >  
  
<!-- Mensagem RenegotiateQoS -->  
<!ELEMENT RenegotiateQoS (session_key, mediaPipe_id, QoS, isOK)>
```

Referências Bibliográficas

[1] Soares, L. F. G.; Lemos, G.; e Colcher, S. "Redes de Computadores – Das LANs, MANs às Redes ATM". Editora Campus, 2ª edição, 1995.

[2] Holzmann, G. J. "Design and Validation of Computer Protocols". *Bell Laboratories*, Editora Prentice-Hall, 1991.

- [3] Busse, I.; Deffner, B.; Schulzrinne, H. "Dynamic QoS Control of Multimedia Applications based on RTP". *Proceedings of the First International Workshop on High Speed Networks and Open Distributed Plataforms*, St. Petersburg, Russia, Junho 1995.
- [4] deCarmo, L. "Core Java Media Framework". Editora Prentice-Hall, 1999.
- [5] Schulzrinne, H.; Rao, A.; Lanphier, R. "Real Time Streaming Protocol (RTSP)". *RFC2326, IETF Network Working Group*, Abril 1998.
- [6] Schulzrinne, H. "A comprehensive multimedia control architecture for the Internet". *Proceedings of International Workshop on Network and Operating System Support for Digital Audio e Video (NOSSDAV)*, St.Louis, Missouri, Maio 1997.
- [7] Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V. "RTP: A Transport Protocol for Real-Time Applications". *RFC1889*, Janeiro 1996.
- [8] Fielding, R.; Gettys, J.; Mogul, J; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. "Hypertext Transfer Protocol - HTTP/1.1". *RFC2616*, Junho 1999.
- [9] Wilde, E. "Wilde's WWW. Technical Foundation of the World Wide Web". Editora Springer, 1999.
- [10] Whitehead Jr., E. J.; Wiggins, M. "WEBDAV: IETF Standard for Collaborative Authoring on the Web". *IEEE Internet Computing*, Setembro 1998.
- [11] Neable, C.; Lyndersay, S. "Communicating XML Data Over the Web with WebDAV". Março 2000. Disponível em:<http://msdn.microsoft.com/xml/articles/xmlandwebdav.html>.
- [12] Colcher, S. "Um Meta Modelo Para Aplicações e Serviços de Comunicação Adaptáveis com Qualidade de Servico". Tese de Doutorado, Puc-Rio, Novembro 1999.

- [13] Tanenbaum, A. S. "Computer Networks". Editora Prentice-Hall, 3ª edição, 1996.
- [14] Gomes, A. T. A. "Um Framework para Provisão de QoS em Ambientes Genéricos de Processamento e Comunicação". Tese de Mestrado, Puc-Rio, Maio 1999.
- [15] Peuker, T. "An Object-Oriented Architecture for the Real-Time Transmission of Multimedia Data Streams". Tese de Doutorado, Universitat Erlangen-Nurnberg, Março 1997.
- [16] Campbell, A.; Coulson, G.; Hutchison, D. "A Quality of Service Architecture". *ACM Computer Communications Review*, Abril 1994.
- [17] Aurrecoechea, C.; Campbell, A.T.; Haun, L. "A Survey of QoS Architectures". *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, Vol.6 No.3, pg. 138-151, Maio 1998.
- [18] Bernet, Y.; Stewart, J.; Yavatkar, R.; Andersen, D.; Tai, C.; Quinn, B.; Lee, K. "Windows Networking Group - Winsock Generic QoS Mapping (draft) – 3.1". Setembro 1998. Disponível em: <http://www.sockets.com/winsock2.htm>.
- [19] Shin, M. K.; Lee, J. Y. "Web-Based Real-time Multimedia Application for the MBone". INET, Genebra, Suíça, 1998. Disponível em: <http://pec.etri.re.kr/~mkshin/inet98/index.htm>
- [20] Chen, Z.; Tan, S.M.; Campbell, R.H.; Li, Y. "Real Time and Audio in the World Wide Web". *World Wide Web Journal*, Volume 1, No.1, Janeiro 1996. Disponível em: <http://choices.cs.uiuc.edu/Papers/New/vosaic/vosaic.html>.
- [21] Husemann, D. "Multimedia Data Streams In Distributed Object-Oriented Operating Systems". Tese de Doutorado, Universitat Erlangen-Nurnberg, Fevereiro 1996.

- [22] Schulzrinne, H.; Rosenberg, J. "Internet Telephony: Architecture and Protocols -- an IETF Perspective". *Computer Networks and ISDN Systems*, vol.31, n.3, pp. 237 - 255, Fevereiro 1999. Disponível em: <http://computer.org/internet/telephony/w3schrosen.htm>.
- [23] Soares, L.F.G. "Modelo de Contextos Aninhados Versão 2.3". Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio, Rio de Janeiro, 1999.
- [24] Rodrigues, R.F.; Muchaluat-Saade, D.C.; Soares, L.F.G. "Modeling, Authoring and Formatting Hypermedia Documents in the HyperProp System". Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio, Rio de Janeiro, Setembro 1999.
- [25] Dubuisson, O. "ASN.1 - Communication between Heterogeneous Systems", Junho 2000. Disponível em: <http://www.oss.com/asn1/booksintro.html>
- [26] "Extensible Markup Language (XML) 1.0 (Second Edition)". *W3C Working Draft*, Agosto 2000. Disponível em: <http://www.w3.org/TR/2000/WD-xml-2e-20000814>.
- [27] "XML Tutorial". Janeiro 2001. Disponível em: http://www.xml101.com/xml/xml_intro.asp.
- [28] Nayak, S. M. "XML and Messaging". *XML Journal, Sys-Con publications Inc*, Janeiro 2000. Disponível em: <http://www.sys-con.com/xml/archives/0101/nayak/index.html>.
- [29] Larmouth, J. "ASN.1 Complete". Maio 1999. Disponível em: <http://www.oss.com/asn1/bookreg.html>.
- [30] Maruyama, H.; Tamura, K.; Uramoto, N. "Xml and Java - Developing Web Applications". Editora Addison-Wesley, Janeiro 2000.

- [31] Mota, O. T. J. D. .D. L. "Serviços com Qualidade na Internet". Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio, Rio de Janeiro, Fevereiro 1999.
- [32] Braden, R.; Zhang, L.; Berson, S.; Herzog, S.; Jamin, S. "Resource ReSerVation Protocol (RSVP)". *RFC2205*, Setembro 1997.
- [33] Stiller, B.; Class, C.; Waldvogel, M.; Caronni, G.; Bauer, D.; Plattner, B. "The Design and Implementation of a Flexible Middleware for Multimedia Communications Comprising Usage Experience". Relatório Técnico (*Computer Engineering and Networks Laboratory*) - TIK, Swiss Federal Institute of Technology Zurich (ETHZ), Zurique, Suíça, Julho 1998.
- [34] Tassel, J. "Quality of Service adaptation using Reflective Java". Tese de Mestrado, Universidade Kent at Canterbury, Setembro 1997.
- [35] Concad, C.; Stiller, B. "The Design of an Application Programming Interface for QoS-based Multimedia Middleware". *22nd IEEE International Conference on Local Computers Networks*, Minneapolis, Minnesota, Novembro 1997.
- [36] Concad, C.; Stiller, B. "A QoS-based Application Programming Interface for Communication Middleware". *SPIE Vol. 3233 for the Voice, Video, and Data Communications Symposium*, Dallas, Texas, Novembro 1997.
- [37] Kerhervé, B.; Pons, A.; Bockmann, G.V.; Hafid, A. "Metadata Modeling for Quality of Service Management in Distributed Multimedia Systems". *First IEEE Metadata Conference*, Maryland, Abril 1996.
- [38] Booch, G.; Rumbaugh, J.; Jacobson, I. "The Unified Modeling Language User Guide". Editora Addison-Wesley, Outubro 1999.

- [39] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "Design Patterns - Elements of Reusable Object-Oriented Software". Editora Addison-Wesley, Setembro 1995.
- [40] Campione, M.; Walrath, K. "The Java Tutorial Second Edition: Object-Oriented Programming for the Internet (Java Series)". Editora Pearson, Janeiro 1999.
- [41] Wilde, E.; Freiburghaus, P.; Koller, D.; Plattner, B. "A Group and Session Management System for Distributed Multimedia Applications". *Workshop on Multimedia Telecommunications and Applications*, Barcelona, Espanha, Novembro 1996.
- [42] LU, G. "Communication and Computing for Distributed Multimedia Systems". Editora Norwood: Artech House, 1996.
- [43] Mendes, P.; Monteiro, E.; Guimarães, N. "Interface de Controlo de Qualidade de Serviço Para Aplicações de Trabalho Cooperativo". 1a. Conferência sobre Redes de Computadores - tecnologia e aplicações, Coimbra, Portugal, Novembro 1998.
- [44] Yamazaki, T.; Matsuda, J. "Adaptive QoS Management for Multimedia Applications in Heterogeneous Environments: A Case Study with Video QoS Mediation". *Special Issue on New Paradigms in Network Management, IEICE TRANSACTIONS COMMUNICATIONS*, VOL. E82-B, NO. 11, Novembro 1999.
- [45] Huard, J.F.; Lazar, A.A. "On End-to-End QoS Mapping". *21th IEEE Annual International Computer Software and Application Conference (COMPSAC)*, Washington, Agosto, 1997.
- [46] Huard, J.F.; Lazar, A.A. "On QoS Mapping in Multimedia Networks". *IFIP fifth International Workshop on Quality of Service (IWQoS)*, Nova Iorque, Maio, 1997.
- [47] Gopalakrishnan, R.; Parulkar, G. M. "Application Level Protocol Implementations to provide Quality-of-Service Guarantees at Endsystems". *Proceedings of Ninth IEEE Workshop on Computer Communications*, Florida, Outubro 1994.

- [48] White, P.P. "RSVP and Integrated Services in the Internet: A Tutorial". *IEEE Communications Magazine*, Maio 1997.
- [49] Nahrstedt, K.; Smith, J.M. "End-Point Resource Admission Control for Remote Control Multimedia Applications". Relatório Técnico da Universidade da Pennsylvania, Abril 1995
- [50] "WAP WSP". *WAPForum Recommendation*, Maio 2000. Disponível em: <http://www.wapforum.org/docs/copyright.htm>.
- [51] "Wireless Application Protocol White Paper". Maio 2000. Disponível em: <http://www.wapforum.org/docs/copyright.htm>.
- [52] Lucena, C.J.P.; Markiewicz, M.E. "Understanding Object-Oriented Framework Engineering". Relatório Técnico da Puc-Rio, Outubro 2000.
- [53] Sampaio Jr, A.B.C. "Especificação de Um Protocolo de Aplicação Hiperfídia com Qualidade de Serviço". Projeto Final de Programação da Puc-Rio, Dezembro 2000.
- [54] Mota, O. T. J. D. .D. L. "IPQoS: Uma Interface em Java para Solicitação de Serviços com QoS na Internet - Versão 1.02". Projeto Final de Programação da Puc-Rio, Janeiro 2001.
- [55] Nahrstedt, K.; Steinmetz, R. "Resource Management in Networked Multimedia Systems". *IEEE Computer*, Maio 1995.
- [56] "Laboratório Telemídia - Infra-estrutura". Fevereiro 2001. Disponível em: <http://www.telemidia.puc-rio.br>
- [57] Goland, Y.; Whitehead, E.; Faizi, A.; Carter, S.; Jensen, D. "HTTP Extensions for Distributed Authoring -- WEBDAV". Fevereiro 1999.

[58] Jones, P. "H.323v4 (Including Editorial Corrections)". Fevereiro 2001. Disponível em: <http://www.packetizer.com/iptel/h323/papers/>

[59] "H.323 Tutorial". *WebForum Tutorial*. Julho 1999. Disponível em: <http://www.webproforum.com/h323/>

[60] Pfeifer, A. L. "Voz sobre IP". Relatório Técnico do Laboratório TeleMídia, Departamento de Informática da PUC-Rio, Rio de Janeiro, Março 2001.

[61] Schulzrinne, H.; Handley, M.; Schooler, E.; Rosenberg, C.J. "SIP: Session Initiation Protocol". *RFC2543*, Março 1999.

[62] Handley, M.; Jacobson, V. "SDP: Session Description Protocol". *RFC2327*, Abril 1998.