Um Modelo de Qualidade de Serviço baseado em ATM para a Plataforma Multiware

Flávio Henrique de Sousa Lima

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Um Modelo de Qualidade de Serviço baseado em ATM para a Plataforma Multiware

Flávio Henrique de Sousa Lima

Julho de 1998

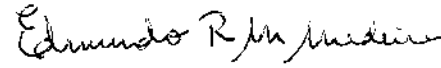
Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira
  Instituto de Computação - UNICAMP (Orientador)
- Prof. Dr. José Marcos Silva Nogueira
  Departamento de Ciência da Computação - UFMG
- Prof. Dr. Nélson Luís Saldanha da Fonseca
  Instituto de Computação - UNICAMP
- Prof. Dr. Ricardo de Oliveira Anido
  Instituto de Computação - UNICAMP (Suplente)

Um Modelo de Qualidade de Serviço baseado em ATM para a Plataforma Multiware

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Flávio Henrique de Sousa Lima e aprovada pela Banca Examinadora.

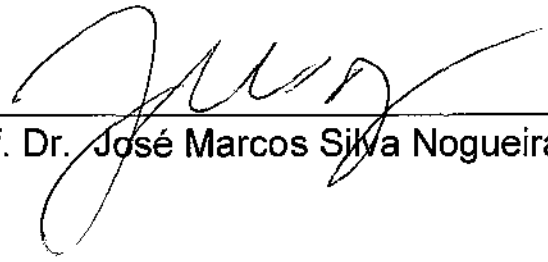
Campinas, 13 de julho de 1998.



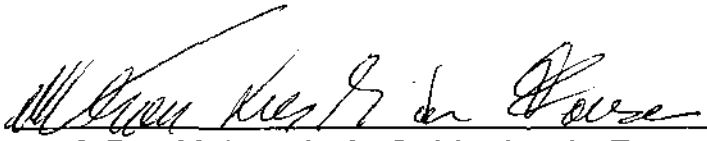
Prof. Dr. Edmundo Roberto Mauro Madeira
Instituto de Computação - UNICAMP
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

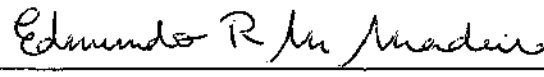
Tese de Mestrado defendida e aprovada em 13 de julho de 1998 pela Banca Examinadora composta pelos Professores Doutores



Prof. Dr. José Marcos Silva Nogueira



Prof. Dr. Nelson Luís Saldanha da Fonseca



Prof. Dr. Edmundo Roberto Mauro Madeira

Agradecimentos

Ao meu orientador, Prof. Edmundo Madeira, pela sua inesgotável paciência, da qual muitas vezes abusei, pelo seu excelente humor, mesmo de manhã cedo, e sobretudo pela figura humana admirável que ele mostrou ser, o que representou um enorme estímulo durante esse tempo todo.

Ao Prof. José Marcos Silva Nogueira, do DCC/UFMG, por haver cedido o Laboratório de Redes de Alta Velocidade para que este trabalho pudesse ser concretizado, e pela agradável convivência durante muitos meses. Aos colegas do Laboratório de ATM do DCC/UFMG, agradeço a acolhida e os muitos momentos agradáveis que compartilhamos.

À minha família, por ser meu apoio e minha referência em meio a tantas dificuldades que enfrentamos juntos. Toda vitória que eu obtiver será também mérito deles.

Aos muitos amigos e colegas do Mestrado, que fizeram desses anos uma agradável convivência e uma grande experiência de vida. Muitas foram as dificuldades que o estímulo e a alegria dos amigos ajudaram a superar.

Finalmente, agradeço a Deus, por sempre dar muito mais do que mereço, e por demonstrar Sua presença de maneiras tão sutis e tão sábias nos momentos em que minha pouca fé havia me abandonado.

Resumo

As recentes mudanças no paradigma de computação em rede, marcadas pelo surgimento de aplicações distribuídas complexas e o uso de multimídia em larga escala, representam um novo desafio com relação aos protocolos e plataformas distribuídas em uso atualmente. Nesse contexto, os aspectos não-funcionais dos sistemas, que descrevem o desempenho das aplicações, tornam-se cada vez mais importantes. Esses aspectos, agrupados sob o nome "Qualidade de Serviço", são endereçados nesta dissertação, sob o contexto de plataformas distribuídas e das redes ATM. Esta dissertação propõe um modelo de Qualidade de Serviço de múltiplos níveis para a plataforma Multiware, englobando a especificação, negociação e controle da Qualidade de Serviço. O modelo proposto é baseado em redes ATM, no modelo RM-ODP e na arquitetura CORBA. Visando a validação do modelo, foi desenvolvido um protótipo que implementa protocolos nos níveis de rede, de transporte e de distribuição.

Abstract

The recent changes occurred on the network computing paradigm, sealed by the appearance of complex distributed applications and the broad use of multimedia, represents a new challenge to the protocols and distributed platforms in use today. In this context, non-functional aspects of the systems, representing the performance of the applications, became very important. Those aspects, under que name "Quality of Service", are addressed here, in the context of distributed platforms and ATM networks. This work proposes a Quality of Service model with multiple levels to the Multiware platform, covering specification, negotiation and control of the quality of service. This model is based on ATM networks, RM-ODP model and CORBA architecture. To validate the model, a prototype has been developed, that implements protocols on network, transport and distribution levels.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Acrônimos

AAL ATM Adaptation Layer

ACID Atomicidade, Consistência, Isolamento e Durabilidade

ADAT Acknowledged Datagram

ARP Address Resolution Protocol

ATM Asynchronous Transfer Mode

BAGNet Bay Area Gigabit Network

BEO Basic Engineering Object

BERKOM Berlin Communication System

B-ISDN Basic Integrated Services Digital Network

CLP Cell Loss Priority

CMIP Common Management Internet Protocol

CORBA Common Object Request Broker Architecture

CPCS Common Part Convergence Sublayer

CPI Common Part Indicator

CRC Cyclic Redundancy Check

CSCW Computer Supported Cooperative Work

DARPA Defense Advanced Research Project Agency

DCE Distributed Computing Environment

DII Dynamic Invocation Interface

DQDB Distributed Queue Dual Bus

DSAP Destination Service Access Point

ESPRIT European Strategic Program for R&D in Information Technology

FASTNAK Fast Negative Acknowledgement

FDDI Fiber Distributed Data Interface

FTP File Transfer Protocol

GFC Generic Flow Control

HEC Header Error Control

HeiRAT Heidelberg Resource Administration Technique

IDL Interface Definition Language

IETF Internet Engineering Task Force

IP Internet Protocol

ISO International Standards Organization

ITU International Telecommunication Union

LI Length Indicator

LLC Logical Link Layer

LRM Local Resource Manager

METS Multimedia Transport System

MIB Management Information Base

MIPS Million Instructions per Second

MPEG Motion Picture Experts Group

MTBF Mean Time Between Failures

MTL Meta Transport Library

MTTR Mean Time to Repair

MTU Maximum Transfer Unit

NNI Network-to-Network Interface

NSA National Security Agency

NSAP Network Service Acess Point

NVP Network Video Protocol

OMA Object Management Architecture

OMG Object Management Group

ONQoS Objeto de Negociação de QoS

OPWA One Pass With Advertising

ORB Object Request Broker

ORL Olivetti Research Lab

PAC Protocolo de Aplicações Convencionais

PDU Protocol Data Unit

P-NNI Private Network to Network Interface

PRR Protocolo de Reserva de Recursos

PT Payload Type

PTR Protocolo de Tempo Real

PVC Permanent Virtual Circuit

PVP Packet Video Protocol

QoS Quality of Service

QoS-A QoS Architecture

QoSME QoS Management Environment

QoSOS QoS on Operating System

QuAL Quality Assurance Language

RACE R&D Advanced Communications Technologies in Europe

RAID Redundant Array of Independent Disks

RETINA Realising the Environment for TINA

RJE Remote Job Entry

RM-ODP Reference Model for Open Distributed Processing

RMON Remote Monitoring

RPC Remote Procedure Call

RSVP Reservation Setup Protocol

RTP Real Time Protocol

SCMP Stream Control Message Protocol

SDU Service Data Unit

SEAL Simple and Efficient Adaptation Layer

SNA Systems Network Architecture

SNMP Simple Network Management Protocol

SSAP Subnetwork Service Attachment Point

ST2 Stream Protocol version 2

TCCA Time Critical Communication Architecture

TCP Transmission Control Protocol

TPS Transações Por Segundo

TSDU Transport Service Data Unit

UDAT Unacknowledged Datagram

UDP User Datagram Protocol

UNI User-to-Network Interface

VCC Virtual Channel Connection

VCI Virtual Channel Identifier

VPI Virtual Path Identifier

WWW World Wide Web

XTP Xpress Transport Protocol

Capítulo 1

Introdução

Nos últimos anos tem sido visto um aumento considerável na velocidade das redes de computadores, com o surgimento de redes de alta velocidade, como FDDI, DQDB, Fast Ethernet e ATM. Tal aumento de velocidade proporcionou que uma gama de novas aplicações, antes inviáveis por restrições de desempenho, fosse criada. Entre essas aplicações estão: videoconferência, multimídia em rede, telemedicina, realidade virtual e várias outras. Além dessas novas aplicações, aquelas já utilizadas passaram a incorporar grandes volumes de dados, explorando os novos limites de velocidade.

Todas essas aplicações possuem em comum o fato de lidarem com grandes quantidades de dados, e de terem restrições temporais. Tais aplicações necessitam que a rede lhes garanta um conjunto de requisitos de comportamento, conhecidos coletivamente como Qualidade de Serviço (QoS - Quality of Service).

As aplicações convencionais, como as de transação, por sua vez passaram a ter um caráter distribuído, e com isto passaram a necessitar de garantias quanto a segurança, disponibilidade, etc. Tais garantias devem ser providas pelo ambiente distribuído no qual a aplicação executa. Esses ambientes distribuídos incorporaram novos serviços, seguindo modelos de distribuição como o RM-ODP (Reference Model for Open Distributed Processing). O RM-ODP trouxe novos serviços, e também novas atribuições às plataformas distribuídas, que passaram a ter preocupações com políticas, federações, etc.

Assim sendo, o aparecimento de novas aplicações e o caráter distribuído dessas aplicações exige da rede e do ambiente distribuído uma série de Qualidades de Serviço. Esta dissertação propõe um modelo de QoS multinível para a plataforma Multiware. Tal modelo atua nos diversos níveis abaixo das aplicações, de forma a garantir que as qualidades citadas sejam providas. A plataforma Multiware, que será estudada em detalhes nesta dissertação, é um ambiente para o desenvolvimento de aplicações distribuídas que está sendo desenvolvido na UNICAMP. O modelo de QoS que será apresentado integra-se à plataforma Multiware, aumentando sua funcionalidade.

1.1 Qualidade de Serviço

A Qualidade de Serviço, que será definida posteriormente, traduz-se em um conjunto de métricas de qualidade para uma aplicação. Tais métricas devem ser interpretadas em diferentes níveis. Por exemplo, o item disponibilidade diz respeito ao suporte à distribuição, enquanto que o fato de uma aplicação ser síncrona diz respeito ao protocolo de transporte utilizado. Por isto, a abordagem escolhida foi tratar a QoS em vários níveis. Nesta dissertação será proposta a divisão nos níveis de distribuição, de transporte e de infra-estrutura de rede.

O nível de distribuição diz respeito às características de distribuição embutidas na aplicação. Por exemplo, a transparência de localização, que torna a chamada de uma aplicação independente de sua localização. Algumas das características de QoS relativas à distribuição são providas pelo ambiente distribuído no qual a aplicação é executada, em ambientes como DCE, ANSAware ou CORBA. Porém algumas dessas características, mesmo sendo muito importantes, ainda não existem nessas plataformas, e por isto foram incorporadas neste modelo.

O nível de transporte cuida dos aspectos da comunicação fim-a-fim, como o controle de fluxo e de erro. Outro controle a ser feito pelo nível de transporte é das características de sincronismo e de tempo real existentes nas aplicações. O nível de infra-estrutura de rede trata dos aspectos específicos de cada tecnologia de rede, como a reserva de recursos. Cada tecnologia de rede trata da QoS de uma forma específica, sendo que algumas tecnologias não possuem nenhum recurso nesse sentido. Para este projeto, a tecnologia ATM (Asynchronous Transfer Mode) foi escolhida, por ser esta a que possibilita uma maior flexibilidade no uso dos recursos e um maior número de mecanismos de QoS.

Cada um desses níveis citados possui um conjunto específico de métricas de QoS, que esta dissertação propõe organizar em um modelo orientado a objeto. A orientação a objeto foi escolhida como alternativa de modelagem e implementação devido a vários quesitos. O primeiro deles foi o fato da plataforma Multiware ser inteiramente orientada a objeto, o que facilitou a incorporação do modelo à plataforma. Outro motivo foi a existência de padrões e ferramentas que facilitam o projeto e a transição para a implementação de forma suave [58]. Não menos importante é o fato de que a arquitetura CORBA é também orientada a objeto, o que determinou a possibilidade de integrar o modelo a ambientes CORBA. Da mesma forma, o modelo ODP integra-se perfeitamente ao paradigma de orientação a objetos [34].

1.2 Trabalhos relacionados

O modelo QoSME (QoS Management Environment) [27, 26] provê abstrações para o gerenciamento de QoS, dividindo o modelo nas camadas de aplicação, de "runtime" e de sistema de rede. Na camada de aplicação há uma linguagem chamada QuAL (QoS Assurance Language) que inclui abstrações em nível de linguagem para o gerenciamento de QoS. Através de QuAL é possível especificar e negociar requisitos de QoS, e também monitorar a QoS através de MIBs. No nível de "runtime" há interfaces com o sistema operacional (QoSOS - QoS on Operating System) e com a camada de transporte (QoSockets - QoS in Sockets) para prover uma interface independente do sistema de rede. O nível de sistema de rede trata dos protocolos de comunicação suportados pelo QoSME, que são: TCP, ST2 e AAL.

Vogel descreve uma arquitetura de QoS através de exemplos (Quality Query by example) para aplicações multimídia [37, 71, 72]. Nessa arquitetura, o usuário escolhe determinadas métricas de QoS, como o tipo de mídia ou a taxa de frames, vê o resultado e pode modificar novamente as métricas até encontrar valores que o satisfaçam. Além dessa interface do usuário com a QoS, são definidas também as interfaces com o sistema operacional, com o sistema de transporte e com o servidor de multimídia.

Em [9, 7, 11, 73] é descrita uma arquitetura de QoS contendo serviços e mecanismos para gerência de QoS e controle de fluxos de mídia contínua. Essa arquitetura é baseada em camadas e planos. As camadas são: física, de enlace, de rede, de transporte e de plataforma distribuída. Os planos são: de protocolo, de manutenção de QoS e de gerência de fluxo. O plano de protocolo é dividido em dois outros planos: de usuário e de controle. A função do plano de usuário é fazer a transferência de dados desejada. O plano de controle é responsável pelos mecanismos que garantem o correto funcionamento do protocolo. O plano de manutenção de QoS possui um conjunto de módulos de gerência, que agem em cada camada realizando a monitoração e a manutenção. Através de um contrato de serviço, esses gerentes mantém o nível de QoS desejado. O plano de gerência de fluxo é responsável pelo estabelecimento de fluxos e pela negociação, renegociação e adaptação de QoS.

Um bom resumo das arquiteturas de QoS é mostrado em [1]. Todos estes modelos propostos na literatura de QoS são melhor discutidos no capítulo 5, que trata do modelo aqui desenvolvido.

1.3 Estrutura da dissertação

Esta dissertação está organizada como se segue. Após esta introdução, no capítulo 2 é definida a Qualidade de Serviço e todos os seus componentes, criando uma base conceitual

e definindo o escopo de trabalho. Esse capítulo discute sobre como é tratada a QoS nos vários níveis de comunicação compreendidos entre a rede física e as aplicações de usuário. São citados também os principais requisitos de QoS nesses níveis e suas implicações. O capítulo 3 fala sobre os protocolos de rede e transporte da atualidade, e seu relacionamento com Qualidade de Serviço. Nesse capítulo é traçada uma análise comparativa desses diversos protocolos, e é justificada a escolha feita sobre quais deles implementar nesta dissertação. O capítulo 4 detalha os modelos e arquiteturas que embasaram este trabalho, que são o modelo ODP e a arquitetura CORBA. Nesse capítulo também é mostrada a plataforma Multiware, que é a base acima da qual foi desenvolvido o modelo de QoS proposto. O capítulo 5 apresenta o modelo de QoS aqui proposto em detalhes, discutindo todos os seus componentes. O capítulo 6 apresenta os aspectos de implementação envolvidos, que deram origem a um protótipo. Finalmente, algumas conclusões do trabalho são apresentadas no capítulo 7.

Capítulo 2

Qualidade de Serviço Multinível

2.1 Introdução

O objetivo deste capítulo é conceituar a Qualidade de Serviço, destacar sua importância no atual contexto da computação distribuída, citar os principais esforços de padronização da área e descrever a maior parte dos conceitos que serão trabalhados nos capítulos posteriores.

A Qualidade de Serviço tem sido extensivamente discutida como componente essencial de sistemas de computação distribuída. Particularmente a ISO (International Standards Organization) e o ITU (International Telecommunications Union) têm trabalhado em conjunto, visando construir padrões que garantam uma interpretação coerente de QoS em todos os trabalhos de padronização, visto que a QoS permeia muitas áreas da computação. Este padrão, sob o código ISO/IEC JTC1.21.57 ou ITU-T Q19/7, atualmente está no estágio de DIS (Draft International Standard), devendo brevemente tornar-se um padrão internacional, como uma nova parte do modelo ODP (padrão ITU-T X.905 e ISO/IEC 10746-5) [32, 33].

Outro trabalho importante de padronização vem sendo desenvolvido pelo OMG (Object Management Group), como parte da arquitetura OMA (Object Management Architecture) [50]. O objetivo do trabalho do OMG é garantir uma forma consistente de especificar e gerenciar QoS em ambientes baseados em CORBA (o capítulo 4 descreve a arquitetura CORBA em detalhes). Este trabalho está atualmente em fase de elaboração, e utiliza muitos dos conceitos adotados pela ISO e ITU. Outro órgão que tem mostrado uma crescente preocupação com a QoS é o IETF (Internet Engineering Task Force). O IETF é responsável por criar os padrões de evolução da Internet, tendo portanto um papel muito importante no cenário da computação baseada em rede. O IETF tem criado vários padrões para uma Internet com integração de serviços [74, 3, 5, 65], dando especial ênfase à QoS, principalmente nos aspectos de protocolos com controle de admissão e reserva de

recursos e roteamento baseado em QoS.

Além dos esforços de padronização acima descritos, muitos trabalhos independentes têm sido desenvolvidos no sentido de modelar vários aspectos de QoS, tanto no contexto de aplicações multimídia [62, 28, 25, 64, 23, 55] quanto no contexto de aplicações distribuídas genéricas [2, 31, 21, 46, 57, 24]. Alguns trabalhos são especialmente dirigidos ao modelo ODP [78, 54].

A maior parte deste capítulo é baseada nos conceitos e modelos descritos nos três esforços de padronização citados anteriormente, os quais, em função de sua representatividade, podem ser descritos como os principais trabalhos na área. Há porém há vários outros, dentre os quais estão [50]:

- O projeto ECTS da ISO (Enhanced Communications Transport Service);
- O projeto TCCA da ISO (Time Critical Communication Architecture);
- A DARPA (Defense Advanced Research Project Agency), que conduz um projeto de pesquisa em QoS;
- O grupo Eurocontrol, que tem estudado requisitos de QoS em ambientes CORBA;
- O projeto RETINA (Realizing the Environment for TINA), que pesquisa aspectos de QoS relacionados à infra-estrutura de redes, como parte do projeto TINA;
- O grupo X-Open, que tem estudado a QoS em ambientes baseados em ATM;

Como suporte ao trabalho de todos esses grupos, vários congressos internacionais abordam o tema, dentre os quais se destaca o IFIP IEEE IWQoS, exclusivamente voltado ao estudo da QoS, no qual foi apresentada parte desta dissertação [38].

2.2 Definições

Um sistema genérico pode ser definido pelas funções que ele cumpre em benefício de seus usuários. O cumprimento dessas funções pode ser visto como o "comportamento funcional" do sistema. Além de um comportamento funcional, um sistema exibirá um outro tipo de comportamento durante o cumprimento de suas funções, que é justamente a forma como a tarefa foi cumprida: quanto tempo foi gasto, quantos recursos, que falhas ocorreram, etc. Pode-se imaginar a Qualidade de Serviço como esse conjunto de aspectos "não-funcionais" do comportamento de um sistema.

O Modelo de Referência para Processamento Distribuído Aberto (RM-ODP - Reference Model for Open Distributed Processing) [34], define Qualidade de Serviço como "um

conjunto de qualidades relacionadas ao comportamento coletivo de um ou mais objetos". Jan de Meer [46, 44] define QoS como "a qualidade do desempenho de um serviço e a qualidade do resultado do serviço desempenhado". No contexto de aplicações multimídia, Vogel [71] define QoS como "o conjunto de características quantitativas e qualitativas de um sistema de multimídia distribuído, que são necessárias para obter a funcionalidade requerida de uma aplicação".

No contexto desta dissertação, a Qualidade de Serviço pode ser definida como um conjunto de fatores de qualidade, interpretados em diferentes níveis, que regulam e descrevem o comportamento de uma aplicação. A QoS regula o comportamento de uma aplicação quando é definida "a priori", e o ambiente de comunicação é preparado para estar em conformidade com esses fatores. A QoS descreve o comportamento quando, como resultado da execução da aplicação, obtém-se uma medida da QoS efetivamente provida, durante ou após a sua execução. Essas duas formas de QoS estão em constante realimentação, de acordo com um modelo de gerência, conforme mostrado na Figura 2.1. Gerência de QoS é o conjunto de atividades de monitoração, controle e administração de aspectos de Qualidade de Serviço. As atividades de gerência são direcionadas pelos requisitos dos usuários, pelo ambiente do sistema e pelas políticas em vigor. A atividade de gerência de QoS utiliza um conjunto de mecanismos para cumprir suas funções.

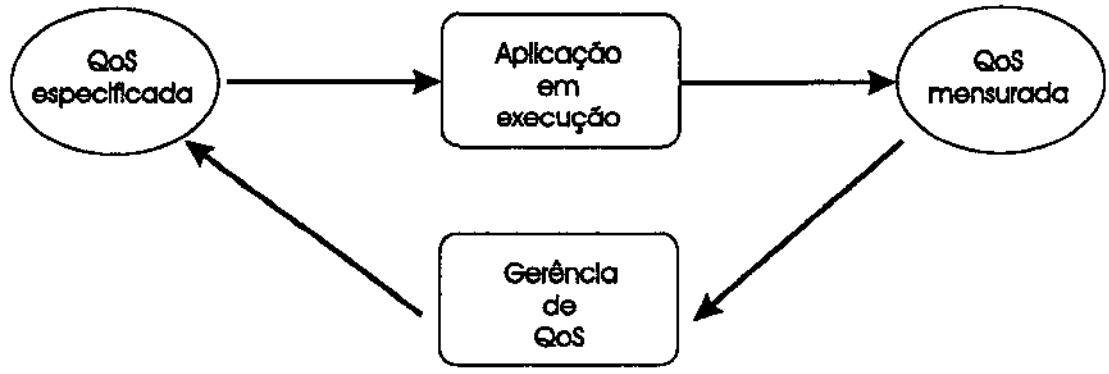


Figura 2.1: Ciclo de QoS

Juntamente com a definição de Qualidade de Serviço, torna-se necessário definir diversos conceitos que serão usados durante toda a dissertação. Para facilitar o entendimento, e também por padronização, optou-se por seguir os conceitos definidos nos padrões ISO/IEC 13236 e ISO/IEC 13243 [32, 33], que estão relacionados abaixo:

Característica de QoS É um aspecto quantificável de QoS, que é definido independentemente dos meios pelos quais ele é representado ou controlado.

Mecanismo de QoS É um mecanismo específico, que pode utilizar elementos de proto-

colos, parâmetros e contextos para suportar o estabelecimento, monitoração, manutenção, controle e pesquisa de QoS.

Contexto de QoS Representa a informação de QoS que é mantida, interpolada ou extrapolada por uma ou mais entidades e usada na gerência de QoS;

Dado de QoS É toda informação de QoS que não seja requisito, por exemplo, os valores das medições.

Informação de QoS É toda informação relacionada à QoS, que pode ser dividida em quatro categorias: contextos, parâmetros, requisitos e dados de QoS;

Medição de QoS Um ou mais valores observados, que se relacionam a características de QoS.

Parâmetro de QoS É um tipo de informação de QoS que é transportada entre entidades, como parte de um mecanismo de QoS.

Requisito de QoS É um tipo de informação de QoS que expressa, em parte ou inteiramente, uma informação requerida para gerenciar uma ou mais características de QoS, por exemplo: um valor máximo, um valor ótimo ou um valor limite. Um requisito de QoS é representado por parâmetros durante a comunicação entre entidades.

Atributo de QoS É um atributo de um objeto gerenciado relacionado à QoS.

Controle de QoS É o uso de mecanismos de QoS para modificar a condição de um sistema, de forma que um conjunto de características de QoS seja alcançado em alguma atividade, enquanto esta atividade está em andamento.

Pesquisa de QoS É o uso de mecanismos de QoS para determinar propriedades do ambiente relacionadas à QoS.

Estabelecimento de QoS É o uso de mecanismos de QoS para criar condições para alguma atividade, antes que a atividade ocorra, de forma a alcançar um conjunto de características de QoS.

Manutenção de QoS É o uso de mecanismos de QoS para manter um conjunto de características de QoS dentro de valores aceitáveis para alguma atividade, enquanto a atividade está em andamento.

Monitoração de QoS É o uso de medições de QoS para estimar os valores efetivamente alcançados de um conjunto de características de QoS.

Função de Gerência de QoS É uma função especificamente destinada a cumprir um conjunto de requisitos de QoS de um usuário ou aplicação, utilizando para isto um ou vários mecanismos de QoS.

2.3 Contextualização

Após definirmos os principais conceitos que serão utilizados, restringiremos o escopo a ser trabalhado. Os conceitos acima citados podem ser aplicados a qualquer sistema de computação. Porém, a partir deste ponto da dissertação, estes conceitos serão trabalhados no contexto de aplicações distribuídas baseadas em objetos. Ao restringir-se o escopo a aplicações distribuídas pressupõe-se a existência de uma infra-estrutura de comunicação entre as aplicações, tanto no ponto-de-vista de hardware (linhas de comunicação, modems, roteadores, etc) quanto no ponto-de-vista do software (sistemas operacionais, protocolos de comunicação, etc), cujo desempenho terá efeito determinante no nível de QoS alcançado. Ao restringir-se o escopo às aplicações baseadas em objetos, pretende-se direcionar o trabalho ao ambiente alvo, que é a plataforma Multiware, a ser descrita no capítulo 4.

O objetivo ideal de um modelo de QoS é prover Qualidade de Serviço fim-a-fim [44, 21]. Isto é, deve-se modelar a QoS em todos os elementos intermediários entre o produtor e o consumidor dos dados. De acordo com o direcionamento acima exposto, teremos três elementos centrais onde trabalhar a QoS: a infra-estrutura de comunicação, o sistema operacional e a plataforma distribuída, mostrados na Figura 2.2. O sistema operacional representa um contexto bem específico e complexo, que já foi tema de outros trabalhos na área [26, 6], e por isso será tratado somente sob o ponto-de-vista de implementação (vide capítulo 6). Assim sendo, os pontos focais a serem desenvolvidos no modelo a ser proposto serão a infra-estrutura de comunicação e a plataforma distribuída.

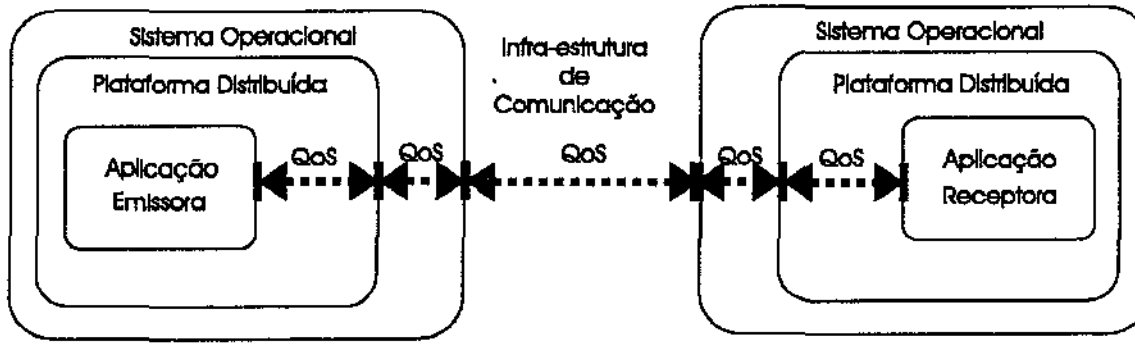


Figura 2.2: Elementos de QoS fim-a-fim

2.4 Interfaces e Negociação de QoS

Um dos principais componentes da gerência de QoS é a interface com o usuário, que é determinada em função do tipo de negociação de QoS exigido. Neste contexto, "usuário" pode significar uma pessoa interagindo com o sistema, ou pode significar também uma aplicação-usuária trocando parâmetros com o sistema.

Em alguns sistemas, a QoS é especificada somente na etapa de projeto, e garantida através de decisões do projeto, como a prioridade de tarefas, a alocação de memória, a velocidade do canal utilizado, e nesse caso não se teria nenhum tipo de interface. Outro tipo de especificação de QoS é aquela baseada em dados históricos. Através de medições feitas em comunicações anteriores, pode-se inferir requisitos de QoS para um certa categoria de comunicação. Pode-se também calcular deterministicamente esses requisitos, caso se tenha acesso a todo o padrão de transmissão. Isto acontece, por exemplo, em sistemas de distribuição de vídeo, onde é analisado o padrão de movimento da imagem, de onde se pode calcular "a priori" os requisitos de QoS. Novamente, não haveria nenhuma interface neste caso.

A QoS pode ainda ser especificada antes do estabelecimento da conexão, permanecendo estática durante a conexão. Neste caso teríamos uma especificação estática de QoS. Outro modelo possível é ter-se uma especificação de QoS, seguida de uma etapa de negociação, onde haveriam duas interfaces, de especificação e de negociação. Finalmente, pode-se ter um modelo mais complexo, onde além de especificar e negociar a QoS no estabelecimento da conexão, poder-se-ia renegociar a QoS na fase operacional da conexão. Resumindo, teremos as seguintes alternativas de negociação de QoS:

- Especificação de QoS somente durante o projeto (sem interface);
- Especificação de QoS através de dados históricos (sem interface);
- Especificação estática de parâmetros, durante o estabelecimento da conexão;
- Especificação e negociação durante o estabelecimento da conexão;
- Especificação e negociação durante o estabelecimento e renegociação na fase operacional.

Pode ainda haver aplicações específicas onde, por suas características, seja mais prático haver negociação e renegociação automáticas, sem nenhuma interface com o usuário. Isto poderia ocorrer com aplicações de tempo real e aplicações tolerantes a falhas, por exemplo.

Outra alternativa de interface é quando inexiste a possibilidade de renegociação dinâmica, e a aplicação tem seu comportamento altamente dependente do nível de QoS provido. Nestes casos uma das alternativas é criar aplicações adaptativas, isto é, aplicações

que modificam seu comportamento em função do nível de QoS provido. Um exemplo clássico são aplicações de vídeo-conferência, que podem sacrificar a qualidade da transmissão em função da QoS provida, por exemplo, reduzindo a resolução, retirando a cor, diminuindo o tamanho da janela, aumentando a compressão, etc [7, 10].

Uma interface de QoS "inteligente" deve permitir que o usuário especifique seus requisitos de acordo com sua visão do sistema. Isto implica em traduzir ou mapear os parâmetros de QoS de acordo com o nível em que se trabalha (os mecanismos de mapeamento de QoS serão explicados posteriormente). Novamente utilizando a transmissão de vídeo como exemplo, caso o usuário seja uma aplicação, parâmetros como número de quadros por segundo, atraso entre quadros e algoritmo de compressão serão adequados. Caso o usuário seja uma pessoa, o tamanho da janela de exibição, o uso ou não de cor, a resolução e a qualidade do som serão parâmetros mais significativos. A Figura 2.3 mostra um rascunho de interface com um usuário humano, e abaixo é mostrado o esboço de uma interface análoga para uma aplicação-usuária, utilizando um objeto C++.

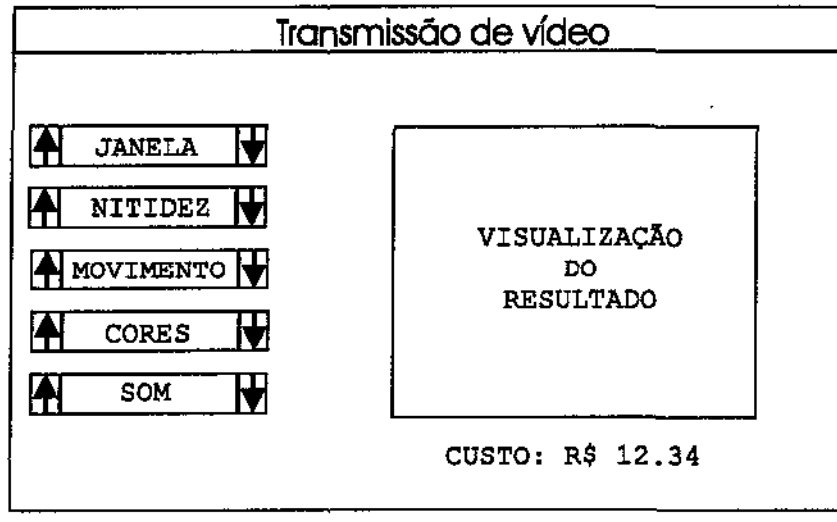


Figura 2.3: Interface de QoS com usuário humano

```
class QoS_para_video{
private: // parametros traduzidos
   int largura_de_banda;
   int MTU;
   int jitter;
   int delay;

   int traduz_QoS{};
   int reserva_recursos{};
```

```
public: // parametros para a aplicacao-usuaria
   int frames_por_segundo;
   int intervalo_entre_chegadas;
   int bits_de_cor;
   int bits_de_som;
   int perda_de_frames;
   int custo;
   int submeta_qos{};
}
```

O modelo acima descrito pode ser aplicado à especificação estática e também à negociação na etapa de estabelecimento de conexão, através de sucessivos ciclos de submissão de parâmetros e retorno, onde o retorno indicaria a existência dos recursos e o custo associado. Haveria então uma forma do usuário enviar um aceite definitivo, quando o contrato estaria firmado.

A renegociação dinâmica de QoS introduz vários complicadores na interface com o usuário. Uma renegociação de QoS pode ser provocada tanto pelo usuário quanto pelo sistema. O sistema pode provocar uma renegociação na ocorrência de erros que afetem a QoS, ou quando o contrato precisa ser revisto (acabou o tempo contratado, acabaram os créditos pagos pelo usuário ou mudou o preço dos recursos). O usuário pode provocar uma renegociação quando quer mudar seu padrão de QoS.

Caso o usuário do sistema seja uma aplicação, é possível criar uma interface assíncrona, por onde a aplicação usuária seja avisada da necessidade de renegociação, e por onde ela possa comunicar ao sistema que abriu um processo de renegociação. Por ser uma comunicação entre aplicações, pode-se criar um modelo que permita até várias renegociações por segundo. Caso o usuário seja humano, além da dificuldade de interromper suas atividades e conseguir sua atenção, há uma limitação no número de renegociações que pode ser feito. Uma alternativa seria, na etapa de negociação, entrar em acordo com o usuário quanto aos aspectos prioritários de QoS na comunicação. Por exemplo, em uma videoconferência, o usuário pode priorizar o som em relação à imagem, que seria sacrificada em caso de degradação da QoS. Uma vez acordados estes parâmetros, a renegociação seria feita sem interromper o usuário.

Outra questão importante para a negociação de QoS é o número de parceiros na comunicação. A comunicação pode ser par-a-par (peer-to-peer), de um para vários (1xN) ou de vários para vários (NxN). Na comunicação par-a-par a negociação envolve somente definir se os requisitos de QoS serão os mesmos nos dois sentidos da comunicação, ou se serão diferentes. As comunicações 1xN e NxN introduzem uma maior complexidade na negociação, principalmente se há parceiros heterogêneos. Neste caso haverá uma "árvore de QoS", definindo os requisitos em cada conexão e em cada ramo da árvore. Nesse caso,

a admissão de chamadas e a reserva de recursos serão mais complexas, porque além de depender dos requisitos da nova conexão, dependerão do ponto de entrada na árvore de QoS. Por haver essa complexidade extra, as comunicações NxN geralmente negociam a QoS com um gerente de QoS, e não diretamente com cada parceiro. As comunicações 1xN podem negociar a QoS com o elemento único ou com um gerente de QoS.

2.5 Mecanismos de QoS

Conforme definido anteriormente, os mecanismos de QoS podem atuar em diversos elementos de diversos níveis, visando o estabelecimento, monitoração, manutenção, controle e pesquisa de QoS. Além de poder atuar em diversos níveis, os mecanismos de QoS também podem atuar em diversas fases da comunicação. O padrão ISO 13236 [32] estabelece três fases onde os mecanismos podem atuar. São elas: fase de predição, fase de estabelecimento e fase operacional.

Os mecanismos utilizados na fase de predição são baseados na coleta e análise de dados históricos relativos à QoS alcançada em comunicações anteriores do mesmo tipo. De posse desses dados é possível calcular o nível de QoS que deve ser requisitado na fase de estabelecimento. É possível também calcular o impacto no nível dos recursos, caso mais esta conexão seja aceita. Por serem altamente dependentes do tipo de aplicação, esses mecanismos são geralmente implementados localmente, não havendo até o momento nenhum tipo de padronização.

Na fase de estabelecimento entram em ação os mecanismos de negociação de QoS discutidos anteriormente que, em suma, buscam determinar um conjunto de requisitos de QoS de concenso entre as partes da comunicação.

Uma vez de posse do conjunto de requisitos de QoS acordados, entram em ação os mecanismos de controle de admissão e de reserva de recursos, que trabalham juntos. A tarefa do controle de admissão é determinar se, com os recursos disponíveis no momento, é possível cumprir os requisitos de QoS da nova conexão sem prejudicar os níveis de QoS de todas as conexões já admitidas. Esses recursos envolvem buffers de recepção de pacotes, velocidade de conexão, alocação de CPUs, filas do sistema operacional, etc. O mecanismo de controle de admissão pode ser implementado diretamente pela aplicação, pelo sistema operacional, pelo protocolo de comunicação, ou até mesmo pela infra-estrutura de rede, como é o caso nas redes ATM (vide capítulo 6).

Uma vez admitida a conexão, o sistema pode utilizar um mecanismo de reserva de recursos para associar os recursos necessários à nova conexão. Neste ponto convém explicar que há basicamente dois tipos de comprometimento do sistema em relação à reserva de QoS: reservas garantidas e reservas melhor esforço (best effort). Nas reservas garantidas os recursos são exclusivamente associados a uma conexão, de forma independente

do padrão de tráfego das demais conexões. Nas reservas melhor esforço, os recursos são garantidos de forma estatística. Isto é, geralmente a soma dos recursos reservados para todas as conexões é superior aos recursos totais disponíveis. Porém, através de um tratamento estatístico, garante-se uma probabilidade mínima de que os recursos estarão livres quando necessários. Outra diferença é que a reserva exclusiva pode envolver a penalização de conexões para cumprir os requisitos contratados, não ocorrendo o mesmo em reservas melhor esforço. Métodos de acesso de mídia compartilhada, como o ethernet, utilizam reservas melhor esforço, enquanto que sistemas de rede de acesso exclusivo, como ATM e DQDB, geralmente utilizam reservas garantidas [52].

Na fase operacional destacam-se os mecanismos de monitoração e de filtragem. A monitoração de QoS envolve principalmente a coleta, tratamento e transmissão de dados de QoS, além dos alarmes [29]. Os dados de QoS coletados podem realimentar outros mecanismos, como a renegociação. Os alarmes têm o objetivo de avisar à função de gerência de QoS quando o sistema não conseguir manter os níveis de QoS. Sabendo disto, a função de gerência pode tomar ações como o uso de rotas de comunicação alternativas ou a remoção de conexões não prioritárias. A monitoração de QoS não tem padrões específicos, principalmente por ser este mecanismo e seus parâmetros muito dependentes do tipo de aplicação a que se destina. O padrão OSI recomenda utilizar-se dos padrões de gerência já consolidados, como SNMP (Simple Network Management Protocol), CMIP (Common Management Internet Protocol), ou mais recentemente, RMON (Remote Monitoring Protocol).

Outro mecanismo da fase operacional é o uso de filtros. Um filtro é um mecanismo que transforma os dados com o objetivo de alterar suas características de QoS [73]. Os filtros são mais utilizados quando há vários parceiros heterogêneos na comunicação. Um exemplo seria uma transmissão de video-conferência para receptores com diferentes disponibilidades de banda passante. Pode-se transferir um só "stream" e reduzir a qualidade da imagem para os receptores com menor disponibilidade de banda. Há basicamente três tipos de filtros: filtros de descarte, filtros de tradução e filtros implícitos. Os filtros de descarte funcionam descartando seletivamente alguns dados para diminuir os requisitos de QoS. Para descartar os dados com o mínimo de prejuizo à qualidade da transmissão, é necessário que o filtro "conheça" o tipo de codificação de dados sendo utilizado. Por exemplo, em uma transmissão de vídeo MPEG, pode-se descartar alguns frames sem interromper a imagem. Os filtros de tradução modificam os dados, por exemplo, utilizando compressão. Os filtros implícitos funcionam quando o fluxo de dados é dividido em várias conexões. Caso necessário, será possível remover um ou mais canais sem interromper o fluxo dos dados. Isto funcionaria, por exemplo, transmitindo as camadas de melhoramento de imagem de MPEG-2 em canais distintos.

2.6 QoS nos níveis de protocolos

Para determinarmos a Qualidade de Serviço, é preciso definir o conjunto de métricas de interesse para uma determinada aplicação, e estabelecer um conjunto de valores-limite para essas métricas, caracterizando os requisitos de QoS. Cada métrica deve ser coletada e interpretada em seu nível apropriado. Por exemplo, as métricas de controle de fluxo devem ser interpretadas no contexto do protocolo de transporte utilizado. Há, no entanto, métricas que possuem significado sob mais de um contexto. Por exemplo, o "throughput" de uma aplicação pode ser interpretado pelo protocolo de transporte, visando a alocação de buffers, e pela arquitetura da rede, visando a reserva de banda passante.

No contexto desta dissertação, a QoS foi dividida nos níveis de distribuição, de transporte e de infra-estrutura de rede. A Figura 2.4 mostra o relacionamento desses níveis com o modelo OSI.

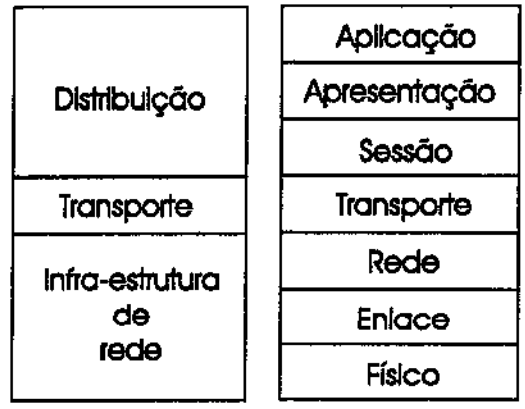


Figura 2.4: Níveis de QoS e o modelo OSI

O nível de distribuição age nas plataformas distribuídas, garantindo os requisitos de QoS de alto nível, de acordo com os recursos do modelo e da plataforma distribuída. Nessa dissertação, assumiremos o RM-ODP [34] como modelo de referência, e a arquitetura CORBA [51, 70] como padrão para as plataformas distribuídas. O modelo ODP e a arquitetura CORBA serão estudados no capítulo 4.

O nível de transporte age nos protocolos de transporte, dotando-os dos elementos necessários para a negociação da QoS. O nivel de infra-estrutura de rede age nas arquiteturas de rede, usando suas características próprias para realizar a reserva de recursos. O capítulo 3 descreve alguns protocolos de rede e de transporte dotados de mecanismos de QoS.

2.6.1 QoS na infra-estrutura de rede

Os níveis físico e de enlace determinam muitas características de QoS, porém apresentam pouca flexibilidade, por serem baseados em componentes de hardware. No capítulo 6 esses níveis serão descritos para a arquitetura utilizada na implementação, que é o ATM (Asynchronous Transfer Mode). Nesta seção somente será citado o nível de rede. O nível de rede controla a entrega de dados na inter-rede. Várias características do nível de rede determinam a QoS que será entregue aos níveis superiores, algumas delas serão citadas em seguida.

Roteamento

O roteamento determina o caminho que os pacotes de dados seguirão entre a fonte e o destino. A capacidade de entrega de QoS do nível de rede depende inteiramente da soma das capacidades individuais de cada elemento intermediário. Uma analogia possível seria com uma seqüência de tubos conectados, onde a vazão máxima é determinada pelo tubo mais fino. Da mesma forma, a capacidade de QoS fim-a-fim vai depender do elemento intermediário mais lento, ou que introduz maior atraso.

Assim sendo, é necessário que o algoritmo de roteamento conheça as métricas de qualidade de serviço. O algoritmo de roteamento deve ser integrado com a reserva de recursos, de forma a evitar os elementos intermediários que não possam prover a QoS desejada. Um determinado nível de QoS só será provido se houver um "caminho" entre fonte e destino, por onde todos os elementos intermediários garantam aquele nível de QoS. Os novos protocolos de reserva de recursos, como o RSVP, são baseados nesse paradigma.

Controle de buffers

A reserva de buffers de transmissão e de recepção é um dos elementos mais usados para garantir a QoS desejada. Mesmo que a rede não possibilite a reserva de outros recursos, é possível prover algum nível de QoS através da reserva de capacidade de transmissão e recepção. No nível de rede, esses buffers existem em cada elemento intermediário, para evitar congestionamentos e conseqüente perda de dados. Através da reserva de buffers, é possível alocar a uma determinada conexão uma certa capacidade de transmissão.

Uma outra utilidade dos buffers é a compensação do retardo variável na entrega de pacotes. Acoplando-se um temporizador aos buffers, é possível compensar variações de retardo (jitter), e com isso limitar o jitter de uma conexão, que é uma importante métrica de QoS. Os buffers também têm grande importância no controle de fluxo, que é uma característica do nível de transporte.

Controle de erros

O controle de erros, embora geralmente associado ao nível de transporte, pode ser realizado também no nível de rede. O controle de erros é uma métrica importante de QoS, seja pela sua existência ou pela ausência. Muitas aplicações, como as de transação, são baseadas no paradigma de que o meio de transmissão é capaz de entregar mensagens

sem erro. Sendo assim, é importante poder limitar a quantidade de erros a um valor probabilístico máximo, que vai depender das características do meio e dos algoritmos utilizados para detecção e correção de erros. Outras aplicações, como a transmissão de vídeo, podem suportar uma certa quantidade de erros, que é compensada pela redundância da informação. Uma transmissão de vídeo tipicamente transfere 24 a 30 quadros por segundo. Um erro que modifique parte de um único quadro pode ser facilmente tolerado. Além disso, cada quadro só possui valor se puder ser apresentado dentro de um determinado limite de tempo, e a correção de erros poderia trazer atrasos intoleráveis.

2.6.2 QoS no nível de transporte

O nível de transporte controla os recursos de comunicação fim-a-fim, e também faz a interface com os níveis superiores. Muito freqüentemente, os dados do nível de transporte são entregues diretamente às aplicações, principalmente no caso de aplicações multimídia. Vejamos a seguir cada uma das características relevantes do nível de transporte.

Controle de buffers

Da mesma forma que no nível de rede, o nível de transporte exerce controle sob buffers, sendo que neste nível os buffers são fim-a-fim. Isto possibilita um melhor controle sobre o fluxo de dados entre emissor e receptor. Muitos protocolos de transporte, conforme veremos posteriormente, possuem interfaces próprias para a alocação de buffers de transmissão e recepção.

Controle de fluxo

O controle de fluxo permite evitar que a diferença de velocidade de transmissão entre transmissor e receptor provoque a perda de dados. Isto é feito através de diversos mecanismos que permitem sinalizar ao emissor a proximidade de estouro dos buffers do receptor, o que faz com que ele diminua a taxa de emissão de pacotes. No TCP, por exemplo, isto é feito através de um mecanismo conhecido como "slow start", baseado na perda de pacotes [12].

Controle de taxa

O controle de taxa permite explicitamente limitar a taxa de envio de dados de um emissor. Esse controle é geralmente feito através de um mecanismo de fichas (tokens). Poucos protocolos, como o XTP, realmente implementam controle de taxa [75]. O controle de taxa funciona também como mecanismo de policiamento, ao evitar que os usuários utilizem mais recursos da rede do que o que foi contratado. O controle de taxa também auxilia o controle de fluxo, ao limitar a quantidade de dados que um emissor pode enviar em uma unidade de tempo.

Controle de erros

O controle de erros no nível de transporte é feito fim-a-fim, o que o torna mais adequado a redes com baixa taxa de erros que o controle no nível de rede. A ele aplicam-se as mesmas

considerações que no nível de rede.

2.6.3 QoS no nível de distribuição

O nível de distribuição é geralmente regulado através dos padrões de uma arquitetura. Nesta dissertação, assumiremos o modelo ODP por referência [34], que é descrito no capítulo 4. Em relação à Qualidade de Serviço, o modelo ODP adota uma abordagem bastante genérica. O modelo somente indica em que pontos uma interface de QoS deve existir, e quais as relações entre a QoS e os pontos-de-vista de computação e de engenharia. O modelo também mostra que a provisão de QoS é uma propriedade essencial de sistemas construídos de acordo com o modelo ODP [34]. Apresentaremos a seguir alguns elementos do modelo ODP e seu relacionamento com a Qualidade de Serviço.

Contratos

Um contrato ODP é um acordo que regula a cooperação entre objetos [35]. Um contrato pode ter aspectos dinâmicos e estáticos. Isto é, os termos do contrato podem ser modificados durante a interação. Um contrato especifica os papéis associados aos objetos, seus relacionamentos de cooperação e o tipo de comportamento que invalida o contrato. Além disso, um contrato pode especificar os aspectos de QoS da interação.

Há uma categoria especial de contratos, chamados contratos de ambiente (environment contracts), onde um objeto especifica suas restrições em relação ao ambiente, e vice-versa. Estas restrições são freqüentemente atributos de Qualidade de Serviço. O modelo ODP não define uma notação para especificar QoS em nível de linguagem computacional.

Modelo de Binding

O ponto-de-vista computacional especifica um modelo de ligação (binding), que regula a forma pela qual a associação entre interfaces computacionais é feita. Um "binding" cria um caminho de comunicação entre interfaces computacionais [36]. O "binding" pode ser implícito, que é criado pela ORB sempre que um objeto cliente referencia uma interface de operação no servidor, ou explícito. O "binding" explícito é subdividido em "binding" primitivo e composto. Um "binding" primitivo permite a ligação entre somente duas interfaces computacionais, e é iniciado por uma das entidades comunicantes. O "binding" composto permite a associação entre duas ou mais interfaces, e é controlado por um objeto especial, chamado objeto de "binding". Além de criar e manter a associação entre objetos computacionais, outra função do objeto de "binding" é lidar com os aspectos de QoS da comunicação. O objeto de "binding" tem uma interface de controle, que possibilita a negociação de QoS. Esta interface pode ser usada para notificar mudanças nos padrões de QoS, habilitando a renegociação.

A partir do ponto-de-vista de engenharia, o objeto de "binding" transforma-se nos objetos de infra-estrutura que controlam o canal de comunicação. O canal de comunicação é explicado em detalhes na seção 4.5.2. Nesta estrutura, além do objeto "binder", outro

elemento que atua no controle de QoS é o objeto de protocolo (vide Figura 4.2). O objeto de protocolo usa a infra-estrutura da rede para prover as facilidades de comunicação, de acordo com a necessidade dos objetos comunicantes [35]. O Objeto de Protocolo pode ter uma interface de controle, onde os Objetos Básicos de Engenharia (BEO - Basic Engineering Objects) podem especificar suas necessidades de QoS e receber notificações vindas da rede, que podem disparar ações de renegociação. Um modelo de "binding" baseado em QoS é proposto em [46].

Trader

O modelo ODP define um objeto especial, chamado Trader, cuja função é mediar o anúncio e a descoberta de interfaces [35]. O Trader trabalha através de ofertas de serviço, que são um conjunto de informações sobre uma interface. Essas ofertas de serviço são anunciadas e descobertas através dos papéis que um objeto pode ter: exportador e importador. O objeto exportador notifica o Trader de suas ofertas de serviço. O objeto importador consulta o Trader quando ele necessita de algum serviço (Figura 2.5).

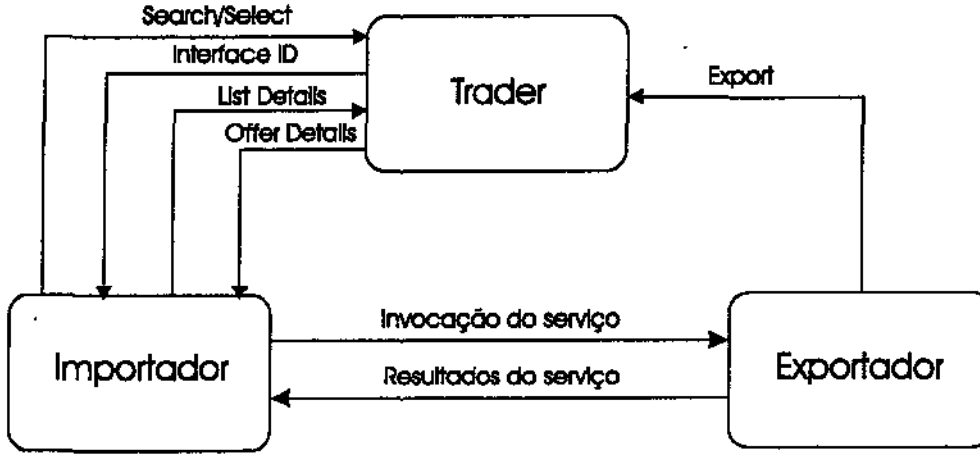


Figura 2.5: Interface do Trader

Nesse contexto, é definida também uma política de "trading", que regula o comportamento do Trader. As informações contidas nas ofertas de serviço freqüentemente contém informações de QoS. Porém, neste nível, as informações de QoS dizem respeito a requisitos de alto nível, como segurança, tolerância a falhas, disponibilidade, entre outros. Para saber mais sobre o Trader, consulte [40].

Capítulo 3

Suporte a QoS nos protocolos de rede e transporte

No capítulo 2 a Qualidade de Serviço foi conceituada e situada nos diversos níveis da comunicação, desde as redes físicas até as aplicações de usuário. Naquele capítulo também fez-se uma discussão conceitual sobre as diversas métricas de QoS e seu impacto na comunicação.

O objetivo deste capítulo é estudar protocolos que implementam aspectos de QoS nos níveis de rede e de transporte, e discutir suas principais características. Os níveis de rede e transporte foram escolhidos porque são os primeiros níveis nos quais tem-se a flexibilidade de introduzir software com características de QoS, visto que os protocolos de enlace geralmente são implementados em hardware. No caso particular das redes ATM esta afirmação não é válida, já que a AAL5, que equivale ao nível de enlace, é muitas vezes implementada em software, porém no nível de kernel do sistema operacional, onde qualquer modificação se torna complexa, passível de erros e de pouca flexibilidade [30, 22]. Isto posto, podemos considerar o nível de rede como o ponto de partida para uma implementação de QoS em software.

O nível de rede controla o roteamento, que é uma característica importante de QoS, conforme vimos no capítulo anterior. Outras características, como o controle de erros, tornam o nível de rede essencial para a provisão de QoS. O nível de transporte é também muito importante, por ser o primeiro nível onde tem-se a comunicação fim-a-fim, o que significa que este é o primeiro nível a controlar diretamente a interação entre emissor e receptor. Nesse nível pode-se mais eficientemente mensurar e administrar os parâmetros de QoS, utilizando uma semântica que muitas vezes é a mesma que é entregue à aplicação. Por exemplo, em aplicações de transmissão de áudio, os quadros manipulados pelo nível de transporte muitas vezes são diretamente entregues à aplicação, sem mais níveis intermediários. No modelo aqui proposto, acima do nível de transporte existe ainda o nível de

distribuição, que foi estudado no capítulo 2, e que é descrito em detalhes nos capítulos 4 e 5.

Ao final do capítulo, após serem citadas as principais características desses protocolos, serão discutidos os motivos da adoção do XTP como protocolo para a implementação deste projeto.

3.1 Definições

Um Protocolo de Reserva de Recursos (PRR) tem por objetivo alocar recursos na infra-estrutura de comunicação, de forma a garantir que a qualidade de serviço não seja prejudicada por outras conexões concorrentes. Esse protocolo deve "conhecer" a infra-estrutura de rede abaixo dele, e deve ser integrado ao algoritmo de roteamento, de forma que a reserva possa ser feita em todas as sub-redes que compõem a conexão. Há também PRRs direcionados à comunicação em modo datagrama, mas estes não serão tratados aqui. De acordo com as características da rede, os protocolos de reserva de recursos podem ser estatísticos ou garantidos. Para todos os efeitos, será assumido que haja uma rede ATM abaixo do PRR, e portanto, que as reservas são garantidas. Como protocolo de reserva de recursos, será analisado o RSVP, da IETF (Internet Engineering Task Force).

Um Protocolo de Tempo Real (PTR) tem por função controlar o sincronismo da comunicação. Através dele é possível corrigir ou minimizar as variações de QoS inerentes à comunicação, como o atraso e o jitter; controlar a segmentação de mensagens da forma mais adequada à aplicação (framing); regular a retransmissão e o controle de erros, etc. Os Protocolos de Tempo Real podem trabalhar na camada de rede ou de transporte. Neste capítulo serão analisados os protocolos ST2, RTP e XTP. O ST2 trabalha na camada de rede, enquanto RTP e XTP trabalham na camada de transporte.

Nesta dissertação chamaremos de "aplicações convencionais" as aplicações que não possuem requisitos de tempo real, como as aplicações TCP/IP mais conhecidas (telnet, ftp, sendmail, etc). Além de PRR e PTR, é preciso haver um Protocolo de Aplicações Convencionais (PAC), que tenha transmissão garantida (reliable), o que implica em retransmissão e controle de erros, características desnecessárias e indesejáveis em PTRs, porém essenciais em aplicações convencionais. Neste caso há duas alternativas: usar dois protocolos, um PTR e um PAC, ou usar um protocolo que, seletivamente, tenha comportamento de PTR ou PAC. Ambas as alternativas serão aqui consideradas.

3.2 Protocolos de Tempo Real

Protocolos de tempo real têm seu maior uso em aplicações multimídia, além de aplicações de sensoreamento remoto e de análise de dados em tempo real. Nas aplicações multimídia,

o principal objetivo de um PTR é recuperar o sincronismo da informação transmitida ao chegar no destino. Em análise de dados, o principal objetivo é priorizar a chegada de mensagens.

Para a recuperação de sincronismo, um PTR deve incluir informações de temporização durante a transmissão dos dados (timestamps). Esses timestamps podem conter informação de temporização relativas aos outros pacotes, ou ainda informações de tempo absoluto (clockwall time), ou ainda ambos. A vantagem de haver informação de tempo absoluto aparece quando diferentes mídias devem ser sincronizadas, como áudio e vídeo. Esse controle, chamado de "lip sync", é facilitado quando há informação de tempo absoluto [20, 18].

Devido aos requisitos de sincronização, o PTR geralmente não inclui a retransmissão de pacotes com erro. Alguns PTRs somente notificam a recepção de pacotes com erro, e outros nem isto fazem. Outro mecanismo comum aos PTRs é o controle de buffers. O controle de buffers é o mecanismo básico para minimizar os efeitos do atraso e do jitter associados à comunicação. O tamanho, o número e a localização dos buffers são fatores importantes para determinar a eficiência de um PTR.

Alguns PTRs proporcionam a priorização do tráfego. Isto é, tornam possível priorizar determinadas mídias em detrimento de outras. Por exemplo, em uma videoconferência, em caso de haver um congestionamento, o áudio é mais importante que o vídeo. Outra característica desejável em PTRs é a existência de monitoração de QoS. O PTR pode avisar à aplicação quando ocorre uma perda de QoS. Nesse caso, a aplicação pode renegociar sua QoS, ou ainda adaptar-se ao novo padrão de tráfego, no caso das aplicações adaptativas. Ainda outra característica comum aos PTRs é o controle de multicast. Os PTRs que atuam na camada de rede devem necessariamente prover esse controle, enquanto os PTRs da camada de transporte podem utilizar o suporte a multicast provido pela camada de rede, acrescido de mais alguma funcionalidade extra.

3.2.1 XTP - eXpress Transport Protocol

O XTP (eXpress Transport Protocol) é um protocolo estritamente de transporte, que foi desenvolvido para complementar e aperfeiçoar alguns aspectos do TCP, que o tornam pouco adequado para a utilização em novos modelos de comunicação, como redes ATM, troca de mensagens em clusters de servidores, transmissões multimídia, etc. O XTP está em sua versão 4.0 [75], e atualmente seu desenvolvimento está a cargo do XTP Forum, um órgão de padronização que inclui grandes empresas de informática, laboratórios, muitas Universidades e centros de pesquisa. Tamanho esforço deu origem a várias implementações do XTP, que têm sido utilizadas largamente em ambientes de laboratório e de produção.

A principal vantagem do XTP é a sua extrema flexibilidade, onde cada aspecto do protocolo pode ser modificado ou desativado separadamente dos demais. Isto possibilita

que o protocolo seja moldado para atuar como protocolo de aplicações convencionais ou protocolo de tempo real, de acordo com a necessidade. Suas principais características são [75]:

Separação de paradigma e política Os mecanismos de controle do XTP são ortogonais, isto é, pode-se controlar cada um deles separadamente, sem afetar os demais. Por exemplo, o controle do paradigma de comunicação (datagrama, circuito virtual, transações) é separado da política de controle de erros (detecção, detecção e correção), permitindo uma grande flexibilidade na adaptação do protocolo. Pode-se habilitar ou desabilitar cada aspecto do protocolo.

Separação de controle de taxa e controle de fluxo O controle de fluxo opera fim-a-fim, controlando os buffers de comunicação. O controle de taxa opera entre produtor e consumidor. No XTP, há mecanismos para controlar fluxo e taxa independentemente.

Controle explícito de multicast confiável Todos os mecanismos do XTP disponíveis para comunicação unicast estão também disponíveis em modo multicast. Isto permite controlar os mecanismos do protocolo sem se preocupar com o número de parceiros na comunicação.

Independência do serviço de entrega de dados Para que o XTP funcione, somente se faz necessário que a camada de rede proporcione a segmentação e a entrega de dados entre hosts XTP. Isto é possivel virtualmente através de qualquer protocolo de rede, ou ainda de protocolos de enlace. Há implementações de XTP funcionando acima de IP, UDP, AAL5 e outros. O XTP possui um esquema de formato de endereçamento flexível, o que facilita seu uso em diversos protocolos de rede.

O XTP possui outras facilidades, dentre as quais estão [75]:

- Modo de conexão rápida para circuitos virtuais;
- Busca de endereços baseada em chave;
- Escalonamento e priorização de mensagens;
- Suporte a protocolos encapsulados e de convergência;
- Retransmissão e confirmação seletivas;
- Frames fixos e alinhados em 64 bits;
- Identificadores de conexão e de seqüência de 64 bits;

- Negociação de tráfego e de QoS parametrizadas.

A Figura 3.1 mostra o modelo de comunicação do XTP. O protocolo XTP é baseado em contextos, que são o conjunto de informações de estado de uma conexão em cada um dos "endpoints" (pode haver mais que dois "endpoints", já que o XTP é inteiramente baseado em multicast). A operação do XTP é baseada na instanciação e manipulação dos contextos. Cada host XTP pode ter vários contextos ativos, um para cada comunicação. O conjunto formado pelos contextos ativos e pelos "streams" de dados é chamado de associação.

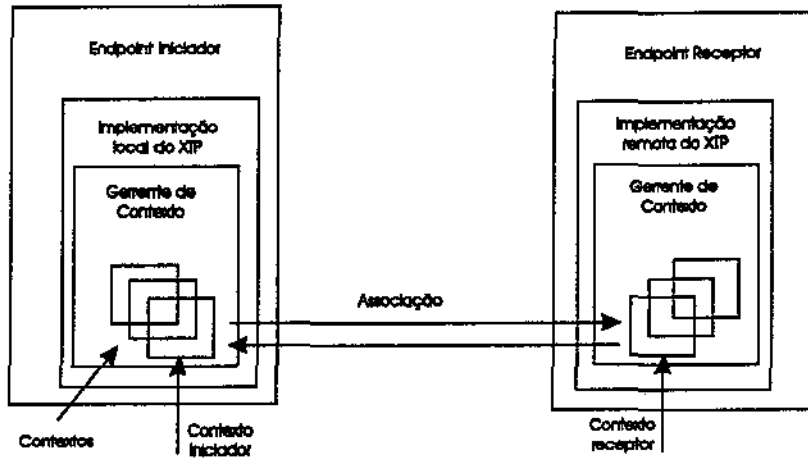


Figura 3.1: Contextos XTP

A Figura 3.2 mostra o estabelecimento de uma conexão XTP. O início de uma sessão é marcado pelo envio de um pacote do tipo FIRST. Conforme visto anteriormente, o XTP possui um endereçamento flexível. O tipo de endereçamento e o próprio endereço de destino são enviados dentro do pacote FIRST, que é o único pacote XTP que carrega informação explícita de endereçamento. Todos os outros pacotes utilizam uma chave, que é o identificador único de um "endpoint".

A Figura 3.3 mostra a máquina de estados dos possíveis contextos do XTP. Ao enviar o pacote FIRST, o contexto no iniciador entra no estado ativo. Ao chegar no receptor, o pacote FIRST é comparado com todos os contextos ativos. Caso o pacote FIRST seja aceito, o contexto no destino passa para o estado ativo, e a associação é criada. Neste ponto são criados dois canais de dados (data streams), um para cada direção. Após todos os dados terem sido transmitidos, flags do cabeçalho são usados para sinalizar o final da conexão. Quando os dois canais de dados são fechados, os endpoints entram no estado inativo. Quando a associação é dissolvida, o contexto é movido para o estado quiescente.

A estrutura do cabeçalho de um pacote XTP é mostrada na Figura 3.4. O campo "key" contém um identificador de conexão de 64 bits. O campo "cmd" contém quinze flags,

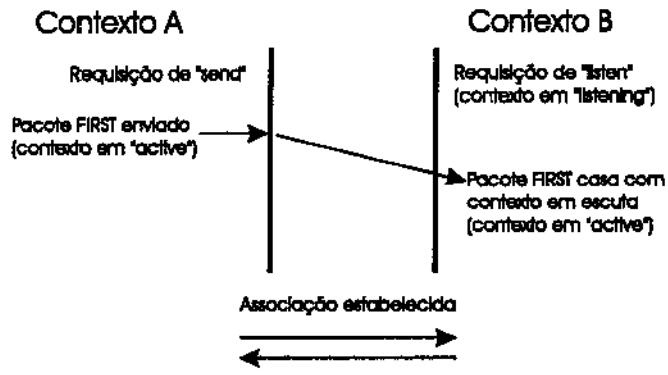


Figura 3.2: Associações no XTP

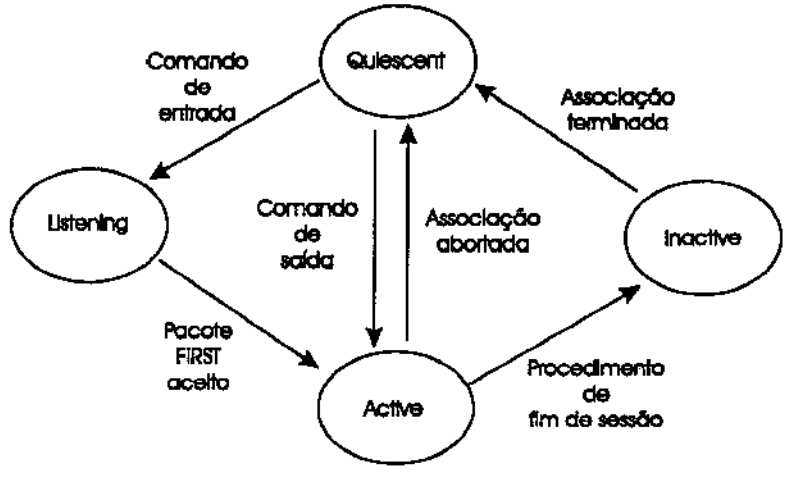


Figura 3.3: Máquina de estados do XTP

usados para habilitar diversos aspectos do protocolo, como os vários tipos de controles de erros, o uso de multicast, etc. O campo "dlen" indica o número de bytes que seguem no pacote. Um pacote XTP pode ter o tamanho de até 65536 bytes. O campo "check" contém um checksum, que pode incluir somente o cabeçalho ou todo o pacote. O campo "sort" indica a prioridade do pacote, que é outro elemento de QoS presente no XTP. Caso o flag SORT esteja ativo, todo o processamento do pacote nas filas de recepção e transmissão obedece à prioridade especificada. O campo "sync" determina o tipo de "handshake" de sincronização que será usado. Finalmente, o campo "seq" contém um número de seqüência dos dados, de 64 bits.

Figura 3.4: Cabeçalho do XTP

O cabeçalho de todos os pacotes XTP possui o mesmo tamanho (32 bytes), o que possibilita que o processamento seja muito mais rápido, já que não há necessidade de calcular o ponto de início dos dados, como no TCP. O fato de não haver informação explícita de endereçamento, mas somente uma chave genérica, torna o XTP adaptável a vários formatos de endereçamento. A Tabela 3.1 mostra os formatos de endereçamento atualmente aceitos pelo XTP. Pode-se notar também que não há nenhuma informação de roteamento no protocolo, embora ainda houvesse na versão anterior do XTP (3.6). Ao retirar a informação de roteamento, optou-se por tornar o XTP um protocolo estritamente de transporte, deixando esta tarefa para o nível de rede. Pode-se notar também o uso intensivo de flags, que tornam o protocolo bastante flexível. Pode-se modificar a operação do protocolo em cada conexão, e até mesmo em cada pacote, para alguns tipos de controle.

Campo *aformat*	Sintaxe
0x00	Endereçamento nulo
0x01	Internet Protocol
0x02	ISO CLNP
0x03	Xerox Network System
0x04	Internetwork Packet Exchange
0x05	Endereço local
0x06	Internet Protocol versão 6

Tabela 3.1: Formatos de Endereçamento no XTP

O XTP também inclui um mecanismo de controle de tráfego bastante flexível, onde o principal componente é uma especificação de tráfego, cujo formato é mostrado na Figura

3.5. Os campos rseq, alloc, echo, rsvd e xkey controlam aspectos genéricos do XTP. O campo "tlen" indica o tamanho da especificação de tráfego. O campo "service" indica o tipo de serviço de transporte desejado, cujas opções são mostradas na Tabela 3.2. O campo "tformat" indica o formato dos parâmetros de tráfego desejados, onde somente um formato foi especificado. O campo "traffic" contém os parâmetros de tráfego, de acordo com o formato desejado. O único formato descrito na versão 4.0 do XTP é mostrado na Figura 3.6. Os parâmetros deste formato são o tamanho do maior segmento a ser transmitido na conexão (maxdata), a taxa máxima de recepção de dados em bytes por segundo (inrate), o maior tamanho de rajada de dados, em bytes (inburst) e os correspondentes no sentido oposto da conexão (outrate e outburst). Outro formato de tráfego foi definido pela Universidade Técnica de Berlim, e pode ser consultado em [75].

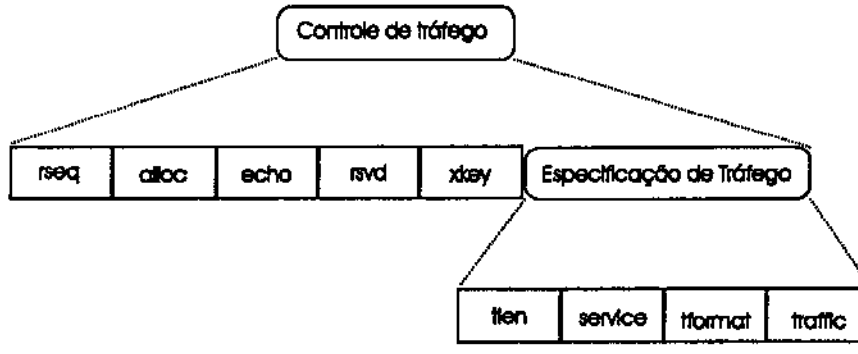


Figura 3.5: Especificação de tráfego

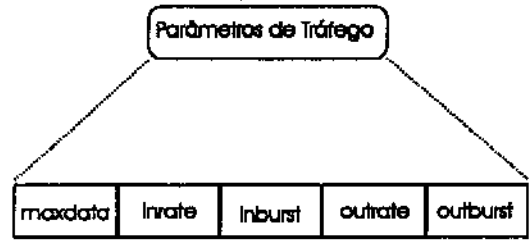


Figura 3.6: Parâmetros de tráfego

Uma vez definida a especificação de tráfego, há uma etapa de negociação, que ocorre durante o estabelecimento da conexão, em duas etapas. Inicialmente, o iniciador da conexão envia sua especificação de tráfego. O receptor tem então três alternativas: aceitar a especificação, rejeitá-la, ou aceitá-la com modificações. Nos dois primeiros casos, é enviado um pacote de diagnóstico, confirmando ou terminando a conexão. No terceiro caso, a especificação é devolvida com as modificações sugeridas, e o iniciador tem a opção de aceitar as modificações, confirmando a conexão, ou rejeitá-las, caso em que a conexão é terminada.

Campo *service*	Tipo de serviço
0x00	Não especificado
0x01	Datagrama não confirmado
0x02	Datagrama com confirmação
0x03	Transação
0x04	Stream unicast confiável
0x05	Stream multicast não-confiável
0x06	Stream multicast confiável

Tabela 3.2: Tipos de Serviço de Transporte no XTP

O controle de fluxo no XTP é baseado em um mecanismo de janelas deslizantes, de acordo com o número de seqüência dos dados. O controle de taxa, que é separado do controle de fluxo, é feito através da especificação de tráfego citada no parágrafo anterior. Dados os valores de taxa e rajada, um cálculo é efetuado, gerando um valor de "crédito", que corresponde ao número de bytes que podem ser transmitidos naquele intervalo.

O controle de erros no XTP é também bastante flexível. A detecção de erros é baseada em um checksum, que pode cobrir todo o pacote ou somente o cabeçalho. O controle de reconhecimento (acknowledgement) pode ser feito de forma síncrona entre emissor e receptor, ou ainda em um modo assíncrono, chamado reconhecimento negativo rápido (fast negative acknowledment). A retransmissão é feita por duas técnicas alternativas: go-back-n ou retransmissão seletiva. Na técnica go-back-n, quando a perda de dados é detectada, é enviado um pacote de controle indicando um número de seqüência a partir do qual deve-se iniciar a retransmissão. Na retransmissão seletiva indica-se explicitamente os intervalos de perda de dados, e somente estes são retransmitidos.

3.2.2 RTP - Real Time Protocol

O RTP (Real Time Protocol), como o nome sugere, é um protocolo de tempo real [63]. Ele foi desenvolvido pela IETF (Internet Engineering Task Force), para dar suporte a aplicações que transmitem dados de tempo real, dando particular ênfase ao uso através da Internet. Essas aplicações de tempo real podem ser áudio, vídeo, ou dados de simulação, por exemplo. Portanto, seu principal objetivo é recuperar o sincronismo de dados isócronos, geralmente de multimídia.

O RTP é também um protocolo estritamente de transporte, podendo utilizar IP, UDP, ST2 ou AAL5 como protocolos de entrega de dados. Seu uso mais comum é acima do UDP, para utilizar-se das capacidades de multiplexação por portas e do checksum do UDP. Ao contrário do XTP, o RTP é orientado a datagramas. O RTP suporta transmissões

multicast, desde que o protocolo de entrega de dados dê suporte a esta funcionalidade.

O RTP não provê nenhum mecanismo de reserva, nem provê garantias de Qualidade de Serviço, deixando que estas funcionalidades sejam providas pelos protocolos abaixo dele. O RTP também não garante a entrega de dados em seqüência, e nem mesmo garante que um certo datagrama seja entregue (entrega não confiável), por ser orientado a datagramas. As principais características do RTP são [63]:

- Identificação do tipo de dado transmitido;
- Numeração de seqüência dos dados;
- Aplicação de timestamps;
- Monitoração de entrega de dados.

Pode-se verificar que o RTP possui duas principais funções: introduzir controles de tempo real nos dados e monitorar a entrega dos mesmos. Através dos controles introduzidos é possível recuperar o sincronismo e adaptar os dados a diferentes receptores. Através da monitoração é possível reagir a uma Qualidade de Serviço insatisfatória, por exemplo aumentando a banda reservada, redirecionando o tráfego ou diminuindo a qualidade do dado transmitido.

A identificação do tipo de dado transmitido permite separar ou juntar diferentes mídias, de acordo com os requisitos e características do receptor. A numeração de seqüência dos dados permite estimar a perda de dados no trajeto. A aplicação de timestamps permite sincronizar mídias transmitidas separadamente, como áudio e vídeo, e a monitoração da entrega, conforme dito anteriormente, permite ajustar a qualidade de serviço da comunicação.

O RTP foi construído para prover a informação de sincronismo necessária a uma aplicação, podendo ser integrado à aplicação ao invés de funcionar como uma camada separada. O RTP é um "framework" de protocolo propositalmente incompleto [63]. Dentro desta estratégia, já foi disponibilizado o código-fonte de implementações do RTP, que foi ligado à própria aplicação diretamente. Isto facilita o desenvolvimento das aplicações de tempo real e torna mais rápido o processamento dos dados.

O RTP é composto basicamente de dois protocolos: O RTP (Real Time Transport Protocol), responsável pelo transporte de dados de tempo real, e o RTCP (Real Time Control Protocol), que monitora a Qualidade de Serviço e entrega informações sobre a QoS para os participantes da comunicação. O RTP opera criando dois canais de comunicação, com identificadores de nível de transporte diferentes (ports), sendo um "port" para o transporte de dados e outro para controle. Quando se transmite duas mídias, como áudio e vídeo, são utilizados dois pares de "ports" independentes.

O formato de um pacote de dados RTP é mostrado na Figura 3.7. O campo "version" identifica a versão do RTP utilizada, sendo a versão 2 a mais recente. O campo "padding" é um único bit que, quando setado indica o uso de octetos de enchimento no final do pacote, onde o número de octetos de enchimento está também no final do pacote. O campo "extension" indica o uso de um cabeçalho de extensão. O campo "CSRC count" indica a quantidade de fontes que contribuiram para a formação do "stream". O campo "marker" é utilizado para marcar o final de eventos significativos para a aplicação, como o final de um quadro de dados. O campo "payload type" indica o tipo de mídia sendo transmitida, que pode variar ao longo do tempo. O campo "sequence number", de 16 bits, contém um número de seqüência dos dados, que inicia com um valor randômico (por razões de segurança, evitando ataques do tipo "plaintext") e é acrescido de um a cada pacote de dados transmitido. O campo "timestamp", de 32 bits, representa o instante de coleta do primeiro octeto do pacote de dados, que deve ser obtido de um relógio que incremente o tempo monotonicamente e linearmente. Além deste timestamp no cabeçalho de dados, o RTP possui um timestamp no protocolo de controle, que envia informação de tempo absoluto ("clockwall time"), de acordo com o padrão NTP (Network Time Protocol). Os timestamps são utilizados tanto para sincronizar diferentes mídias tanto para calcular o jitter. O campo SSRC identifica a fonte de sincronização utilizada na transmissão. O campo "CSRC list" contém uma lista dos identificadores de cada uma das fontes que contribuiram para a formação do "stream". A quantidade de fontes é indicada no campo "CSRC count", explicado anteriormente.

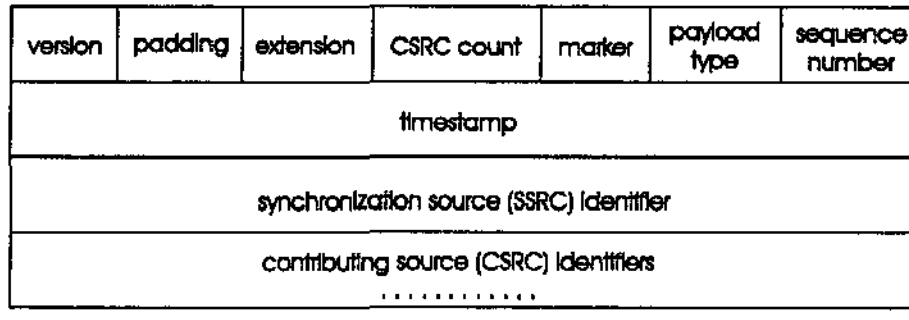


Figura 3.7: Formato do pacote RTP

O protocolo de controle, chamado RTCP, baseia-se na transmissão periódica de pacotes de controle a todos os participantes de uma sessão. O RTCP é transmitido da mesma forma que os pacotes de dados, exceto que utilizando um "port" distinto. Suas principais funções são [63]:

- Disponibilizar informação sobre a qualidade da distribuição dos dados;
- Controlar os participantes em uma distribuição de dados;

- Controlar a taxa com que são enviadas as informações de controle;
- Prover informação de identificação dos participantes na distribuição.

O RTCP possui cinco tipos de pacotes, que são:

SR - Sender Report Utilizado para a transmissão e recepção de estatísticas dos participantes ativos na distribuição;

RR - Receiver Report Utilizado para a recepção de estatísticas de participantes não ativos;

SDES - Source Description Utilizado para a identificação dos participantes;

BYE Indica o fim de participação;

APP Reservado para funções específicas de aplicações.

Os pacotes RTCP contêm uma parte fixa, similar ao cabeçalho de dados, e uma parte composta de elementos estruturados de tamanho variável. A própria taxa com que são enviados os pacotes de controle é também variável.

A quantidade, o conteúdo e a periodicidade dos pacotes de controle varia de forma a acomodar distribuições de dados que iniciam em alguns poucos participantes e podem chegar à casa dos milhares. O tráfego de controle é manipulado de forma a corresponder a uma pequena fração da banda de dados. Na especificação do RTP sugere-se que, caso exista, a reserva de banda seja limitada a cinco por cento para o tráfego de controle [63].

Os pacotes SR e RR, que são os mais significativos, incluem estatísticas sobre a perda de pacotes, o atraso e o jitter da comunicação, contadores de pacotes transmitidos e recebidos, além de informação de tempo absoluto, de acordo com o padrão NTP.

3.2.3 ST2 - Stream Protocol version 2

O ST2 (Stream Protocol version 2) é um protocolo experimental de reserva de recursos, especificado pela IETF, com o objetivo de prover garantias de qualidade de serviço na transmissão de dados de tempo real através de uma internet [17]. A primeira versão deste protocolo, chamada de ST, foi desenvolvida no final da década de 70, tendo sido muito utilizada em experimentos para a transmissão de voz, vídeo e dados de simulação através da Internet. Uma nova versão do ST, chamada ST2, foi especificada em 1990 através da RFC1190, desta vez incorporando os avanços na transmissão de dados multimídia, especialmente as novas redes de alta velocidade. Em 1995 foi feita outra revisão na especificação do ST2 através da RFC 1819, onde o objetivo foi a correção de erros, a

simplificação e a garantia de interoperabilidade entre implementações de ST2. Esta revisão do protocolo novamente mudou seu nome para ST2+, que é a versão atual [17] (por simplificação, usaremos "ST2" como o nome do protocolo, porém referindo-se à versão mais recente).

O ST2 é um protocolo de reserva de recursos e de transmissão de dados no nível de rede, orientado a conexão, que opera no mesmo nível que o IP [17]. Seu objetivo é suportar a entrega de "streams" de dados em modo unicast ou multicast, para aplicações que necessitam de Qualidade de Serviço. Suas principais aplicações são para o transporte de dados multimídia em tempo real, e também para simulação distribuída. O objetivo do ST2 não é substituir o IP, mas complementá-lo, ao incluir garantias de QoS para dados de tempo real. O objetivo é que o IP seja utilizado para aplicações convencionais e o ST2 para aplicações de tempo real. O ST2 trabalha com QoS através da reserva de largura de banda. A reserva de largura de banda, juntamente com mecanismos de acesso e de escalonamento de pacotes garante a QoS desejada.

O ST2 é composto de dois protocolos: o ST (Stream protocol), para entrega de dados, e o SCMP (Stream Control Message Protocol), para o controle das funções do protocolo, onde as mensagens SCMP são transferidas dentro de pacotes ST [17]. O ST e o SCMP são análogos ao IP e o ICMP, respectivamente. A Figura 3.8 mostra como o ST2 se integra às aplicações de tempo real e aos protocolos da família IP.

As principais aplicações a fazerem uso do ST2 são as aplicações multimídia. Na Figura, PVP (Packet Video Protocol) e NVP (Network Voice Protocol) são aplicações típicas de multimídia pela Internet. No nível de transporte há vários protocolos que utilizam o ST2, como o RTP, estudado anteriormente, e o HeiTP (Heidelberg Transport Protocol). Conforme foi dito, o ST2 opera no mesmo nível que o IP. É também possível que pacotes ST sejam encapsulados em pacotes IP, por exemplo para atravessar roteadores que não suportem o ST2. Este encapsulamento não é recomendável, já que não haveria como garantir a QoS nesse trecho da rota. Outra configuração possível mostrada é usar o ST para transportar os protocolos de transporte UDP e TCP. Tal configuração também não é recomendada, já que TCP e UDP não possuem suporte a tráfego de tempo real, mascarando a funcionalidade do ST2. Da mesma forma que o IP, o ST é independente da sub-rede abaixo dele. Embora não tenha sido mostrado, o ST2 utiliza o protocolo ARP para a resolução de endereços, da mesma forma que o IP.

O ST2 é baseado em um modelo de comunicação em dois passos. Primeiramente, os canais de tempo real são criados, na fase de configuração de "streams". Nesta fase são selecionadas as rotas e os recursos são reservados. Na segunda fase, de transferência de dados, os canais de tempo real são utilizados para transferir dados com requisitos de QoS. Esta divisão é feita porque, na primeira fase, não é necessário garantir a QoS na transmissão. A arquitetura do ST2 é composta dos seguintes elementos [17]:

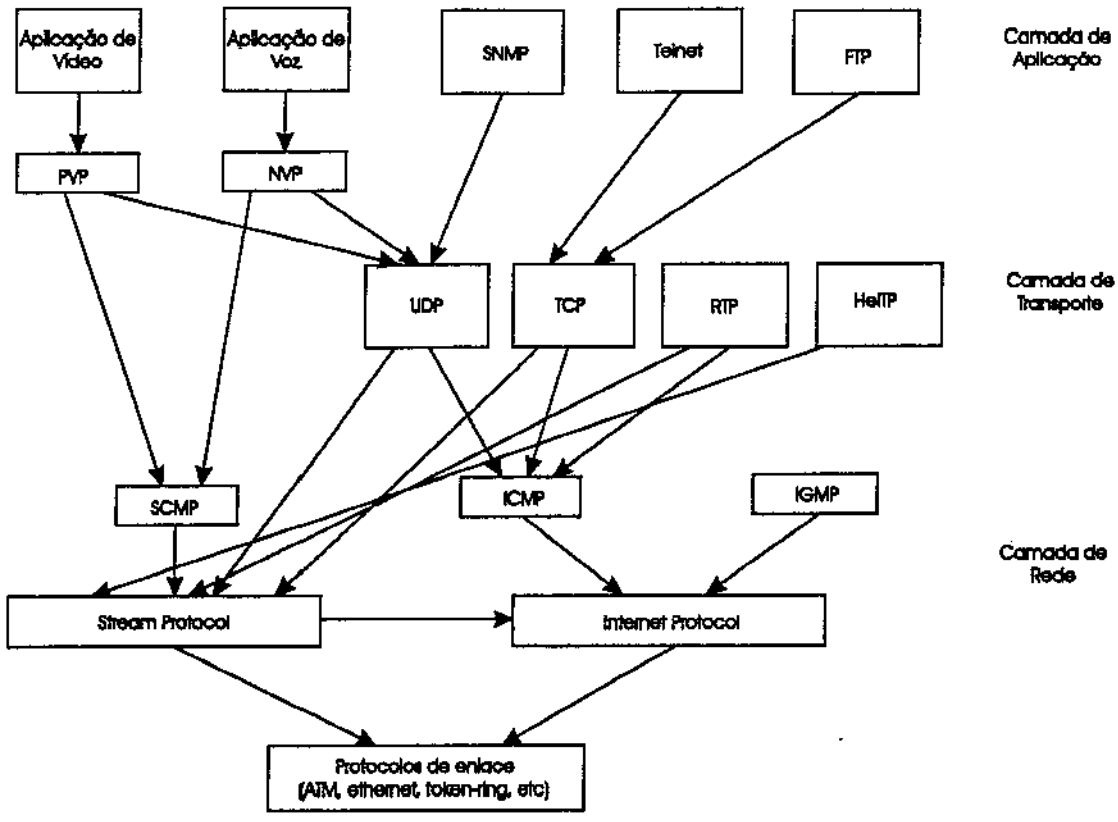


Figura 3.8: Interação do ST2 dentro da família IP

- Um protocolo de transferência de dados em tempo real usando "streams";
- Um protocolo para estabelecer "streams" de tempo real baseado em uma especificação de fluxo;
- Uma especificação de fluxo, expressando os requisitos de tempo real;
- Uma função de roteamento;
- Um gerenciador de recursos locais, para controlar os recursos de QoS.

A função de roteamento e o gerenciador de recursos locais não são especificados pelo ST2, assumindo-se sua existência. Um exemplo de gerenciador de recursos locais é o HeiRAT (Heidelberg Resource Administration Technique). A seguir comenta-se cada um desses elementos.

O protocolo de transferência de dados, conhecido como ST, define o formato dos pacotes de dados que pertencem ao "stream". Todo pacote de dados contém um identificador, que o distingüe dos diversos "streams".

O protocolo de estabelecimento de "streams", conhecido como SCMP, é responsável por estabelecer, manter e terminar "streams" de tempo real. Ele depende da função de roteamento para selecionar o caminho entre fonte e destino. Em cada roteador do caminho é feito um pedido de reserva de recursos, de acordo com a especificação de fluxo. O gerenciador de recursos locais é responsável por efetivar a reserva de recursos localmente.

A especificação de fluxo é uma estrutura de dados que representa os requisitos de QoS das aplicações. A especificação de fluxo é carregada pelo protocolo de estabelecimento de "streams", e entregue a todos os roteadores do caminho. Cada roteador chama então o gerenciador de recursos locais, que lê a especificação de fluxo e reserva a QoS. O conteúdo da especificação de fluxo é transparente ao protocolo de estabelecimento de "streams", não sendo interpretado por ele. A especificação do ST2 define um formato padrão de especificação de fluxo, onde uma única taxa de transmissão é entregue a todos os destinos (no caso de multicast). O documento, porém, deixa livre a criação de outros formatos de fluxo.

A função de roteamento é externa ao ST2, isto é, não é definida por ele, mas tem de existir. A função de roteamento é chamada pelo ST2 para cada um dos roteadores do caminho, e uma vez definida uma rota, ela é fixa durante a duração do "stream", o que é uma diferença significativa em relação ao IP. Esta capacidade é conhecida como "route recording" (gravação de rota), que não deve ser confundida com "source routing", que é uma característica do IP que não é suportada pelo ST2. A função de roteamento pode escolher uma determinada rota baseada no número de "hops", no uso dos recursos ou em outra métrica. O protocolo de estabelecimento de "streams" não sabe qual é, nem interfere na métrica utilizada.

O gerenciador de recursos locais, como o próprio nome diz, é responsável pelos recursos de cada roteador. Esses recursos podem ser: utilização das CPUs, espaço em memória, alocação de buffers, e principalmente, controle da banda passante. Entre as funções do gerenciador de recursos locais, estão [17]:

Controle de admissão de chamadas Determina se, dada uma especificação de fluxo, o roteador tem recursos suficientes para atender o pedido enquanto mantém a QoS dos outros "streams";

Cálculo de QoS Calcula a QoS que o roteador é capaz de prover, dadas as condições de tráfego atuais;

Reserva de recursos Reserva os recursos, de acordo com a especificação de tráfego.

Controle de QoS Escalona os "streams" abertos de acordo com os requisitos de QoS.

Além dessas funções, o gerenciador de recursos locais pode incluir funções de policiamento e regulação de tráfego, entre outras. A transmissão de dados no ST2 guarda algumas diferenças em relação ao IP. Uma delas é que os "streams" de dados são unidirecionais. Outra diferença é que o ST2 não suporta a fragmentação de pacotes nos nós intermediários. Isto é evitado comunicando aos roteadores e aos níveis superiores o tamanho máximo dos pacotes de dados. O formato do pacote de dados do ST2 é mostrado na Figura 3.9.

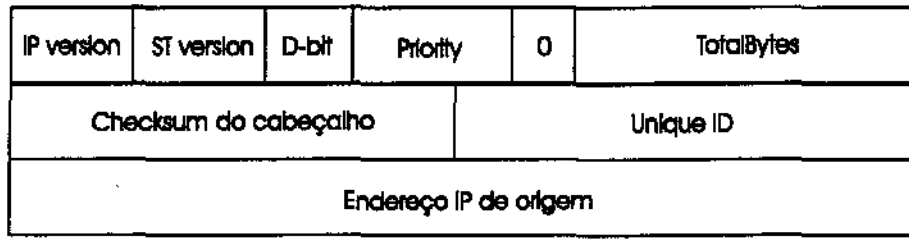


Figura 3.9: Pacote de dados ST

O campo "IP version" indica a versão de protocolo usada, onde o valor 5 representa o ST2. O campo "ST version" indica a versão de ST utilizada. O "D-bit" diferencia pacotes ST de pacotes SCMP. O campo "priority" especifica a importância relativa entre os "streams" sendo transmitidos. O campo "TotalBytes" contém o tamanho do pacote. Há também um checksum do cabeçalho. O campo "Unique ID" contém um identificador único do "stream", utilizado pelos roteadores para associar uma reserva de QoS ao "stream". Todo o cabeçalho dos pacotes ST ocupa somente 12 bytes, demonstrando a simplicidade do protocolo.

A especificação de fluxo, que contém os parâmetros de QoS a serem negociados, possui diversos formatos, cada um com um número associado, de acordo com a Tabela 3.3. As únicas especificações obrigatórias são a especificacão nula e a especificação ST2+. A especificação ST2+ é composta de cinco parâmetros: classe de QoS, precedência, tamanho da mensagem, taxa de emissão de mensagens por segundo e atraso fim-a-fim. A classe de QoS indica se a reserva de QoS deve ser previsível ou garantida. A precedência especifica a importância relativa entre as conexões sendo estabelecidas no momento. Os outros parâmetros são auto-explicativos.

A negociação da especificação de fluxo é também bastante simples. Inicialmente é feita a reserva dos recursos locais, através do gerenciador de recursos locais (LRM - Local Resource Manager). Caso aceita, a especificação é entregue ao LRM de destino. Caso aceita no destino, a especificação retorna pelo caminho definido, efetivando a reserva nos nós intermediários. Posteriormente ao estabelecimento da conexão, pode-se ajustar as reservas através de um pacote SCMP.

0	Especificação nula
1	ST versão 1
2	ST versão 1.5
3	RFC 1190
4	HeiTS
5	BerKom
6	RFC 1363
7	ST2+

Tabela 3.3: Especificações de Fluxo no ST2

3.3 Protocolos de Reserva de Recursos

A reserva de recursos é o mecanismo utilizado para garantir, estatisticamente ou deterministicamente, a alocação de recursos suficientes para uma determinado padrão de transmissão, expresso através dos parâmetros de QoS. A reserva de recursos pode acontecer em vários níveis, particularmente no nível de enlace, caso do ATM, no nível de rede, ou em protocolos de nível de transporte, caso do RSVP, que é o protocolo que será estudado em seguida.

3.3.1 RSVP - Reservation Setup Protocol

O RSVP (Reservation Setup Protocol) é um protocolo de reserva de recursos criado pela IETF, destinado a funcionar em uma internet com integração de serviços [77, 76, 74]. Seu objetivo é entregar requisições de QoS aos roteadores no caminho entre emissor e receptor, e manter o estado das conexões nos hosts e roteadores que manipulam o "stream" de dados. As requisições RSVP resultam na reserva de recursos ao longo do caminho que os dados seguirão, embora possa haver casos em que a reserva de recursos não seja necessária.

O RSVP requisita recursos para fluxos unidirecionais, isto é, os recursos são reservados em uma só direção. Deste modo, na definição do protocolo, emissor e receptor possuem papéis diferentes, embora uma aplicação possa atuar como emissor e receptor ao mesmo tempo. O RSVP é um protocolo somente de controle, a exemplo do ICMP (Internet Control Message Protocol) e IGMP (Internet Group Message Protocol), atuando no nível de transporte. O RSVP utiliza o IP como protocolo para o transporte de dados, suportando IP versões 4 e 6. O RSVP foi desenhado para suportar grandes grupos de multicast e receptores heterogêneos, com diferentes requisitos de QoS [77].

Por ser um protocolo somente de controle, o RSVP não é um protocolo de roteamento. Ele trabalha em conjunto com protocolos de roteamento, que são consultados para obter

as rotas de cada conexão, para então requisitar os recursos necessários. O RSVP suporta protocolos de roteamento unicast e multicast, particularmente aqueles utilizados correntemente na Internet. Para acomodar receptores heterogêneos, o RSVP é orientado ao receptor. Isto é, o receptor é responsável por iniciar o pedido de reserva de recursos. Desta forma, os pedidos de reserva de recursos seguem o caminho de "subida" dos dados, do receptor para o emissor, enquanto que o controle de QoS ocorre onde os dados entram na rede, isto é, seguem o caminho de "descida". Tal comportamento é mostrado na Figura 3.10.

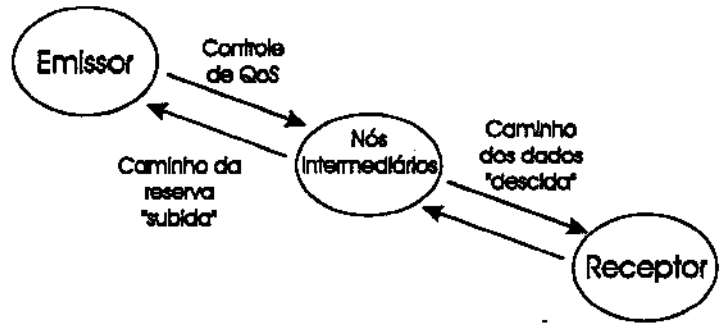


Figura 3.10: Fluxo de requisições e de controle de QoS no RSVP

A Figura 3.11 mostra a interação entre o "daemon" RSVP, o "daemon" de roteamento e os componentes do controle de tráfego nos roteadores. O classificador e o escalonador de pacotes ficam no caminho dos dados, existindo em cada host e roteador da rota. O classificador de pacotes é responsável por determinar a rota e a classe de QoS associada a cada pacote. Após serem classificados, os pacotes são encaminhados ao escalonador de pacotes, que é responsável pela alocação dos recursos de transmissão, de acordo com o protocolo de enlace presente na interface pela qual o pacote será enviado. Caso a interface tenha um protocolo com recursos de QoS, como as redes ATM, estes recursos serão utilizados. Caso seja uma mídia passiva, como linhas digitais, o próprio escalonador controla e aloca a capacidade de transmissão. O escalonador também é responsável por controlar a alocação da CPU dos hosts ou roteadores, e dos buffers de transmissão. Os "daemons" RSVP nos hosts e roteadores trocam mensagens de controle, por onde as requisições de QoS são enviadas. O "daemon" RSVP se comunica com o "daemon" de roteamento para determinar os próximos hosts/roteadores para onde devem ser enviadas as requisições de QoS. Em cada passo do caminho, o "daemon" RSVP se comunica com o módulo de controle de admissão, que verifica a viabilidade de aceitar uma nova conexão. Uma vez obtida a aprovação do controle de admissão, o "daemon" RSVP requisita ao classificador e ao escalonador de pacotes a capacidade de transmissão para aquela conexão. Caso a nova conexão não seja aprovada, uma mensagem de erro é retornada ao receptor, que é quem origina o pedido de reserva de recursos.

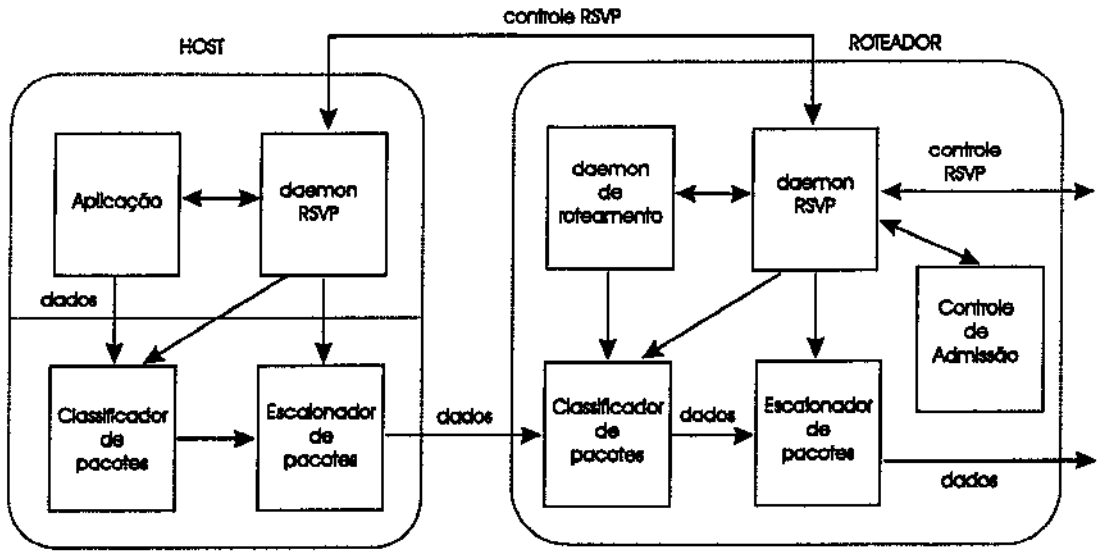


Figura 3.11: Arquitetura do RSVP

A arquitetura do RSVP foi projetada para suportar a comunicação em modo multicast para grandes grupos, como ocorre nos "broadcasts" de vídeo pela Internet. Nesses casos a quantidade de membros, sua localização e a topologia da árvore de distribuição que eles determinam muda constantemente. Para acomodar essa característica, o RSVP mantém um "soft state" nos roteadores. Isto é, as conexões e os pedidos de reserva de recursos são mantidos através do envio periódico de mensagens de "refresh" por toda a árvore de distribuição. Quando as mensagens de "refresh" não são recebidas por um roteador ou host dentro de um período de tempo, a conexão e o pedido de reserva de recursos entram em timeout e são revogados.

O controle das reservas de QoS no RSVP é baseado no conceito de sessão. Uma sessão é um fluxo de dados com um destino particular e um protocolo de transporte específico. O destino de uma sessão pode ser definido pelo endereço IP de destino, ou através de um mecanismo de portas genérico, que possibilite a multiplexação no nível de transporte. A versão atual do RSVP somente suporta portas UDP e TCP. O destino de uma sessão pode ser único ou múltiplo (unicast e multicast), assim como a origem. Assim sendo, a comunicação no RSVP pode ser um-para-um, um-para-muitos, muitos-para-muitos ou muitos-para-um, caso em que várias fontes contribuem para criar um fluxo de destino único, como uma imagem de vídeo composta de vários "quadros" com imagens vindas de locais diferentes. Uma árvore de distribuição com várias fontes e destinos é mostrada na Figura 3.12.

Uma requisição de reserva de recursos do RSVP é composta de dois elementos: uma especificação de fluxo (FlowSpec) e uma especificação de filtro (FilterSpec), formando um descritor de fluxo (flow descriptor). A especificação de fluxo descreve os parâmetros

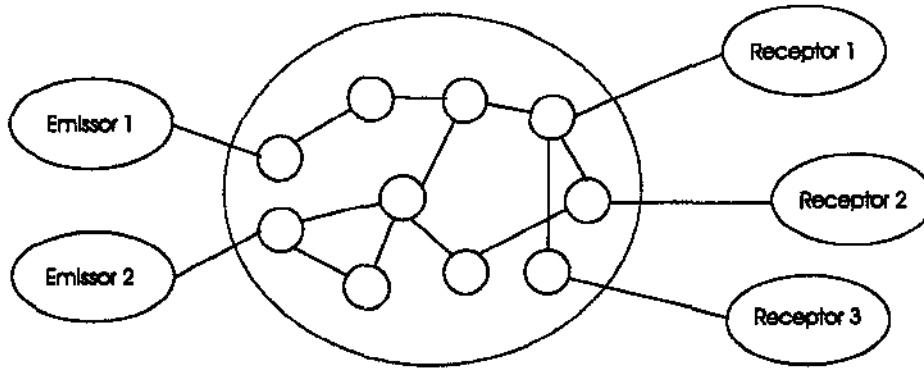


Figura 3.12: Árvore de distribuição RSVP

de QoS desejados, e é utilizada para modificar parâmetros do escalonador de pacotes. A especificação de filtro define quais pacotes de dados devem ser regidos pela especificação de fluxo, e controla os parâmetros do classificador de pacotes. Pacotes de dados pertencentes a uma sessão, e que não correspondem à especificação de filtros da sessão são tratados sem controle de QoS (best effort). O modelo de reserva de recursos do RSVP é composto de um único passo: o receptor envia o pedido, que "sobe" atravessando cada nó do caminho, onde ele pode ser aceito ou rejeitado, mas não renegociado. Como somente o resultado aceito/rejeitado é devolvido ao receptor, não há como saber o nível de QoS fim-a-fim obtido. Visando corrigir esta deficiência, o RSVP suporta o modelo OPWA (One Pass with Advertising), que consiste em enviar pacotes de controle RSVP no sentido dos dados, visando determinar a QoS fim-a-fim. Os resultados dessa varredura são entregues ao receptor, que assim pode ajustar a reserva de acordo com os resultados.

O RSVP também suporta diversos estilos de reserva de recursos, onde variam a forma de compartilhamento das reservas e a forma de determinar os emissores nos filtros da sessão. Com relação ao compartilhamento, as reservas podem ser distintas ou compartilhadas. Em uma reserva distinta, cada sessão gera um pedido de reserva de recursos, independente da árvore de distribuição. Na reserva compartilhada, caso haja um fluxo que tem a mesma origem e sofre uma "bifurcação" na árvore, os pedidos de reserva são aglutinados, e somente o maior deles é efetivado. Por sua vez, a determinação dos emissores nos filtros pode ser feita com base em coringas, que representem uma classe de emissores, ou através de uma lista explícita. A Tabela 3.4 resume os estilos de reserva permitidos pelo RSVP [77].

Por ser destinado a operar na Internet, onde tecnologias recentes e antigas coexistem, o RSVP suporta a operação transparente através de roteadores que não o suportam. Por ser baseado em IP, tanto na versão 4 quanto na versão 6, não há dificuldade em enviar pacotes RSVP através de roteadores que não o suportam. O "daemon" de roteamento informa ao "daemon" RSVP a seqüência de nós no caminho. Caso algum desses nós não

	Reserva distinta	Reserva compartilhada
Emissor explícito	Fixed-Filter (FF)	Shared-Explicit (SE)
Emissor por coringa	-	Wildcard-Filter (WF)

Tabela 3.4: Estilos de Reserva no RSVP

suporte o RSVP, o único efeito será que a mensagem de controle não será interpretada por ele, e o pacote será reenviado ao próximo nó RSVP de forma transparente.

Resumindo, as principais características do RSVP são [77]:

- Faz reserva de recursos para conexões unicast e multicast, adaptando-se a mudanças de grupo e de rotas;
- Possui mecanismo de reserva simplex, em uma só direção;
- É orientado ao receptor, que é quem inicia e mantém as requisições de fluxo;
- Mantém um "soft state" nos roteadores, mantido por mensagens periódicas de "refresh";
- Provê vários modos de reserva (estilos);
- Proporciona sua operação transparente através de roteadores que não o suportam.

3.4 Alternativas de projeto

Conforme dissemos anteriormente, um dos objetivos deste projeto foi dotar a plataforma Multiware de um protocolo de tempo real, com atributos de Qualidade de Serviço e integração com o nível de distribuição, particularmente com a arquitetura CORBA. Outro objetivo foi desenvolver um protocolo de transferência de dados (de tempo real e dados convencionais) que fosse baseado em redes ATM, e que levasse as características de QoS das redes ATM até os níveis superiores, possibilitando que o usuário do nível de transporte pudesse diretamente especificar parâmetros de QoS que seriam mapeados até as redes ATM. Outro objetivo foi que a implementação desenvolvida fosse baseada em um protocolo padronizado e de ampla aceitação, visando ampliar suas possibilidades de uso.

Diante desses objetivos, os protocolos acima apresentados foram analisados, tomando-se também por base análises feitas em outros trabalhos [48, 66]. Um dos principais requisitos foi que o protocolo deveria possibilitar o transporte de dados de usuário. Por este motivo, o protocolo RSVP foi descartado. Obviamente haveria a alternativa de

utilizar o RSVP para a reserva de recursos e outro protocolo para a transferência dos dados. Porém esta alternativa introduziria complexidade, por envolver interações entre dois protocolos e mais um algoritmo de roteamento.

O protocolo RTP mostrou-se pouco adequado para o projeto por duas razões. A primeira delas é o fato dele ser orientado a datagramas, não possuindo circuitos virtuais. Conforme dissemos anteriormente, um dos requisitos seria que o protocolo deveria suportar tanto aplicações convencionais quanto aplicações de tempo real. Muitas aplicações convencionais necessitam de um modo de conexão virtual, principalmente aquelas que não suportam a perda de dados, nem a chegada de dados fora de ordem, dois requisitos que não são tratados pelo RTP. Outro motivo para descartar-se o RTP é que ele não possui nenhum mecanismo de reserva de recursos nem de garantia de QoS, que são essenciais ao projeto. Havia a alternativa de introduzir essas facilidades no protocolo, porém o paradigma no qual o RTP é baseado (datagramas) dificulta em muito a inclusão de mecanismos de reserva de recursos.

Descartados RSVP e RTP, tanto o ST2 quanto o XTP apareceram como alternativas adequadas ao projeto. Ambos são baseados em conexões e possuem requisitos de Qualidade de Serviço. O ST2 possuia a vantagem de já incluir um mecanismo de reserva de recursos que poderia funcionar acima de redes ATM, mais especificamente, acima de AAL5. Uma desvantagem do ST2 seria o fato de operar no nível de rede. Embora o nível de rede seja o mais adequado para lidar com a reserva de recursos, este nível não possibilita um nível semântico adequado para a interação entre emissor e receptor, que é necessária para um controle mais estrito dos parâmetros de QoS. Há também o fato de que, quanto mais baixo o nível em que opera o protocolo, mais camadas e mais funcionalidades devem ser acrescentadas até torná-lo adequado à comunicação em plataformas e arquiteturas distribuídas, como em CORBA.

O fato de maior peso a favor do XTP foi a sua extrema flexibilidade. Por suportar diversos paradigmas de comunicação, diversos estilos de interação, vários mecanismos de controle de erros, de fluxo e de taxa, o XTP pode adequar-se à tranmissão de dados de tempo real ou de aplicações convencionais. Por operar no nível de transporte, o protocolo XTP pode introduzir mais controles nos dados, adequando-os às características de tempo real necessárias no projeto. A negociação de parâmetros de QoS no XTP também mostrou-se mais adequada, por incluir a modificação dos parâmetros de QoS durante a negociação, e por suportar vários conjuntos de parâmetros. Embora não possua reserva de recursos diretamente, o XTP pode ser modificado para tal fim. Outro ponto positivo foi a sua grande disseminação, e a existência de um órgão de padronização exclusivo para o protocolo (XTP Forum). Por estas e outras razões, o protocolo XTP foi escolhido como base para o modelo e para a implementação do projeto.

Capítulo 4

O Modelo ODP, a arquitetura CORBA e a Plataforma Multiware

Nos últimos anos tem sido visto um aumento contínuo no uso de sistemas de computação distribuída. Isto acontece por causa da grande demanda de interligação de sistemas de processamento de informações, e por causa da própria natureza distribuída da informação. O que está sendo feito é a adequação dos sistemas de informação às características distribuídas da própria informação.

Neste capítulo é descrito o modelo ODP, que endereça as novas exigências de sistemas abertos, e a arquitetura CORBA, que é um padrão muito disseminado para a implementação de objetos distribuídos. Posteriormente estuda-se a plataforma Multiware, que é baseada no modelo ODP e na arquitetura CORBA, e é a base onde está localizado o modelo de QoS proposto nesta dissertação.

4.1 O Modelo ODP

O RM-ODP (Reference Model for Open Distributed Processing) é um modelo de computação distribuída aberta, cujo objetivo é o desenvolvimento de padrões que permitam a distribuição dos serviços de processamento de informações em um ambiente de recursos heterogêneos, em múltiplos domínios organizacionais [34, 35, 36]. Isto implica na criação de componentes de infra-estrutura e funções que solucionem os problemas na criação e programação de sistemas distribuídos.

De acordo com o modelo ODP, os sistemas distribuídos exibem as seguintes características:

Distribuição geográfica Freqüentemente os sistemas de informação são separados por grandes distâncias, o que divide as interações entre remotas e locais;

Concorrência Todos os componentes de um sistema distribuído devem ter a propriedade de poderem ser executados em paralelo;

Falta de estado global Um sistema com diversos componentes em estados diversos impede que se tenha conhecimento de todos os estados em um dado instante;

Falhas parciais Qualquer componente de um sistema distribuído pode falhar, independentemente dos demais. O sistema como um todo deve continuar funcionando, ainda que com sua funcionalidade reduzida.

Além dessas características, se assumirmos que os sistemas distribuídos podem conter várias organizações e várias tecnologias, temos as seguintes características adicionais:

Heterogeneidade Sistemas distribuídos são construídos usando várias tecnologias, que podem ser acrescentadas ou modificadas de forma independente. Essa heterogeneidade pode aparecer em diversos componentes, como: hardware, sistema operacional, protocolos de comunicação, linguagens de programação, etc.

Autonomia Um sistema distribuído normalmente não possui um controle unificado, podendo estar espalhado por diversos domínios organizacionais, cada um com seu controle próprio. O grau de autonomia pode ser diversificado, de acordo com o sistema.

Evolução Sistemas distribuídos são dinâmicos, isto é, evoluem de acordo com as mudanças de tecnologia, de política, etc. Essas mudanças podem ser transparentes ou não, dependendo da forma como afetam o sistema.

Mobilidade Vários elementos de um sistema distribuído podem ser móveis, como os nós, os provedores de informação, os sistemas, etc. Essa mobilidade pode ser motivada por fatores de desempenho, por características do sistema, ou pela própria mobilidade dos usuários.

Para lidar com todas essas características, o modelo ODP objetiva a construção de sistemas que tenham as seguintes propriedades [36]:

Abertura Significa a possibilidade de prover portabilidade e interoperabilidade aos sistemas. A portabilidade implica que um componente de um sistema aberto possa ser executado em diversos ambientes sem modificação. Esses componentes podem ser: máquinas, sistemas operacionais, ambientes gráficos, etc. A interoperabilidade implica que o sistema suporte interações entre ambientes com componentes heterogêneos, como: protocolos de comunicação, linguagens de programação e representação de dados.

Integração Significa a possibilidade de integrar sistemas e recursos em um componente único de forma simples. Isso implica na criação de sistemas integrando diferentes arquiteturas, por exemplo.

Flexibilidade Implica na capacidade de suportar o ciclo de evolução de um sistema, incluindo manutenção e reconfiguração.

Modularidade É a capacidade de desenvolver sistemas com uma organização estrutural, de forma que seus elementos sejam autônomos, porém inter-relacionados. A modularidade é essencial para suportar a flexibilidade.

Federação É a capacidade de combinar sistemas de diferentes domínios técnicos e administrativos para alcançar um objetivo comum.

Gerência É a propriedade de monitorar, controlar e gerenciar os recursos do sistema, para proporcionar o controle de políticas, por exemplo.

Provisão de QoS É a propriedade de cumprir requisitos de qualidade sobre o comportamento de um sistema. A QoS é discutida em maiores detalhes no capítulo 2.

Segurança Um sistema ODP deve prover requisitos de segurança, da mesma forma que aplicações comuns. Porém, no modelo ODP, a segurança possui dificuldades maiores, devido, por exemplo, à mobilidade de seus componentes.

Transparência É a propriedade de mascarar para as aplicações os detalhes e as diferenças usadas para resolver os problemas causados pela distribuição. Entre esses aspectos estão a heterogeneidade e a localização dos componentes. As transparências serão discutidas ainda neste capítulo.

4.1.1 Pontos-de-vista

No modelo ODP, os pontos-de-vista são diferentes visões de um sistema, cada uma contendo diferentes interpretações dos requisitos do sistema. Essa divisão foi motivada pelo fato de que a especificação completa de qualquer sistema distribuído de porte razoável é uma atividade extremamente complexa, que teria de ser dividida de alguma forma. A divisão em pontos-de-vista facilita a especificação de sistemas distribuídos abertos [69, 56].

Cada ponto-de-vista é uma abstração que incorpora a especificação do sistema inteiro. Os pontos-de-vista não são camadas de uma arquitetura, mas sim projeções completas de um sistema. Cada ponto-de-vista tem uma linguagem associada, que expressa conceitos e regras relevantes a uma área particular. Os cinco pontos-de-vista definidos pelo modelo ODP são [36]:

Ponto-de-Vista de Empresa Lida com os aspectos gerenciais, expressando os objetivos e políticas de um sistema.

Ponto-de-Vista de Informação Lida com a semântica da informação e com o processamento de informação.

Ponto-de-Vista Computacional Trata da decomposição funcional do sistema em objetos que interagem através de interfaces.

Ponto-de-Vista de Engenharia Trata dos mecanismos e funções necessários para suportar interações distribuídas entre os objetos.

Ponto-de-Vista Tecnológico Trata das escolhas tecnológicas do sistema.

Para representar um sistema ODP sob um ponto-de-vista, é preciso definir um conjunto estruturado de conceitos a partir dos quais a especificação possa ser representada. Esses conceitos formam uma linguagem específica de cada ponto-de-vista. Neste capítulo citaremos alguns conceitos, particularmente nos pontos-de-vista computacional e de engenharia.

4.1.2 Transparências

Outro mecanismo usado para facilitar a especificação e o desenvolvimento de aplicações distribuídas no modelo ODP é o uso de transparências. Uma transparência é a habilidade que um sistema tem de mascarar certas dificuldades inerentes aos sistemas distribuídos. As transparências definidas pelo modelo ODP são [36]:

Acesso Mascara diferenças na representação de dados e nos mecanismos de invocação, para possibilitar interações entre sistemas heterogêneos.

Falha Mascara a falha e a possível recuperação de objetos, com o objetivo de prover tolerância a falhas.

Localização Mascara informações sobre a localização espacial de objetos na hora do "binding".

Migração Mascara as mudanças de localização de um objeto, que são feitas com o objetivo de fazer balanceamento de carga, por exemplo.

Relocação Mascara a relocação de uma interface e das interfaces a ela ligadas.

Replicação Mascara o uso de um grupo de objetos "iguais" (compatibilidade comportamental) para realizar uma determinada função, com o objetivo de aumentar o desempenho e a disponibilidade.

Persistência Mascara a ativação e desativação de objetos.

Transação Mascara a coordenação de atividades entre um conjunto de objetos, visando obter consistência.

4.1.3 Conceitos ODP

Nesta seção são explicados alguns conceitos básicos do modelo ODP, que são essenciais para a entendimento deste capítulo. São citados também alguns conceitos relacionados com a Qualidade de Serviço e com o contexto no qual se encaixa a plataforma Multiware.

Objetos

O conceito de objeto no modelo ODP é bastante genérico, de forma a comportar as diversas interpretações presentes nas linguagens de programação e nos ambientes distribuídos. No modelo ODP, objetos são entidades contendo informação e oferecendo serviços. Um objeto é caracterizado pelas suas propriedades de encapsulamento, abstração e comportamento. Do ponto-de-vista de um objeto, um sistema ODP consiste do próprio objeto e do seu ambiente, que é formado por todos os outros objetos [69, 56].

O modelo ODP não restringe a granularidade de um objeto, deixando isto a cargo do desenvolvedor do sistema. O comportamento interno de um objeto também não é restrito, desde que sua interface seja clara, de forma a encapsular esse comportamento. As interações entre objetos podem ser de vários tipos, por exemplo: síncronas, assíncronas, unicast, multicast, etc.

Encapsulamento e abstração

A abstração é a característica que permite aos objetos esconder seus aspectos internos dos demais objetos. O encapsulamento é a propriedade que determina que a informação contida em um objeto seja acessível somente através de sua interface. Isto garante que as mudanças de estado de um objeto serão provocadas somente através de mudanças em seu estado interno ou através de interações em sua interface. Isto é, não existem "efeitos colaterais" provocados por mudanças no estado interno de outros objetos.

Comportamento e estado

O comportamento de um objeto é caracterizado pelo conjunto de possíveis mudanças de estado que podem acontecer, e pode ser definido como o conjunto de todas as ações nas quais o objeto pode tomar parte [36]. O estado é exatamente a condição atual de um objeto. Como descrito na teoria de máquinas de estado finito (finite state machines), o estado de um objeto, somado ao comportamento que o ambiente impõe ao objeto, define o conjunto de possíveis estados futuros.

Os conceitos de comportamento e estado levam ao conceito de compatibilidade comportamental. Dois objetos possuem compatibilidade comportamental quando um deles pode substituir o outro sem que o ambiente observe qualquer mudança. Este conceito é muito útil em vários aspectos práticos do modelo ODP, por exemplo para suportar tolerância a falhas através de replicação, ou para o suporte a grupos genéricos.

Interfaces

O modelo ODP determina que um objeto pode ter várias interfaces, onde cada interface representa uma parte do comportamento do objeto relacionada com um subconjunto particular das possíveis interações. O suporte a várias interfaces é útil quando se pensa nos diversos papéis que um objeto pode ter. Por exemplo, um objeto pode ter uma interface de gerência e uma interface de usuário. Ou ainda, um objeto pode modificar sua interface de acordo com as mudanças de política a que está sujeito.

4.1.4 Linguagens ODP

Conforme explicado anteriormente, a cada ponto do modelo ODP corresponde uma linguagem, que descreve os conceitos e as regras de estruturação necessárias para especificar em detalhes aquele ponto-de-vista. No contexto desta dissertação, são particularmente importantes os pontos-de-vista de computação e de engenharia. Por isto, descreveremos os aspectos mais importantes dessas linguagens. As linguagens de empresa, informação e tecnológica podem ser consultadas nas referências [34, 36].

Linguagem de Computação

Uma especificação computacional define a decomposição funcional de um sistema ODP em objetos que interagem através de interfaces. No ponto-de-vista computacional, as aplicações e funções ODP consistem de configurações de objetos computacionais interativos.

Para melhor compreender a linguagem de computação é preciso inicialmente definir alguns conceitos. Um "sinal" é uma ação atômica que resulta em uma comunicação unidirecional entre um objeto iniciador e um objeto respondedor. Uma "operação" é uma interação entre um objeto cliente e um objeto servidor, que pode ser um anúncio ou uma interrogação. Um "anúncio" é uma invocação, iniciada por um objeto cliente, resultando na transferência de informação do objeto cliente para o objeto servidor na forma de uma requisição de função a ser efetuada pelo objeto servidor. Uma "interrogação" é composta por uma invocação, da mesma forma que no anúncio, seguida de uma segunda interação, chamada terminação, no sentido contrário, do objeto servidor para o objeto cliente, como resposta à interrogação. Um "fluxo" é a abstração de uma seqüência de interações, resultando na transferência de informação entre um objeto produtor e um

objeto consumidor. Uma interface de sinal é uma interface onde todas as interações são feitas na forma de sinais. De forma análoga podemos definir uma interface de operação. Uma interface "stream" é uma interface onde todas as operações são fluxos.

Regras de Ligação

A linguagem de computação é baseada na ligação (binding) de interfaces. Conforme explicado anteriormente, há três formas de ligação de interfaces: através de interfaces de sinal, de "stream" e de operação. Os três tipos de interface podem ser definidos através de sinais, que são a menor unidade de interação possível.

Além da categorização acima, as ligações podem também ser divididas em ligações implícitas (implicit binding) e ligações explícitas (explicit binding). Uma ligação implícita ocorre sempre que um objeto cliente referencia uma interface de operação no servidor sem que haja uma ligação anterior entre eles. Neste caso uma ligação implícita é estabelecida, seguindo a seguinte seqüência de passos [36]:

- Uma interface de operação no cliente é criada com um tipo de assinatura complementar ao tipo da interface do servidor;
- A interface de operação do cliente é ligada à interface de operação do servidor;
- O objeto servidor é invocado usando a interface de operação criada;
- A interface do cliente é removida quando a operação termina (facultativo);

A ligação explícita pode ainda ser dividida em ligações primitivas e ligações compostas. Uma ligação primitiva liga diretamente dois objetos computacionais. Uma ligação composta (Figura 4.1) envolve dois ou mais objetos computacionais, que são ligados através de um objeto de ligação (binding object). Pode-se imaginar uma ligação composta como um conjunto de ligações simples entre um objeto computacional e o objeto de ligação.

Para que ocorra uma ligação primitiva entre interfaces é necessário que ambas interfaces sejam do mesmo tipo, que tenham assinaturas de tipos complementares e que sejam de causalidade complementar. As ligações compostas possuem os mesmos requisitos, com exceção de que passamos a nos referir ao "template" do objeto, e de que os parâmetros da interface devem ser subtipos do "template" do objeto, o que é uma generalização do caso anterior.

Linguagem de Engenharia

A linguagem de engenharia é composta de conceitos, regras e estruturas para a especificação de um sistema ODP a partir do ponto-de-vista de engenharia. Uma especificação de engenharia define os mecanismos e funções requeridas para suportar interações distribuídas entre objetos em um sistema ODP [36].

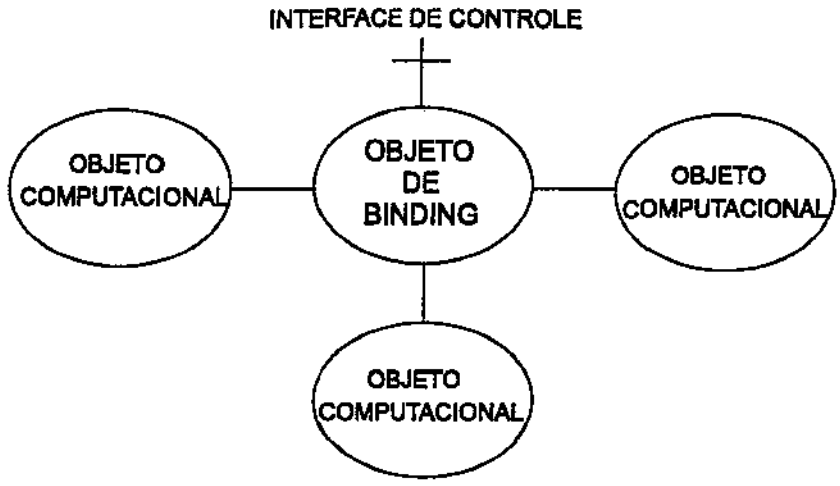


Figura 4.1: Binding Composto

Para possibilitar uma melhor compreensão desta seção, apresentaremos alguns conceitos inerentes à linguagem de engenharia.

Objeto Básico de Engenharia Um BEO (Basic Engineering Object) é um objeto de engenharia que requer o suporte de uma infra-estrutura distribuída. É o objeto da aplicação.

Cluster É uma configuração de objetos básicos de engenharia formando uma unidade com respeito a desativação, checkpoint, reativação, recuperação e migração. Um segmento de memória virtual em um sistema UNIX é um exemplo de cluster.

Gerente de Cluster É um objeto de engenharia responsável pela gerência dos Objetos Básicos de Engenharia agrupados em um cluster.

Cápsula É um conjunto de objetos de engenharia formando uma unidade com respeito ao encapsulamento de processamento e armazenamento. Uma cápsula é formada por um ou mais clusters. Um processo, tal como visto por um sistema UNIX, é um exemplo de cápsula.

Gerente de cápsula É um objeto de engenharia responsável pela gerência dos objetos de engenharia agrupados em uma cápsula.

Núcleo É um objeto de engenharia responsável pelo controle do processamento, armazenamento e funções de comunicação, disponibilizando estas funções aos objetos de engenharia subordinados ao nó do qual ele faz parte. Um kernel de sistema operacional é um exemplo de núcleo.

Nó É um conjunto de objetos de engenharia formando uma unidade de localização no espaço, o que engloba as funções de processamento, armazenamento e comunicação. Um computador, seu software básico e software de aplicação são um exemplo de um nó.

Canal É uma configuração de objetos de engenharia (stubs, binders, objetos de protocolo e interceptadores), com o objetivo de prover um conjunto de interfaces que propicie interações entre Objetos Básicos de Engenharia em diferentes clusters.

Stub É um objeto de engenharia responsável por interpretar as interações gerenciadas pelo canal e efetuar as transformações necessárias na sintaxe dos dados.

Binder É um objeto de engenharia responsável por manter uma ligação distribuída entre Objetos Básicos de Engenharia. Esta ligação pode ser simples, onde somente dois objetos de engenharia interagem, ou composto, onde vários objetos podem interagir.

Interceptador É um objeto de engenharia residente na fronteira entre dois domínios, com a função de monitorar ou assegurar o cumprimento de políticas, ou ainda de transformar os dados com o fim de adaptá-los às mudanças de política entre domínios.

Objeto de Protocolo É um objeto de engenharia que se comunica com outros objetos de protocolo do mesmo canal, com o objetivo de manter a interação entre Objetos Básicos de Engenharia através de um ou vários protocolos de comunicação.

Checkpoint É um gabarito de um objeto, juntamente com seu estado e estrutura, que pode ser usado para instanciar outro objeto de engenharia com as mesmas características e o mesmo estado que o objeto original no momento do checkpoint.

Desativação Consiste na criação de um checkpoint de cluster, seguida pela remoção do cluster.

Clonagem Consiste na instanciação de um cluster a partir de um checkpoint.

Recuperação Consiste na clonagem de um cluster após sua falha ou remoção.

Reativação Consiste na clonagem de um cluster após a sua desativação.

Migração É a tarefa de mover um cluster para uma cápsula diferente.

Canais de comunicação

O canal é o elemento básico do modelo ODP para suportar interações distribuídas e transparentes entre objetos de engenharia. Esta interação pode ser de qualquer dos

tipos citados anteriormente (operação, sinal ou fluxo), e pode consistir na transferência de dados, gabaritos de cluster ou ainda referências de interfaces de engenharia.

Um canal é uma configuração de stubs, binders, objetos de protocolos e interceptadores interconectando um conjunto de objetos de engenharia. O arranjo de um canal é mostrado na Figura 4.2, e pode ser definido como um grafo acíclico com os objetos stubs como vértices mais externos. Cada caminho entre objetos stubs consiste na sequência: stub, binder, objeto de protocolo, binder e stub, ou ainda: stub, binder, objeto de protocolo, interceptador, objeto de protocolo, binder e stub. Um canal pode ainda possuir múltiplos pontos de conexão (para tráfego multicast, por exemplo), controlados por vários objetos de protocolo.

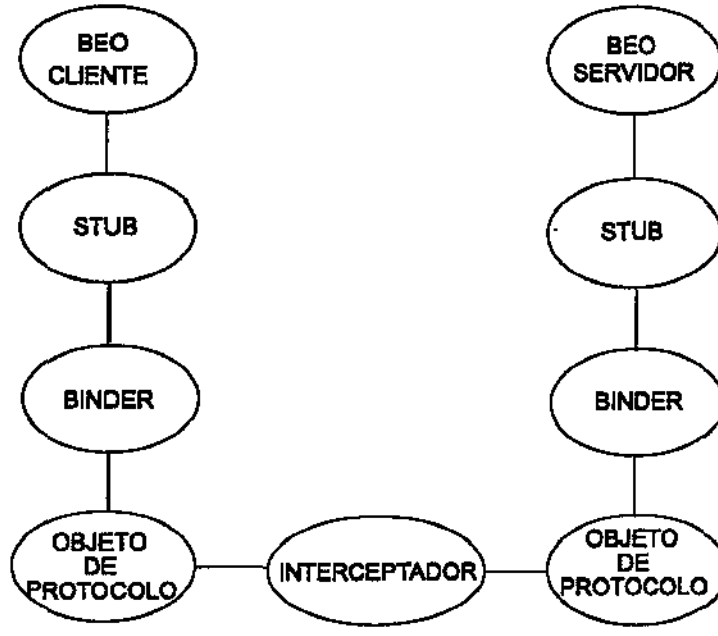


Figura 4.2: Arquitetura do Canal

As tarefas de configuração do canal e de controle da Qualidade de Serviço são reguladas através de interfaces de controle que podem estar presentes em todos os objetos mostrados na Figura, embora sua presença seja opcional, e dependente das necessidades específicas de cada interação.

Stubs

O objetivo básico dos stubs é prover a conversão dos dados transferidos através das interações. Todos os Objetos Básicos de Engenharia são ligados localmente a stubs, que podem registrar os dados que passam através do canal, provendo um mecanismo de contabilidade, ou ainda podem aplicar controles, criando um mecanismo de segurança. Os stubs podem possuir quatro tipos de interfaces: uma interface com os objetos básicos de engenharia a ele ligados, uma interface com o binder, uma interface opcional de controle,

e ainda um quarto tipo de interface, por onde ele pode interagir com objetos externos ao canal, por exemplo contactando um "cartório eletrônico" para realizar funções de autenticação. Quando é estabelecido um canal onde os Objetos de Protocolo são diferentes, a tradução é feita pelo objeto interceptador, que tem de existir neste caso. Uma interação através de um stub deve incluir um identificador e um tipo, que serão usados pelo stub para garantir que os dados a serem transferidos são compatíveis com o tipo de canal sendo utilizado.

Binders

Os objetos binder são responsáveis por gerenciar a integridade do canal fim-a-fim. Os objetos binder podem também prover a transparência de relocação. Para isto eles podem monitorar as falhas de comunicação no canal e recriar as ligações que forem interrompidas por erros. Da mesma forma que os stubs, os objetos binder podem ter quatro tipos de interfaces, dos quais dois são obrigatórios: uma interface com o stub, e uma ou mais interfaces com os objetos de protocolo. Além dessas, ele pode ter uma interface de controle, conforme descrito anteriormente, e uma interface para comunicação com objetos externos ao canal, que seria necessária para prover a transparência de relocação citada. A interface de controle, além de prover o gerenciamento de QoS, pode modificar a configuração ou remover integralmente o canal.

Objetos de Protocolo

Os objetos de protocolo disponibilizam funções de comunicação para os objetos de engenharia. Cada objeto de protocolo possui um conjunto de regras que o descrevem, como o protocolo TCP/IP, por exemplo. Novamente, há quatro tipos de interfaces possíveis para o objeto de protocolo: uma interface com o objeto binder, uma interface com outros objetos de protocolo, que deve também suportar a interação com interceptadores, uma interface de controle e uma interface para comunicação com objetos externos ao canal. No caso do objeto de protocolo, esses objetos externos podem ser serviços de diretório, por exemplo.

Quando os objetos de protocolo usados em uma interação são diferentes, é necessário que haja um interceptador para fazer as traduções de protocolo necessárias. Um interceptador pode também ser necessário no caso de os protocolos serem iguais, porém pertencendo a domínios diferentes. Neste caso pode haver um conflito de nomeação, que seria resolvido pelo interceptador.

Interceptadores

Um interceptador é um objeto de engenharia colocado na fronteira entre domínios, destinado a prover checagens e transformações nas interações que atravessam essa fronteira. Um interceptador deve ter obrigatoriamente uma interface com cada objeto de protocolo ao qual esteja ligado, além de uma interface optativa de controle. Um interceptador precisa conhecer as assinaturas dos Objetos Básicos de Engenharia ligados ao

canal, de forma que ele possa saber quais os tipos de interação suportados pelo canal.

Estruturação de objetos de engenharia

Nesta seção mostraremos as principais regras de estruturação dos objetos de engenharia, e também as principais interfaces existentes entre clusters, cápsulas, nós, seus respectivos objetos de gerência e os objetos do canal de comunicação.

Regras de Cluster

Um cluster é composto por um conjunto de Objetos Básicos de Engenharia, que estão associados a um gerente de cluster, ao qual estão ligadas as interfaces de gerência dos BEOs. Além desta ligação, todos os BEOs são também ligados ao núcleo do nó, através de uma interface que provê a função de gerenciamento do nó. Há também ligações entre os BEOs e os canais de comunicação. Essa estrutura é mostrada na Figura 4.3.

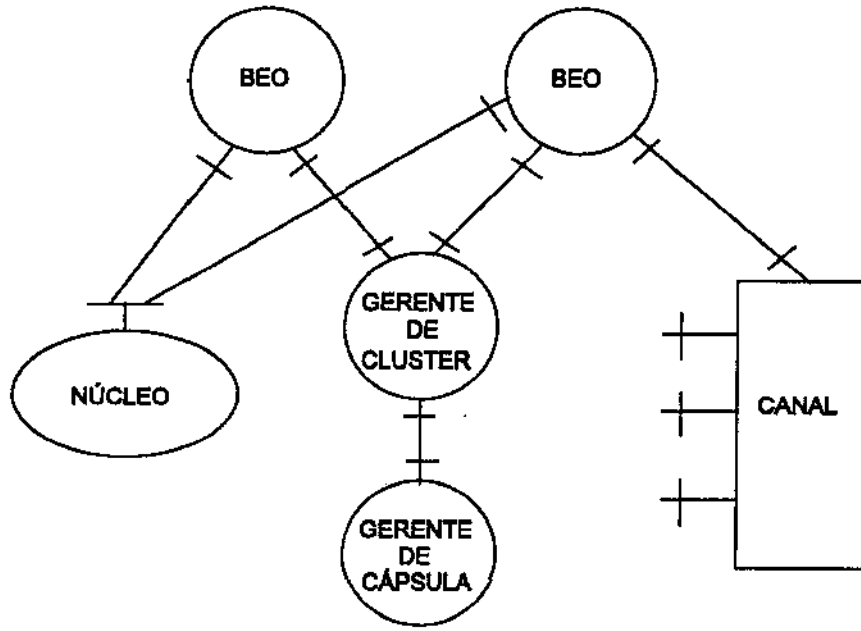


Figura 4.3: Modelo de Cluster

O gerente de cluster é um objeto ou conjunto de objetos responsável pela função de gerenciamento, e também por manter a política de gerenciamento para os BEOs sob seu controle. O gerente de cluster pode também interagir com o gerente de cápsula ou com objetos externos para fazer sua função.

Um cluster está sempre inteiramente contido em uma única cápsula. Esta restrição deve-se em parte à necessidade de prover funções de segurança, e também às dificuldades de gerenciamento. As funções de segurança de um cluster podem ser providas por um objeto localizado no mesmo cluster, ou ainda por uma função externa, através de interações seguras.

A comunicação entre BEOs localizados no mesmo cluster pode ser feita através de uma ligação local, por exemplo, através de memória compartilhada. A comunicação entre BEOs localizados em clusters diferentes deve ser obrigatoriamente feita através de um canal ODP. Embora a comunicação entre BEOs no mesmo cluster não utilize um canal, os identificadores de comunicação (binding endpoint identifier) devem ser do mesmo tipo daqueles utilizados em um canal, de forma que, caso haja a migração de um BEO para outro cluster, isto seja feito de forma transparente.

Regras de Cápsula

Uma cápsula é a estrutura imediamente superior ao cluster. Uma cápsula é composta de clusters, gerentes de cluster e de um gerente de cápsula. A forma genérica de uma cápsula é mostrada na Figura 4.4.

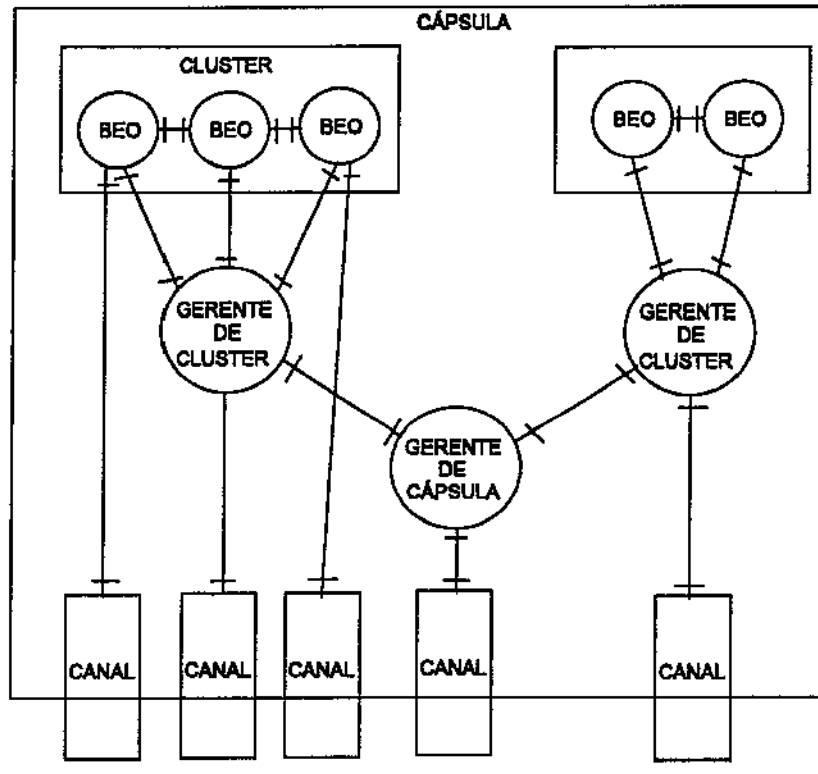


Figura 4.4: Modelo de Cápsula

Todos os BEOs contidos em uma cápsula são ligados a uma mesma interface de gerência de nó, enquanto que todos os BEOs dentro dos clusters são ligados ao seus respectivos gerentes de cluster. Todos os gerentes de cluster são ligados ao gerente de cápsula. Os canais de comunicação ligados a BEOs contidos em uma cápsula também fazem parte da cápsula, conforme mostrado na Figura 4.4. A instanciação de cápsulas é feita pelo núcleo do nó, utilizando um "template" que especifica uma configuração inicial de clusters, gerentes de cluster, gerente de cápsula e dos objetos dos canais de comunicação.

O gerente de cápsula provê a função de gerenciamento e o controle da política para os clusters contidos na cápsula. De forma análoga ao gerente de cluster, o gerente de cápsula pode ter interfaces com objetos internos para a implementação da política especificada. As estruturas que controlam as interações entre os gerentes de cluster, o gerente de cápsula e o núcleo não são especificadas pelo modelo ODP, sendo classificadas como detalhes de implementação.

Regras de nó

Conforme explicado anteriormente, um nó consiste de um núcleo e de um conjunto de cápsulas. Pode-se comparar um nó a uma estação de trabalho, um núcleo ao sistema operacional, e as diversas cápsulas às máquinas virtuais ou processos. Todos os BEOs em um nó compartilham as capacidades de processamento, de armazenamento e de comunicação presentes no nó. A estrutura de um nó é mostrada na Figura 4.5.

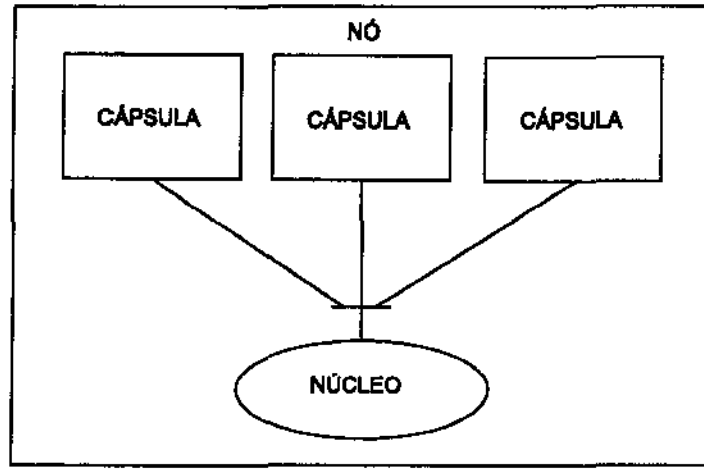


Figura 4.5: Modelo de Nó

O núcleo de um nó é responsável pela função de gerenciamento de nó e pela aplicação das políticas definidas. Cada cápsula de um nó pode ter um conjunto de políticas diferentes, e todos os BEOs contidos em uma mesma cápsula estão sujeitos à mesma política. De acordo com a política escolhida, pode-se ter ligações entre o núcleo e outras funções ODP para que a política possa ser aplicada.

Os procedimentos exatos envolvidos na criação (instanciação) de um nó não são definidos pelo modelo ODP. Porém, o modelo diz que esses procedimento devem incluir a alocação de processamento, armazenamento e funções de comunicação, bem como a iniciação da função de gerência de nó, para possibilitar a criação de ligações distribuídas. Além disso, a iniciação do nó deve alocar os canais de comunicação. O conjunto de protocolos de comunicação criados nesta etapa determina o conjunto de domínios de comunicação do qual o nó faz parte.

4.2 Arquitetura CORBA

Como resultado da crescente necessidade do uso de computação orientada a objetos e de computação distribuída, desenvolvedores de software, fabricantes de computadores e usuários uniram-se em torno de um consórcio internacional, com o objetivo de descrever padrões de computação distribuída orientada a objetos, o OMG (Object Management Group). O OMG possui hoje quase 800 membros, sendo atualmente o maior grupo de padronização da área. O objetivo principal do OMG é criar especificações de gerência de objetos, de forma que haja um ambiente de trabalho padronizado para o desenvolvimento de aplicações, obtendo com isto maior reusabilidade, portabilidade e interoperabilidade em ambientes abertos, orientados a objeto, distribuídos e heterogêneos [51, 70].

Como resultado desse esforço, foi criada a OMA (Object Management Architecture). A OMA é uma arquitetura de referência, na qual estão baseadas todas as outras especificações do OMG. A OMA descreve os objetivos, a terminologia e os modelos conceituais em que se baseia o projeto. Além disso, OMA descreve uma arquitetura concreta de comunicação entre objetos distribuídos, que é o núcleo no qual são agregadas todas as outras funcionalidades. Esta arquitetura é conhecida como CORBA (Common Object Request Broker Architecture) [51].

A arquitetura CORBA descreve os mecanismos necessários para possibilitar a comunicação e a integração entre sistemas distribuídos baseados em objetos. Fazendo uma analogia com o hardware, CORBA seria um "barramento" distribuído de objetos. Tal modelo é baseado em clientes e implementações de objetos interagindo através de um núcleo chamado ORB. A Figura 4.6 mostra a interação entre um cliente e uma implementação de objeto através do ORB.

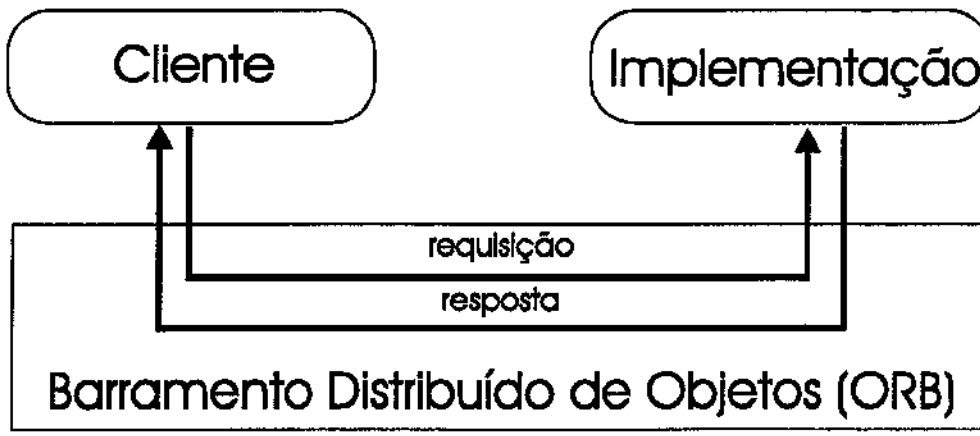


Figura 4.6: Visão simplificada de CORBA

Neste modelo, o cliente é a entidade que deseja fazer uma operação em um objeto, e a implementação contém o código e os dados que implementam o objeto. Esses dois ele-

mentos podem estar, e geralmente estão, em computadores distintos. A função do ORB é implementar os mecanismos necessários para encontrar a implementação de objeto associada à requisição, fazer com que a requisição trafegue na rede, preparar a implementação de objeto para receber a requisição, entregar os dados à implementação no formato adequado, colher os resultados e devolvê-los ao cliente. Tudo isto deve ser feito de forma transparente ao cliente, isto é, independente da localização, da linguagem de programação, do sistema operacional e de quaisquer outros aspectos que não a própria interface do objeto [51].

Obviamente o comprometimento com todas essas transparências e a necessidade de flexibilidade introduzem uma série de complexidades em relação ao modelo simplificado mostrado acima. A arquitetura CORBA deve acomodar clientes de diversos tipos, até mesmo clientes sem noção de orientação a objetos. Deve acomodar também várias sintaxes de invocação, sendo que esta complexidade é também refletida no lado da implementação do objeto. Uma visão mais detalhada da arquitetura CORBA é mostrada na Figura 4.7.

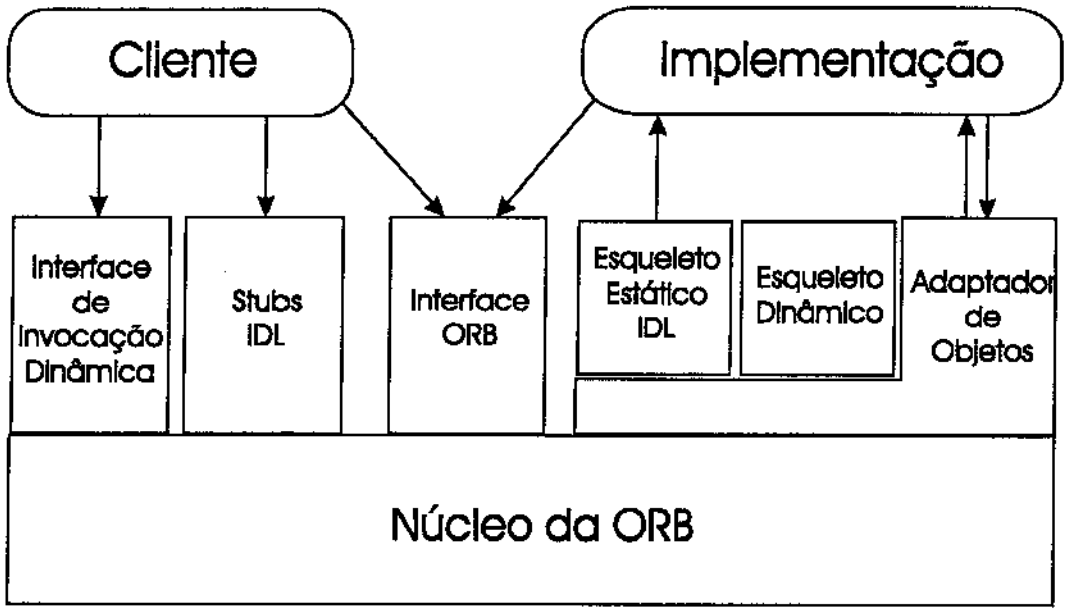


Figura 4.7: Visão detalhada de CORBA

4.2.1 Clientes

Todo o funcionamento de CORBA se inicia quando um cliente deseja invocar um objeto. O cliente enxerga o ambiente CORBA tipicamente através do mapeamento em uma linguagem de programação. Esse mapeamento é padronizado em CORBA, de forma que, para que o código de um cliente funcione em outro ambiente, basta que seja recompilado. A arquitetura CORBA define mapeamentos para linguagens orientadas a objeto, como

Smalltalk e C++, e também para linguagens procedurais, como C. O mapeamento inclui a definição de como tipos de dados simples e compostos são traduzidos, e a especificação de procedimentos para invocar objetos. No caso das linguagens orientadas a objeto, os objetos CORBA aparecem como objetos nativos da linguagem. No caso de linguagens puramente procedurais, é definida uma interface de "stubs" para fazer este mapeamento. O mapeamento define também a interação entre as chamadas ORB e os processos do cliente. Geralmente as chamadas são síncronas, isto é, o cliente é bloqueado até que o resultado da invocação seja retornado, embora possa haver uma interface adicional assíncrona.

No nível de linguagem, o cliente não conhece maiores detalhes do ambiente, como qual implementação de ORB está sendo utilizada, ou que adaptador de objeto é usado pela implementação. Tudo o que um cliente conhece sobre o objeto que deseja invocar resume-se ao que está descrito em sua interface. Para fazer uma invocação, o cliente inicialmente obtém uma referência do objeto servidor, que é utilizada em todas as invocações. Esta referência é geralmente mapeada como um ponteiro "opaco", podendo ser convertida para um formato "armazenável", como um tipo "string", e reconvertida para um ponteiro no momento da invocação.

A partir desse ponto há duas formas de interagir com o ORB (invocação estática e dinâmica), que se diferenciam pela forma como os parâmetros são passados. A invocação estática é feita através de stubs, e a invocação dinâmica através da DII (Dynamic Invocation Interface). As duas formas serão descritas na seção seguinte. Utilizando stubs, a requisição é feita em uma única chamada, enquanto que na DII a requisição é construída usando-se várias chamadas em seguida.

4.2.2 Linguagem de Definição de Interface

A Linguagem de Definição de Interface (IDL - Interface Definition Language), como o próprio nome diz, é a linguagem utilizada para descrever as interfaces dos objetos, tanto no lado servidor quanto no lado cliente. Uma definição de interface escrita em IDL deve especificar todos as operações, parâmetros, estruturas e tipos de dados presentes em um objeto. Uma vez existindo uma especificação IDL de um objeto, deve haver um mapeamento da especificação para a linguagem de programação utilizada pelo cliente.

Para facilitar o aprendizado, a linguagem IDL possui as mesmas regras léxicas que C++, e sua gramática é um subconjunto de C++ acrescido de operadores específicos para descrever o mecanismo de invocação de operações. Por ser uma linguagem puramente declarativa, IDL não inclui a parte algorítmica da gramática de C++. A seguir é mostrado um exemplo de especificação IDL. Neste exemplo, é definido um objeto "círculo" que contém duas operações: le-raio e muda-raio. A operação muda-raio ilustra o uso de um parâmetro somente de entrada. Em IDL, os parâmetros podem ser somente de entrada (In), somente de saída (Out), ou de entrada e saída (Inout) [51].

```
Interface circulo{
   struct posicao{
      float x, y;
   }
float le-raio{};
void muda-raio{In float r};
```

4.2.3 Interface de Invocação Dinâmica

A Interface de Invocação Dinâmica (DII - Dynamic Invocation Interface), como o nome indica, permite a criação e a invocação de requisições de forma dinâmica. O objetivo é que um cliente possa fazer requisições sem conhecer a interface do objeto invocado em tempo de compilação. Durante a execução do programa, o cliente pode obter os nomes das operações e os parâmetros (através de um repositório de interfaces, por exemplo), e passá-los para o ORB através de uma ou várias chamadas. Do ponto-de-vista semântico, uma requisição através da DII tem o mesmo efeito que uma requisição feita através de "stubs".

4.2.4 Repositório de Interfaces

O repositório de interfaces é um serviço que oferece informações persistentes sobre interfaces em uma forma disponível em tempo de execução. O repositório de interfaces contém toda a informação codificada em IDL, que são os nomes e tipos das operações e dos parâmetros. Além dessa informação, o repositório pode conter informações sobre versões, sobre a depuração dos programas, bibliotecas de stubs, rotinas de formatação de objetos, etc. O repositório de interfaces pode ser usado pelo ORB durante sua operação, mas principalmente pelos clientes, que através dele podem conhecer a interface de um objeto e fazer requisições dinâmicas em tempo de execução.

4.2.5 Interface ORB

No ambiente CORBA existem algumas operações que são comuns a todas as implementações de ORB, e que não dependem nem do tipo de invocação feita pelo cliente, nem do tipo de adaptador de objetos utilizado na implementação. Este pequeno conjunto de funções, que são implementadas pelo próprio ORB, são disponibilizadas através da interface ORB. Um exemplo são as operações com as referências de objeto, como: converter uma referência em "string" e vice-versa, duplicar uma referência, checar a equivalência entre referências, etc. Há outras operações, como: checar a existência de um objeto, inicializar o ORB e o adaptador de objetos, etc.

4.2.6 Núcleo do ORB

O núcleo do ORB tem por principal função a comunicação entre clientes e implementações de objetos através de requisições. O núcleo do ORB recebe as requisições, realiza as transformações necessárias durante o trajeto na rede, invoca a implementação do objeto através do adaptador apropriado, coleta os resultados e os entrega ao cliente. Para realizar esta função, o núcleo do ORB pode ainda contactar outros serviços, como os repositórios de interfaces e de implementações.

A arquitetura CORBA descreve o funcionamento de um ORB somente através de suas interfaces. Isto possibilitou que vários tipos de implementações de ORB fossem criadas. Pode haver ORBs baseadas em servidor, baseadas no sistema, baseadas em bibliotecas, ou ainda implementadas nos clientes e servidores. Nos ORBs baseadas em servidor, uma ou várias máquinas centralizam todas as requisições, roteando cada requisição para a atual localização da implementação do objeto. Os ORBs baseados no sistema são implementadas pelo próprio sistema operacional, visando maior desempenho, robustez e segurança. Os ORBs baseados em biblioteca possuem o próprio código do ORB em bibliotecas do sistema, sendo adequadas para objetos leves e cuja implementação possa ser compartilhada. Finalmente, a implementação do ORB pode ser feita dentro de rotinas localizadas no cliente e na implementação dos objetos [51].

4.2.7 Adaptadores de objeto

Os adaptadores de objeto representam a principal interface entre as implementações de objetos e o ORB. Em um ambiente CORBA existe uma grande diversidade de categorias de objetos, que podem ser diferentes quanto à granularidade, estilo de implementação, tempo de vida ou política, entre outras categorias. Como exemplo dessa diversidade, podemos citar ambientes orientados a transações on-line, ambientes de computação científica, ambientes de eventos em tempo real, etc. Para acomodar esses requisitos, a arquitetura CORBA define que haja adaptadores de objetos específicos para cada um desses ambientes. Os adaptadores de objetos disponibilizam vários serviços às implementações de objetos, entre os quais estão [51]:

- Geração e interpretação de referências de objetos;
- Invocação de métodos;
- Ativação e desativação de objetos;
- Mapeamento de referências de objetos para implementações;
- Registro de implementações;

- Controle da segurança nas interações.

4.2.8 Esqueletos estáticos e dinâmicos

Os esqueletos ("skeletons") estático e dinâmico são os correspondentes do lado servidor às interfaces de stubs e de DII do cliente. São geralmente interfaces do tipo "up-call", isto é, são rotinas escritas pelas implementações de objetos, de acordo com as interfaces definidas, e que são chamadas pelo ORB na ocorrência de uma invocação remota. Essas rotinas são dependentes do mapeamento utilizado para a linguagem de programação específica, e também dependem do adaptador de objetos utilizado.

Os esqueletos podem ser específicos para uma dada interface (esqueleto estático), ou genéricos (esqueleto dinâmico). Um esqueleto estático pode ser gerado a partir de uma especificação em IDL, para uma dada linguagem, e contém um código específico para manipular chamadas que correspondem à interface descrita. O esqueleto dinâmico interage com o adaptador de objetos, trocando informações sobre a implementação e sobre a chamada, no caso da implementação de objeto não estar registrada neste adaptador de objetos.

A implementação informa ao adaptador os parâmetros das operações, que podem ser manipulados diretamente pelo adaptador. Durante a invocação, o adaptador informa, um a um, os valores dos parâmetros de chamada, e a implementação devolve os parâmetros de retorno ou as exceções levantadas.

Cabe lembrar também que qualquer combinação de tipo de chamada estática ou dinâmica pode ocorrer entre clientes e implementações. Isto é, o uso de uma chamada via DII no cliente não necessariamente implica no uso de um esqueleto dinâmico no lado servidor, o mesmo valendo para chamadas estáticas.

4.3 A Plataforma Multiware

A Multiware é uma plataforma de suporte a computação distribuída em desenvolvimento na UNICAMP [41, 42, 47]. Seu objetivo é auxiliar o processo de desenvolvimento de aplicações distribuídas orientadas a objeto. O nome "Multiware" vem da existência de diversas camadas na plataforma, envolvendo tanto o nível de "middleware" quanto camadas acima e abaixo desse nível [43]. A plataforma Multiware está sendo desenvolvida de acordo com os conceitos do modelo ODP e da arquitetura CORBA. A estrutura da Multiware é mostrada na Figura 4.8. Descreveremos a seguir cada uma das camadas da Multiware.

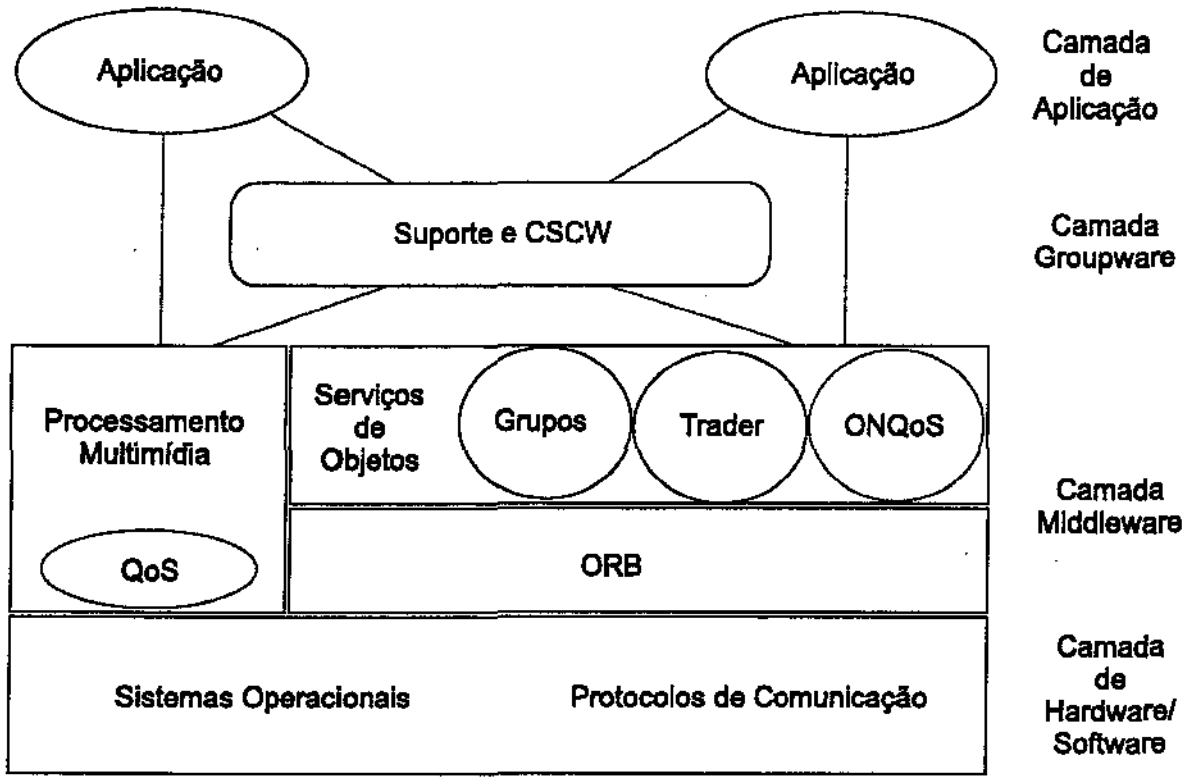


Figura 4.8: Arquitetura da Multiware

4.3.1 Camada de hardware/software

O camada de hardware/software contém os elementos de suporte a computação distribuída, como o sistema operacional e os protocolos de comunicação. Também estão incluídos aqui os protocolos baseados em chamadas remotas, como RPC (Remote Procedure Call). Esses elementos constituem a base de comunicação, acima da qual as plataformas distribuídas são construídas.

Também está incluida nesta camada a arquitetura de rede na qual a Multiware está baseada. Graças às camadas de abstração presentes nos níveis superiores, pode-se utilizar uma ampla gama de tecnologias de rede abaixo da Multiware. A única restrição apresentada é que a tecnologia escolhida consiga suportar os requisitos das aplicações multimídia suportadas pela Multiware, o que implica em utilizar-se uma rede de alta velocidade. Atualmente o protótipo da plataforma está instalado em uma rede FDDI (Fiber Distributed Data Interface), de 100 Mbps [41]. Nesta dissertação, conforme descrito no capítulo 6, foram implementados os protocolos e alguns objetos da Multiware em uma rede ATM de 155 Mbps, onde pode-se chegar a 622 Mbps ou valores superiores sem necessitar de nenhuma mudança substancial na plataforma. Outra alternativa de tecnologia para a plataforma seria Fast Ethernet comutado, onde tem-se a velocidade de 100 Mbps, que

pode ser dedicada integralmente a cada servidor, desde que se tenha um switcher para assegurar a comutação. Outra alternativa seriam redes metropolitanas baseadas no protocolo DQDB (Distributed Queue Dual Bus), que também podem chegar a velocidades superiores a 155 Mbps. Desta forma, embora não se excluam as outras tecnologias, as tecnologias de rede preferenciais para a Multiware seriam: ATM, FDDI, Fast Ethernet e DQDB.

O protocolo de rede atualmente utilizado no protótipo da Multiware é o TCP/IP, principalmente porque as implementações de CORBA nas quais a Multiware está baseada (Orbeline e Orbix) utilizam o TCP/IP como protocolo padrão. Porém, o protocolo TCP/IP apresenta sérias limitações quanto à utilização de multimídia, principalmente por não prover nenhuma garantia de Qualidade de Serviço e por não possuir requisitos de tempo real. Por estes motivos, nesta dissertação, conforme descrito no capítulo 6, propusemos o uso do protocolo XTP para o controle dos canais multimídia. A proposta é utilizar o protocolo TCP/IP durante a fase de estabelecimento de conexão e durante as chamadas distribuídas, e posteriormente utilizar o XTP para prover canais full-duplex, de tempo real e com garantias de QoS para o tráfego multimídia.

O protocolo RPC é aqui citado porque a Orbeline, que é uma das implementações de CORBA utilizadas na plataforma Multiware, é baseada em RPC. A Orbix, que é uma plataforma mais moderna, que está substituindo a Orbeline no projeto Multiware, já é inteiramente baseada em sockets TCP/IP, dispensando o uso de RPC.

Foram escolhidos dois sistemas operacionais para a plataforma: O AIX, da IBM, e o Solaris, da Sun Microsystems, ambos baseados no UNIX. A escolha de sistemas UNIX deveu-se às características de escalabilidade, estabilidade e programabilidade deste sistema operacional. Além dos serviços básicos providos pelo sistema operacional (controle de processos, acesso à rede, controle de arquivos, etc), é desejável que o sistema operacional possua características de tempo real e de múltiplas linhas de execução (threads), para suportar aplicações de tempo real, particularmente aplicações multimídia.

4.3.2 Camada Middleware

A função da camada Middleware é proporcionar às aplicações um ambiente de processamento distribuído aberto, baseado em padrões de objetos distribuídos. A camada Middleware é dividida em três subcamadas: processamento multimídia, ORB e serviços de objetos [43].

A subcamada de processamento multimídia é responsável por controlar o tráfego multimídia. Mais especificamente, são funções desta subcamada: o estabelecimento de canais multimídia, o controle da QoS durante a transmissão, a sincronização entre diferentes mídias e a apresentação de objetos multimídia como áudio e vídeo. Foi desenvolvido em [13] um protocolo para a sincronização do tráfego multimídia. Conforme dissemos

anteriormente, como parte desta dissertação desenvolvemos extenções ao protocolo XTP para o controle dos canais de tempo real e para o provimento de QoS. A etapa inicial de negociação da QoS, no entanto, é feita pelo Objeto de Negociação de QoS, que está localizado na subcamada de serviços de objetos [49], e que é explicado no capítulo 6. Conforme pode-se notar na Figura 4.8, a subcamada de processamento multimídia possui acesso direto às funções do sistema operacional, evitando-se assim atrasos que poderiam prejudicar o tráfego multimídia.

A subcamada ORB (Object Request Broker) tem por função prover abstrações do ambiente distribuído, de forma que aplicações distribuídas possam ser desenvolvidas com um mínimo conhecimento do ambiente no qual ela será executada. Essas abstrações estão descritas na OMA (Object Management Architecture), que contém a descrição da arquitetura CORBA. A arquitetura CORBA é hoje o padrão de objetos distribuídos de maior aceitação, tanto pela comunidade acadêmica quanto pelo mercado, tendo suplantado outros padrões alternativos, como ANSAware e o DCE (Distributed Computing Environment). Versões anteriores da Multiware tinham por objetivo suportar tanto CORBA quanto ANSAware e DCE. Nas versões mais recentes, optou-se por definir CORBA como o padrão de middleware. Desta forma, os objetos desenvolvidos para a Multiware, que serão comentados adiante, são baseados em CORBA, ou pelo menos possuem este tipo de interface, além de outras. Como implementação de CORBA foi escolhida inicialmente a ORBeline, da Postmodern Computing, sendo posteriormente substituida pela Orbix, da Iona Technologies [14].

Os serviços de objetos correspondem a serviços de uso genérico em aplicações distribuídas, os quais foram julgados como essenciais dentro do projeto Multiware. Alguns desses serviços de objetos já são fornecidos junto com as implementações de CORBA [49]. Alguns outros, que não existiam, foram especialmente desenvolvidos para a plataforma Multiware. Dentre eles está o Objeto de Negociação de QoS [38], implementado como parte desta dissertação, e que é explicado no capítulo 6. Além do ONQoS, foram desenvolvidos no projeto Multiware um objeto Trader [40], um Objeto de Suporte a Grupos [14, 15, 16] e um Objeto de Gerenciamento Distribuído [53].

4.3.3 Camada Groupware

A camada Groupware tem por objetivo suportar aplicações multimídia de trabalho cooperativo, do tipo CSCW (Computer Supported Cooperative Work). Utilizando os serviços de objetos (particularmente o suporte a grupos) e a subcamada de processamento multimídia, é possível construir aplicações que integrem diversos grupos em uma atividade cooperativa, através de um ambiente virtual compartilhado. Essas aplicações envolvem diversos tipos de interações, que podem ser: simultâneas no tempo e em espaços diferentes, em tempos distintos e no mesmo espaço, em tempos e espaços distintos, ou até no

mesmo tempo e espaço [42]. Exemplos dessas aplicações são: video-conferência, edição cooperativa, correio eletrônico multimídia, fluxo de trabalho (workflow), etc.

Capítulo 5

Um Modelo de QoS Multinível

O tema "Qualidade de Serviço" tem tido grande destaque atualmente. Porém, a maioria da literatura do assunto aborda QoS em um contexto específico, como o de aplicações multimídia, ou ainda em um só nível, seja ele de rede, de transporte ou mesmo de aplicação. O objetivo nesta dissertação é definir um modelo genérico, de múltiplos níveis e baseado em padrões de computação distribuída, mais especificamente, no modelo ODP e na arquitetura CORBA. Antes de aprofundar no modelo proposto, serão vistos na seção seguinte os principais aspectos de alguns modelos de QoS desenvolvidos recentemente.

5.1 Alguns Modelos de QoS

5.1.1 QosME - QoS Management Environment

O QoSME (QoS Management Environment) é um ambiente integrado que controla a QoS entregue às aplicações, fornecendo suporte à criação de aplicações adaptativas, isto é, que adaptem suas características ao nível de QoS entregue pela rede [27, 26]. Utilizando o QoSME, as aplicações podem monitorar, analisar e adaptar-se ao nível de QoS entregue pela rede. A monitoração de QoS também pode ser feita pela própria rede, visando cumprir os requisitos especificados pelas aplicações de forma fim-a-fim. As aplicações podem especificar seus requisitos de QoS, que serão monitorados no nível de transporte através de uma interface de sockets especialmente criada.

A monitoração de QoS em QoSME é feita através da interface QoSockets, que é uma interface de transporte estendida para conter atributos de QoS e monitorá-los. A interface QoSockets consegue detectar violações nos requisitos de QoS especificados e chamar funções previamente definidas pelas aplicações para tratar essas violações. A gerência de QoS em QoSME é garantida através de MIBs (Management Information Bases) que são geradas contendo informações sobre a entrega de QoS fim-a-fim. Essas MIBs podem

ser controladas por gerentes SNMP (Simple Network Management Protocol), que podem adaptar a alocação de recursos de acordo com os dados obtidos das MIBs [27, 26].

A Figura 5.1 mostra a arquitetura de QoSME. A parte principal de QoSME atua em uma camada de runtime, o que permite que as aplicações tenham acesso a um nível de abstração bem alto. Neste modelo aparecem três camadas distintas. Uma camada inferior, que cuida da interface de QoSME com os protocolos de rede e de transporte, e também com o sistema operacional. A camada intermediária é justamente onde reside a funcionalidade de QoSME para especificação, monitoração e adaptação de QoS. No nível superior estão as aplicações QoSME, que especificam seus requisitos de QoS, recebem notificações de monitoração e reagem de forma adaptativa. Descreveremos sucintamente a função de cada elemento da arquitetura.

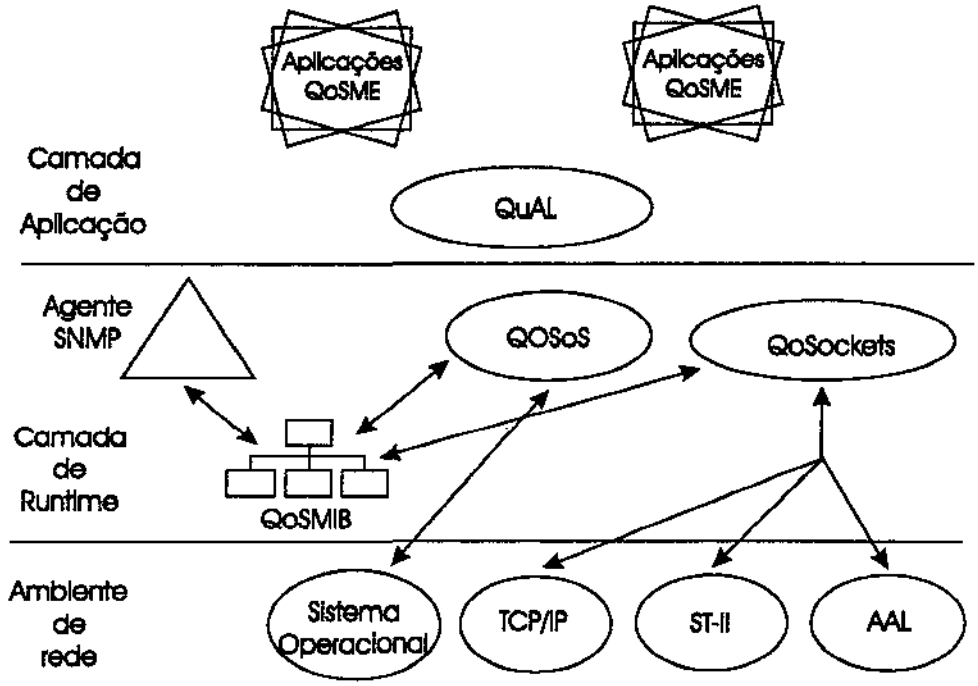


Figura 5.1: A Arquitetura QoSME

A interface QoSockets é uma extensão dos Berkeley Sockets para incluir a negociação, adaptação e gerenciamento de QoS pelas aplicações. Em QoSockets, as aplicações são abstraídas como processos que trocam informações através da passagem de mensagens. As mensagens são recebidas em "ports" de entrada (inports), e enviadas em "ports" de saída (outports). Através de QoSockets é possível associar restrições de QoS aos "ports" e fazer uma posterior negociação. Por ser um mecanismo em nível de transporte, QoSockets torna totalmente transparentes os protocolos de nível inferior, mapeando os requisitos de QoS especificados em parâmetros dos protocolos de transporte e de rede utilizados. Atualmente, QoSockets possui suporte aos protocolos TCP/IP, ST2 e AAL [27, 26].

A especificação dos requisitos de QoS pode ser feita através de uma API própria, ou ainda utilizando a linguagem QuAL, que será descrita a seguir. Para dar suporte a aplicações adaptativas, é possível associar funções manipuladoras de exceções aos eventos gerados em QoSockets. Também há uma interface de QoSockets com as MIBs SNMP, para que se possa registrar a QoS efetivamente entregue às aplicações.

A interface QoSOS faz a interface com o sistema operacional, visando adaptar os requisitos de QoS especificados pela aplicação aos serviços oferecidos pelo sistema operacional, como por exemplo a priorização e o escalonamento de processos. Esta interface também possui acesso às MIBs SNMP, onde são registrados os eventos relacionados aos eventos controlados pelo sistema operacional. Também representado na arquitetura QoSME está um agente SNMP, que interage com as MIBs. Esse agente pode, por exemplo, controlar a alocação de recursos dentro da rede baseado nos dados obtidos das MIBs.

Na camada de aplicação está a linguagem QuAL. QuAL provê um conjunto de extensões da linguagem Concert/C, de forma que o programador de aplicações possa especificar seus requisitos de QoS de forma descritiva, sendo uma alternativa de mais alto nível ao uso da API de QoSockets. Por ser uma linguagem simples, a descrição dos requisitos de QoS é facilitada. Outra vantagem é que a descrição da QoS feita pelo programador pode também ser checada em tempo de compilação. Uma vez compilados, os requisitos de QoS são traduzidos em chamadas ao sistema de runtime [27, 26].

5.1.2 QoS-A - A Quality of Service Architecture

A QoS-A é uma arquitetura de múltiplas camadas de mecanismos e serviços para o gerenciamento de QoS e o controle de fluxos de mídia contínua em redes [9, 10, 8, 6]. A base da arquitetura é a caracterização do conceito de fluxo. Um fluxo, na QoS-A, é a produção, transmissão, e consumo do tráfego de uma única mídia, que é definida por uma série de parâmetros de QoS. Os fluxos, na QoS-A, são unidirecionais, podendo ter um único ou vários destinos (unicast ou multicast). O Objetivo da QoS-A é garantir que os parâmetros de QoS de um fluxo sejam respeitados ao longo de todo o trajeto, o que abrange o controle do dispositivo de transmissão, do sistema operacional e das diversas camadas de protocolos. A arquitetura da QoS-A, mostrada na Figura 5.2, é dividida em um conjunto de camadas e planos. Daremos a seguir uma breve descrição de cada um deles.

A camada superior da QoS-A suporta o uso de plataformas de aplicações distribuídas, que incorporem funcionalidade extra à arquitetura, podendo por exemplo ser ambientes baseados em objetos. Esta camada poderia incorporar mecanismos de negociação de QoS de mais alto nível. As camadas inferiores são chamadas de camadas de orquestração, por proverem serviços de sincronização multimídia, controlando múltiplos fluxos e realizando correções, como o controle de jitter.

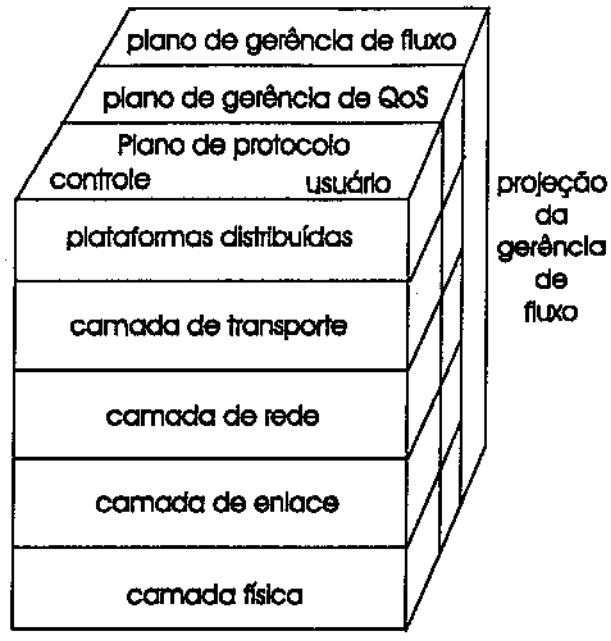


Figura 5.2: A Arquitetura QoS-A

A orquestração nas camadas inferiores é baseada em uma camada de transporte, que provê um serviço de transporte orientado a multimídia (METS - Multimedia Enhanced Transport Service), contendo protocolos de controle de QoS [9, 6]. Este serviço, que é a base de toda a arquitetura, será melhor detalhado a seguir. Imediatamente abaixo estão as camadas de rede, de enlace e física. A Figura 5.3 detalha essas camadas e sua interface com o serviço de transporte.

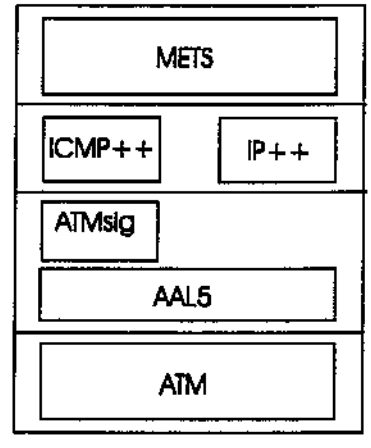


Figura 5.3: Camadas da arquitetura QoS-A

Na camada de rede aparecem os protocolos ICMP++ e IP++. O ICMP++ (Internet Control Message Protocol) é um protocolo de sinalização baseado em um subconjunto do

RSVP (vide capítulo 3), que é encapsulado no protocolo ICMP, que faz parte do conjunto de protocolos TCP/IP. O IP++ é uma versão especializada do protocolo IP, onde o cabeçalho foi modificado para incluir um identificador e uma especificação de fluxo, para que os roteadores possam alocar seus recursos a cada fluxo baseado nesses dados.

Na camada de enlace aparece o protocolo ATMSig, que é um protocolo de sinalização para redes ATM baseado em um subconjunto da UNI (User-to-Network Interface) versão 2.4. Nesta mesma camada está o protocolo AAL5, que é um protocolo simples de entrega de dados em redes ATM, a ser detalhado no capítulo 6. A função da AAL5 é prover a segmentação e remontagem de pacotes IP++ em células ATM. Na camada física está o sistema de transmissão ATM, que inclui placas de comunicação e switchers desenvolvidos pelo ORL (Olivetti Research Lab) [9, 6].

Além das camadas descritas, há também os planos verticais, que são o plano de protocolo, plano de manutenção de QoS, e o plano de gerenciamento de fluxo. O plano de protocolo, dividido em um plano de usuário e um plano de controle, é responsável pela tranmissão dos dados, observando que o plano de controle necessita de mensagens de latência baixa e bidirecionais, e que o plano de usuário necessita transmitir fluxos unidirecionais de alto desempenho. O plano de manutenção de QoS é responsável pela monitoração e manutenção dos níveis de QoS associados aos fluxos. O plano de gerência de fluxos é responsável pelo estabelecimento dos fluxos, pelo mapeamento da QoS entre camadas distintas, e pela renegociação e adaptação de QoS que podem acontecer durante a existência de um fluxo.

São citados a seguir mais alguns detalhes do serviço de transporte da QoS-A, que é a base de toda a arquitetura. O serviço de transporte orientado a multimídia (METS) é um conjunto de protocolos destinados a requisitar, negociar e contratar níveis de QoS. No METS isto é especificado através de um contrato de serviço, acordado durante o estabelecimento dos fluxos de mídia. Esse contrato suporta os parâmetros de QoS de throughput, atraso, jitter e perda de pacotes. Esses parâmetros são arranjados em uma estrutura de contrato de serviço, que possui a forma mostrada a seguir.

```
typedef struct {
   flow-spec-t   flow-spec;
   commitment-t  commitment;
   adaptation-t  adaptation;
   maintenance-t maintenance;
   connection-t  service;
   cost-t        cost;
}service-contract-t;
```

A especificação de fluxo (flow-spec) caracteriza os requisitos de tráfego do ponto-de-vista do usuário (entenda-se por usuário a entidade que usa os serviços do METS, no caso

as plataformas distribuídas). Aqui são especificados parâmetros como o tipo de mídia, a taxa de geração de quadros, a explosividade na transmissão (burstiness), o atraso, o jitter e a taxa de perda de quadros. A estrutura commitment especifica as classes de serviço suportadas pela rede, que são: determinística, estatística ou de melhor esforço (best effort). A estrutura adaptation especifica as ações a tomar no caso da degradação do nível de QoS contratado. As ações possíveis são a renegociação, a indicação da situação da comunicação ao par, ou a desconexão do fluxo. Pode-se também não tomar ação alguma. A estrutura maintenance especifica se aquele fluxo determinado será monitorado, se serão tomadas ações corretivas como vimos acima, ou se nada será feito. A estrutura service especifica o tipo de conexão que será feita, onde pode-se ter um modo de conexão negociada, um modo de conexão rápida ou ainda uma conexão com reserva prévia de QoS.

Para poder integrar as diversas camadas e planos descritos acima, a QoS-A inclui suporte ao sistema operacional. No projeto foi utilizada um versão estendida do sistema operacional Chorus, baseado em um micro-kernel [9, 6]. O Chorus foi estendido na sua API de transporte de dados, onde foram introduzidos pontos de controle para possibilitar a comunicação em tempo real e a monitoração/adaptação da QoS fim-a-fim.

5.1.3 QoS por exemplos

Vogel [37, 71, 72] descreve um modelo de QoS do ponto-de-vista de suas interfaces, enfatizando também os protocolos de negociação de QoS entre aplicações multimídia. No mesmo modelo é descrito um sistema de transporte de multimídia (MMTS - Multi-Media Transport System).

O modelo descrito por Vogel é direcionado a aplicações multimídia. Nesse contexto, são identificados três objetos principais para a negociação de QoS: a base de dados, o cliente e o sistema de transporte multimídia. As interfaces de QoS são definidas por tuplas, que são uma lista de pares de parâmetros e valores associados. É definida também uma divisão entre os tipos utilizados nessas tuplas. Objetos de tipos atômicos são aqueles que têm uma única mídia associada, como texto, imagem e vídeo, ou ainda objetos multimídia multiplexados, como áudio e vídeo nos formatos MPEG ou DVI, por exemplo. Tipos compostos são construídos através da junção de tipos atômicos, com informação adicional descrevendo a forma de sincronização entre as várias mídias atômicas.

O modelo descreve as diversas interfaces entre os componentes de um ambiente multimídia. A primeira interface descrita no modelo é a interface dos usuários com a QoS. O método aqui descrito foi denominado pelos autores de "QoS por exemplos" (Quality Query by example). O método consiste em apresentar ao usuário um conjunto de parâmetros pré-definidos, que representem experiências concretas, como "som com qualidade de telefone". O usuário pode então selecionar um conjunto de opções, sentir o resultado do conjunto, e ajustar os parâmetros novamente até obter aquilo que deseja. A Tabela 5.1

Parâmetro	Valores
Tipo de mídia	{texto, imagem, áudio, vídeo,
	audio-e-vídeo, tipo composto}
Formato de texto	{ASCII, PostScript}
Qualidade de áudio	{Telefone, CD}
Cor	{Preto-e-Branco, tons de cinza, colorido}
Tamanho	{escalável, pequeno, médio, grande}
Taxa de frames	{taxa de TV, taxa reduzida, imagens congeladas}
Delay	{segundos}
Custo	{unidade monetária}

Tabela 5.1: Interface de Usuário em QoS por exemplos

Parâmetro	Valores
Tipo de tela	{1-bit, 8-bits-cinza, 24-bits-colorida}
Dispositivo de áudio	{qualidade de telefone, qualidade de CD}
Armazenamento em disco	{bytes}
Tipo de mídia	{texto, imagem, áudio, vídeo,
	audio-e-vídeo, tipo composto}
Formatos suportados	{ASCII, PostScript, gif, MPEG, DVI}
Throughput	{frames por segundo}
Delay	{segundos}
Classe de garantia	{garantido, melhor esforço}

Tabela 5.2: Interface com o Sistema Operacional

mostra alguns parâmetros utilizados como exemplo em [72].

A segunda interface descrita é a da QoS com o sistema operacional. De acordo com o modelo, a QoS vinda do sistema operacional é dividida em qualidade de dispositivos e qualidade de software. A qualidade de dispositivos está relacionada com a forma como o sistema operacional controla dispositivos físicos do sistema, como a tela ou o alto-falante, ou ainda dispositivos utilizados para buferização dos dados multimídia, como o disco rígido. A qualidade de software está relacionada com o suporte ao formato e à saída das várias mídias, como decodificação de MPEG, formatação PostScript, etc. Os parâmetros definidos para esta interface estão descritos na Tabela 5.2.

A terceira interface apresentada é aquela existente entre a QoS e o servidor multimídia, onde está a base de dados a ser transmitida pela rede. A caracterização dos objetos

Parâmetro	Valores
Tipo de mídia	{texto, imagem, áudio, vídeo,
	audio-e-vídeo, tipo composto}
Formato	{ASCII, PostScript, JPEG, MPEG, DVI}
Tamanho	{bytes}
Throughput	{bytes por segundo}
Delay	{segundos}
Tamanho do pacote	{bytes ou variável}
Jitter	{segundos}
Classe de garantia	{garantido, melhor esforço}
Custo	{unidade monetária}

Tabela 5.3: Interface com o servidor multimídia

multimídia é feita basicamente através do tipo de mídia, do formato e do tamanho. Existe também uma informação de sincronização associada aos objetos, na forma de throughput, atraso e jitter, onde esses valores podem referir-se ao tempo de acesso nos dispositivos de armazenamento ou ao tempo necessário para a efetiva transmissão na rede, que pode ser maior em virtude da decodificação de formatos, por exemplo. Há também uma classe de garantia do serviço, que determina o grau de comprometimento assumido pelo servidor. Os parâmetros definidos pelo modelo estão descritos na Tabela 5.3.

A última interface descrita é aquela entre a QoS e o serviço de transporte multimídia. Por tratar-se de dados multimídia, algumas características são assumidas para esse serviço de transporte. Ele deve ser orientado a conexão, já que dados multimídia são geralmente contínuos e com conexões de longa duração. Supõe-se também que a transmissão seja unidirecional e ponto-a-ponto, já que se assumiu um modelo de servidor multimídia e cliente multimídia. Finalmente, o serviço de transporte deve ser orientado a transmitir unidades de transporte (TSDU - Transport Service Data Unit), e não bytes diretamente. Isto porque dados multimídia são geralmente codificados em blocos, como é o caso de áudio PCM ou vídeo MPEG. Para esta interface são descritos os parâmetros mostrados na Tabela 5.4.

Além de todas estas interfaces descritas acima, o modelo descreve um protocolo de negociação de QoS entre os três elementos citados: o servidor, o cliente e o serviço de transporte. Nesse modelo, a negociação e a renegociação de QoS são tratadas pelo mesmo protocolo. A seguir, mostraremos uma variante do protocolo descrito em [72]. A Figura 5.4 mostra o protocolo, onde as etapas são as seguintes:

1 O cliente pede os parâmetros de QoS de um objeto multimídia particular;

Parâmetro	Valores
Tamanho máximo da TSDU	{bytes}
Throughput	{TSDUs por segundo}
Delay	{segundos}
Jitter	{segundos}
Classe de garantia	{garantido, melhor esforço}
Custo	{unidade monetária}
Controle de erros	{recuperação, taxa de erros}

Tabela 5.4: Interface com o serviço de transporte

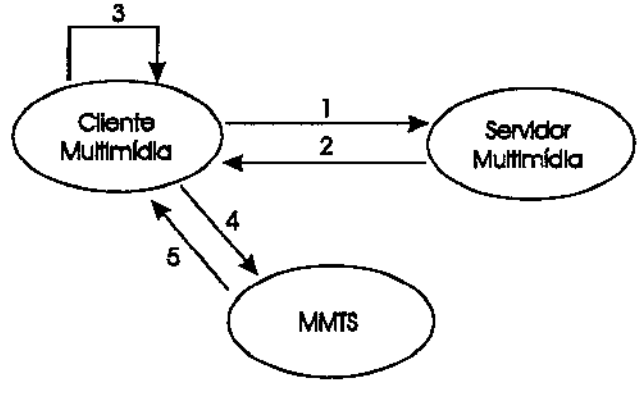


Figura 5.4: Negociação de QoS por exemplos

2 O servidor devolve um conjunto de tuplas representando a QoS associada. Há mais de uma tupla no caso de haver vários formatos do mesmo objeto;

3 O cliente negocia com o sistema operacional local os parâmetros recebidos do servidor, e também com o cliente. O resultado da negociação é traduzido em um formato compatível com o MMTS;

4 O cliente requisita do MMTS a QoS resultante da etapa anterior;

5 O MMTS confirma ou recusa a reserva de QoS.

5.2 Requisitos e Métricas de QoS

A primeira questão a ser resolvida no modelo proposto é a definição dos requisitos e métricas de QoS. Neste contexto, requisitos são as características de comportamento que serão avaliadas, e as métricas são os parâmetros utilizados para avaliar os requisitos.

Algumas definições de requisitos e métricas existem na literatura, mas a maioria é dirigida a uma classe específica de aplicações, geralmente as aplicações multimídia [62, 28, 25, 64, 23]. Dos modelos apresentados na seção anterior, nos modelos QoSME e QoS-A os requisitos são voltados para os protocolos de transporte (atraso, jitter, etc). No modelo de QoS por exemplos algumas métricas de nível mais alto são apresentadas, mas novamente voltadas para aplicações multimídia. O objetivo neste modelo é definir requisitos que se adaptem a várias classes de aplicações, e que trabalhem nos diferentes níveis de abstração que há entre a rede e as aplicações. Utilizando os mecanismos de herança, cada categoria de aplicações pode escolher um subconjunto dos requisitos aplicável ao seu contexto. Ao escolher-se este conjunto de requisitos e métricas, outro objetivo foi escolher parâmetros simples e sumários, que possam ser facilmente especificados e conferidos. A seguir são listados os requisitos e métricas definidas para o modelo proposto.

Sincronismo Síncrono, assíncrono ou isócrono.

Interatividade Throughput, atraso, jitter e MTU.

Prioridade Alta prioridade, baixa prioridade.

Controle de Erros Nenhuma ação, detecção, detecção e correção, ou métrica específica do protocolo.

Capacidade Velocidade agregada de CPU (MIPS, MFLOPS), capacidade de disco e o tamanho da RAM agregada.

Tolerância a falhas Redundância de host, linha de comunicação, CPU, RAM e fonte elétrica, classificação RAID dos discos.

Confidencialidade Classificação A1, B1, B2, C1, C2 ou C3.

Autenticidade Classificação A1, B1, B2, C1, C2 ou C3.

Integridade Classificação A1, B1, B2, C1, C2 ou C3.

Disponibilidade MTBF, MTTR e porcentagem de disponibilidade.

Custo Por minuto de CPU, por megabyte armazenado, por página impressa, por octeto transmitido e por largura de banda utilizada.

Os requisitos de sincronismo controlam a interação temporal entre as aplicações participantes da comunicação. Essa interação é geralmente categorizada em três classes: aplicações assíncronas, síncronas ou isócronas. Aplicações assíncronas são aquelas onde não há nenhum sincronismo explícito entre as partes comunicantes. Por exemplo, os protocolos de transferência de arquivos, como o FTP, geralmente não se preocupam nem permitem especificar quanto tempo será gasto na transmissão. Outro exemplo são aplicações de RJE (Remote Job Entry). Nestas aplicações, um "job" é submetido através da rede a um centro de computação, que devolve os resultados tão logo termine o processamento. Novamente, não há nenhuma forma de conhecer *a priori* o tempo que será gasto na interação. As aplicações síncronas são aquelas onde há um compromisso de tempo de resposta entre emissor e receptor. Por exemplo, em uma aplicação de emulação de terminal, existe um compromisso temporal entre o instante em que se pressiona uma tecla e o instante em que ela é ecoada no terminal.

As aplicações isócronas, que são aquelas mais exigentes em relação ao sincronismo, exigem um compromisso temporal de granularidade menor, no nível da unidade de dados, isto é, em nível de pacotes ou quadros ("frames"). Cada unidade de dados possui um compromisso temporal, tanto em termos absolutos (quanto tempo é gasto do emissor para o receptor) quanto em termos relativos (qual o atraso em relação às unidades de dados anteriores). Um exemplo seria uma transmissão de vídeo, onde existe um compromisso em relação ao número de quadros esperados (por exemplo, 24 quadros por segundo), e ainda em relação ao intervalo entre a chegada dos quadros. A transmissão de voz é outro exemplo desta classe de aplicações.

Os requisitos de interatividade são complementares em relação aos requisitos de sincronismo. Enquanto que o sincronismo especifica somente a categoria na qual a aplicação se encaixa, aqui são especificados os parâmetros exigidos por aquela categoria de aplicação. As métricas utilizadas são: a vazão (throughput), o atraso (delay), a variação do atraso (jitter) e a unidade máxima de transferência (MTU - Maximum Transfer Unit). A vazão

indica a taxa de emissão de bits esperada da aplicação. A unidade aqui utilizada é de bits por segundo, por ser esta a unidade utilizada nas redes ATM (vide capítulo 6). Outras unidades possíveis seriam bytes por segundo, já que a transferência de dados nos protocolos de transporte é orientada a bytes, ou ainda quadros por segundo. Esta última seria particularmente adequada a aplicações multimídia, que transferem seus dados de aplicação em blocos (quadros). Porém, para que o modelo ficasse o mais genérico possível, escolhemos utilizar bits por segundo, que são um "mínimo denominador comum" entre as unidades de transferência de dados. Porém, deve-se lembrar que, ao mapear uma unidade de quadros por segundo para bits por segundo perde-se informação semântica. Esta informação é preservada declarando-se um valor conciso para a MTU.

A métrica de atraso indica o tempo máximo que se espera ser gasto pelo pacote no trajeto entre emissor e receptor. Em aplicações onde o tamanho do pacote tem uma grande variação, como a transmissão de vídeo MPEG, pode-se optar por indicar nesta métrica um valor médio de atraso, ou ainda o valor do atraso para o tamanho máximo do pacote (MTU). A variação do atraso indica o tempo máximo esperado entre as chegadas de pacotes (inter-arrival time). As aplicações que dependem desta métrica geralmente possuem mecanismos para compensar o atraso variável sofrido por um pacote, e devem indicar aqui um valor máximo, que indicaria o limite de tempo no qual a variação do atraso ainda pode ser corrigida. Nesta métrica aplica-se o mesmo exposto para o atraso em relação aos pacotes de tamanho variável.

Cabe aqui uma explicação sobre a relação entre a MTU e o tamanho dos pacotes. Neste contexto, a MTU indica o tamanho da unidade máxima de dados no nível de aplicação, que pode ser difirente da unidade máxima de transferência utilizada pelo protocolo de transporte. A MTU no nível de transporte indica o tamanho máximo de um pacote em uma tecnologia de rede específica. Para a maioria das aplicações, a MTU no nível de aplicação é maior que a MTU no nível de transporte, o que facilita os cálculos no mapeamento de QoS entre esses dois níveis.

Os requisitos de prioridade especificam uma ordenação das aplicações que indique sua importância relativa para o usuário. Aplicações de maior importância ou de requisitos mais estritos receberiam um indicativo de alta prioridade, e teriam precedência na utilização dos recursos do ambiente, particularmente no acesso à rede. Muitos protocolos de transporte, como o XTP, possuem uma métrica de prioridade [75]. Outros, como o TCP, indicam implicitamente essa prioridade, através de bits específicos no cabeçalho [12]. Porém, nas redes ATM, uma célula possui somente um único bit para indicar a prioridade, que pode ser alta (bit CLP = 0) ou baixa (CLP = 1) [68]. Por este motivo, escolhemos como métrica somente estes dois valores, que são suficientes para a maioria das aplicações. A vantagem de se utilizar o indicativo de prioridade em nível tão baixo é que se elimina o "overhead" de controlar a prioridade, que é deixado a cargo dos comutadores

ATM. As redes ATM serão melhor descritas no capítulo seguinte.

Os requisitos de controle de erros são talvez os mais genéricos de todos, por existirem em vários níveis semânticos, que envolvem desde a camada física até as aplicações. As métricas aqui utilizadas são também genéricas, podendo ser aplicadas em qualquer desses níveis, embora sejam particularmente adequadas ao protocolo de transporte. As métricas basicamente indicam a ação a ser tomada na presença de erros de comunicação, que pode ser: nenhuma ação, somente detectar o erro, detectar e corrigir, ou ainda ações específicas do protocolo de transporte. De acordo com a semântica dos dados, pode-se decidir não tomar nenhuma ação na presença de erros, como acontece com a transmissão de vídeo em tempo real. Em outros casos, basta que o erro seja indicado ao usuário, sem que outra ação seja tomada. Finalmente, pode-se tentar detectar e corrigir o erro. Há também ações específicas para cada protocolo. No XTP, por exemplo, a estratégia de controle de erros pode ser: FASTNAK (Fast Negative Acknowledment) ou Go-Back-N (retroceda "n" pacotes). Estas métricas serão melhor explicadas no capítulo que trata dos aspectos de implementação.

Os requisitos de capacidade controlam algumas características do ambiente computacional que afetam diretamente a Qualidade de Serviço. As métricas aqui adotadas não têm a intenção de ser completas, do ponto-de-vista de desempenho do sistema computacional. Se esta fosse a intenção, várias outras métricas teriam de ser acrescentadas [32]. O objetivo aqui é somente delimitar métricas simples e de fácil acesso, que determinem de forma global o comportamento do sistema computacional. São elas: a velocidade agregada de CPU, a capacidade de disco e o tamanho da RAM agregada. A primeira métrica indicaria, em termos gerais, a velocidade de processamento do conjunto de CPUs, que seria medida em MIPS (Milhões de Instruções por Segundo), para o caso onde a maioria das operações fosse de ponto inteiro, ou em MFLOPS (Milhões de operações em ponto flutuante), para os casos onde a maioria das operações fosse de ponto flutuante. A segunda métrica indicaria a capacidade de disco livre passível de uso no sistema, expressa em megabytes. A terceira métrica indicaria a quantidade de memória RAM instalada no sistema, também expressa em megabytes.

Caso o sistema computacional seja de uso específico, outras métricas podem ser utilizadas. Por exemplo, se o ambiente for direcionado a transações on-line, poderia-se utilizar a métrica TPS (Transações por segundo). Caso o ambiente seja direcionado a aplicações WWW (World Wide Web), poderia ser utilizada a métrica de número de "hits" por minuto, que indicaria a quantidade de páginas Web que podem ser recuperadas a cada minuto.

A tolerância a falhas é também um requisito importante para a Qualidade de Serviço de alto nível. A existência desses mecanismos está intimamente ligada à disponibilidade do sistema. Neste contexto, a tolerância a falhas é indicada através da existência de

redundância de hardware. Há mecanismos mais complexos de tolerância a falhas, inclusive introduzindo tolerância de software. Porém, aqui vale o mesmo argumento aplicado aos requisitos de capacidade: o objetivo das métricas não é cobrir todo o espectro de QoS, mas sim disponibilizar métricas simples e sumárias. As métricas aqui utilizadas indicam a existência de redundância nos seguintes itens: redundância de "host", linha de comunicação, CPU, memória RAM e fonte elétrica, além da classificação RAID (Redundant Array of Independent Disks), do sistema de discos. A classificação RAID indica um número, de zero a seis, que indica diversos esquemas de redundância através de paridade nos discos.

Os três requisitos que se seguem estão também intimamente relacionados: a confidencialidade, a autenticidade e a integridade. O primeiro indica o grau de privacidade de uma comunicação, procurando garantir o sigilo dos dados. A autenticidade implica em garantir que os pares da comunicação são realmente quem dizem ser. A integridade implica em garantir que, durante a comunicação, os dados não serão perdidos ou modificados. Para cada um desses requisitos há algoritmos próprios, que asseguram diferentes graus de segurança. Para sumarizar estes requisitos em métricas simples, escolheu-se utilizar a classificação da NSA (National Security Agency), uma agência norte-americana que emite padrões de segurança. A NSA certifica um ambiente de acordo com as classificações: A1, B1, B2, C1, C2 e C3. Os três requisitos apresentados seguem esta classificação quanto aos aspectos aplicáveis em cada um deles, já que a classificação NSA abrange muito mais fatores que estes citados.

Os requisitos de disponibilidade, como o próprio nome indica, têm por objetivo especificar o grau de disponibilidade de um sistema computacional. Novamente, optamos por utilizar métricas sintéticas em prol da simplicidade. As métricas escolhidas foram o tempo médio entre falhas (MTBF - Mean Time Between Failures), o tempo médio entre reparos (MTTR - Mean Time To Repair), expressos em horas, e a porcentagem do tempo em que o sistema se encontra disponível. O MTBF e o MTTR são mais aplicáveis a equipamentos, enquanto que a porcentagem de disponibilidade é uma métrica mais genérica, que pode também englobar a disponibilidade do software.

Finalmente, decidiu-se incluir uma métrica que especificasse um custo associado a cada recurso do sistema. Conforme explica Vogel [72], caso não haja uma métrica de custo que limite o uso de recursos do sistema, não há nenhum motivo pelo qual um usuário não tente reservar sempre o máximo de recursos possível, embora possa haver políticas que restrinjam essa prática. Assim, as métricas indicam o valor de cada recurso: custo por minuto de CPU utilizado, por megabyte armazenado, por byte transmitido e por largura de banda utilizada.

Pode-se verificar que os requisitos de QoS definidos são de duas categorias distintas: há alguns requisitos que são de alto nível, que possuem atributos estáticos ou quase estáticos,

e que representam características do ambiente computacional. A outra categoria representa requisitos de nível mais baixo, que possuem atributos dinâmicos e que representam características da infra-estrutura de rede que suporta o ambiente distribuído. Essas duas categorias, por conta de suas diferenças, serão negociadas com elementos diferentes do modelo, com interfaces e etapas distintas. A seguir é mostrada a divisão dos requisitos de acordo com esta classificação.

Requisitos de Canal Sincronismo, Interatividade, Prioridade e Controle de erros. São de baixo nível e possuem atributos dinâmicos.

Requisitos de Ambiente Capacidade, Tolerância a falhas, Confidencialidade, Autenticidade, Integridade e Disponibilidade. São de alto nível e possuem atributos semi-estáticos.

As etapas de negociação de QoS de acordo com esta classificação serão apresentadas na próxima seção.

5.3 Visão Geral do Modelo

A Figura 5.5 mostra o modelo proposto. Na figura, os círculos indicam os objetos implementados neste trabalho, e os quadrados indicam elementos já existentes na plataforma ORB ou na plataforma Multiware. As linhas pontilhadas indicam chamadas ORB, que podem ser remotas ou locais. As linhas contínuas indicam chamadas locais a objetos que têm de existir no mesmo nó que o objeto chamador.

Visualmente podemos notar que o elemento central do modelo é o Objeto de Negociação de QoS (ONQoS) [38]. O Objeto Trader tem também funções de negociação de QoS, porém em nível mais alto. Os objetos localizados no caminho entre o BEO cliente e o BEO servidor realizam a reserva e a monitoração da QoS na infra-estrutura da rede. A seguir, detalharemos a função de cada objeto do modelo e o processo de negociação de QoS e comunicação no ambiente distribuído.

Na seção anterior descrevemos os requisitos definidos para o modelo e os classificamos em dois conjuntos: requisitos de canal e requisitos de ambiente. Por conta das diferenças entre esses dois conjuntos de requisitos explicadas anteriormente, a negociação de QoS foi dividida em duas etapas. Em uma primeira etapa, os requisitos de ambiente são negociados com o Trader ODP, e na etapa seguinte, os requisitos de canal são negociados através de um Objeto de Negociação de QoS (ONQoS). Os requisitos negociados com o Trader especificam características do ambiente computacional, enquanto aqueles negociados com o ONQoS especificam as características desejadas para o canal de comunicação.

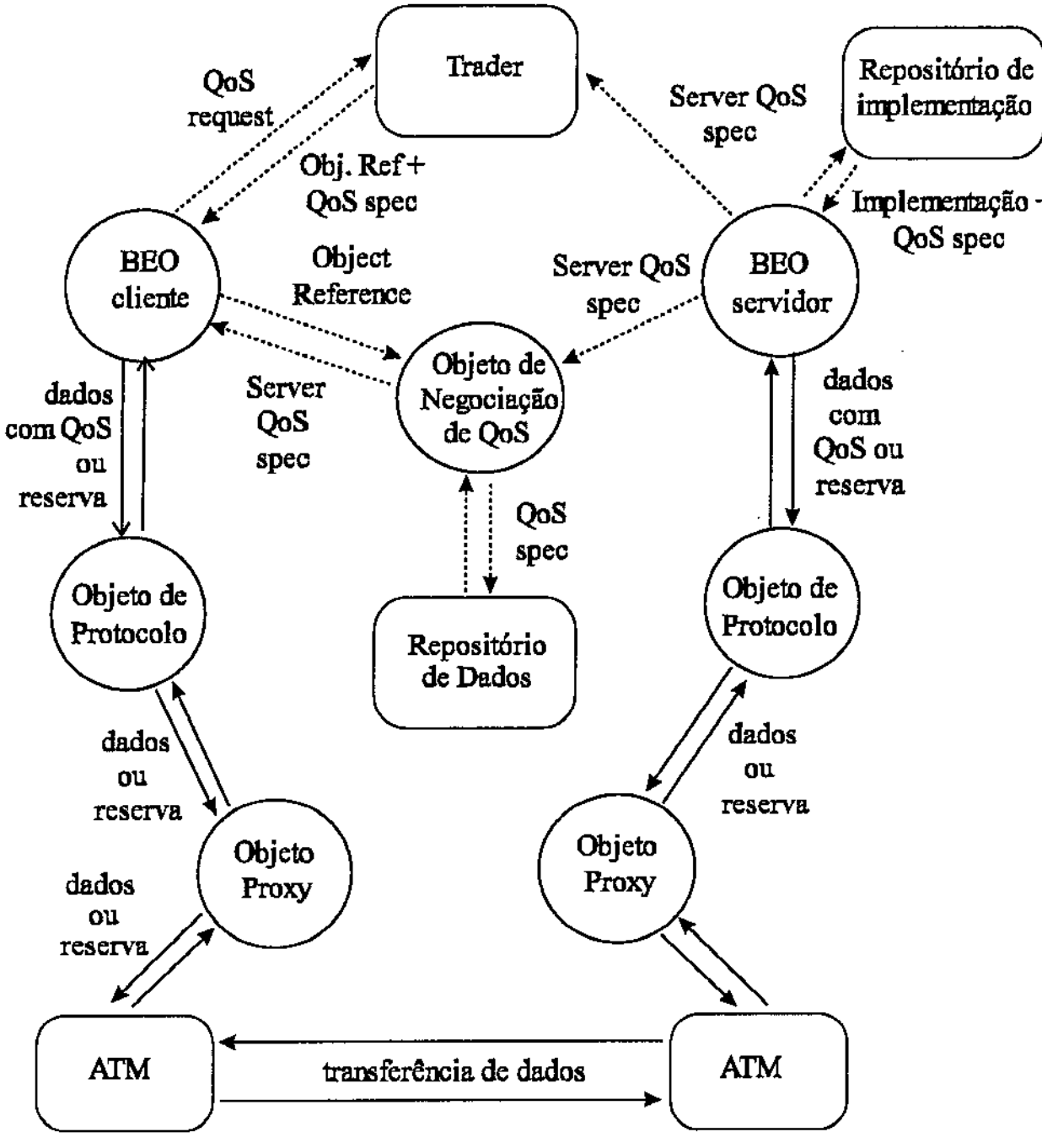


Figura 5.5: O modelo proposto

De acordo com o tipo de aplicação, somente uma das etapas pode ser necessária. Aplicações que só estão preocupadas com requisitos de alto nível usam somente a primeira etapa. Aplicações que só precisam de garantias de rede, utilizam somente a segunda fase. Aplicações mais complexas podem precisar de ambas. Por exemplo, se é necessária uma comunicação de voz autenticada, os requisitos de autenticação são negociadas com o Trader, e os requisitos para a transmissão de voz são negociados com o ONQoS.

A negociação com o Trader usa as interfaces de importação e exportação descritas na seção 4.1. Essas interfaces são bastante flexíveis, e permitem realizar pesquisas por vários critérios. A negociação com o ONQoS utiliza uma interface própria, que será descrita na próxima seção.

Conforme descrito na seção 4.1, no RM-ODP os BEOs (Basic Engineering Objects) são as aplicações usuárias do ambiente. Um BEO pode ter o papel de cliente e/ou servidor. Um BEO atua como cliente quando está invocando um serviço, e atua como servidor quando está respondendo a pedidos de invocação. Detalharemos a seguir o processo de negociação.

Inicialmente o BEO servidor cria seus serviços e registra seus requisitos de ambiente com o Trader. O Repositório de Implementação está no modelo somente para indicar que os requisitos de QoS podem ser armazenados junto com a implementação do Objeto Servidor. O BEO servidor também registra seus requisitos de canal com o ONQoS. O ONQoS tem acesso a um repositório de dados, para que os registros de QoS possam tornar-se persistentes. Os dados registrados, tanto no Trader quanto no ONQoS, indicam os valores limítrofes dos requisitos (mínimos ou máximos, dependendo da métrica). Os requisitos de canal possuem dois valores para cada métrica, que indicam os requisitos limítrofes em cada uma das direções da comunicação (do servidor para o BEO e vice-versa).

Quando um BEO cliente deseja algum serviço, ele tem duas alternativas. Caso ele conheça a referência do Objeto Servidor, ele a submete ao Trader, que devolve a especificação de QoS do servidor, que pode ser checada para verificar se a comunicação pode acontecer. Caso o BEO cliente queira qualquer servidor que se encaixe em seus requisitos, ele envia os requisitos desejados, e o Trader retorna uma ou várias referências de objeto. Esta é a primeira etapa de negociação.

Na segunda etapa, quando o BEO cliente já tem a referência do BEO servidor, ele a envia ao ONQoS. A partir da referência, o ONQoS obtém os requisitos de QoS do BEO servidor e os entrega ao BEO cliente. Os requisitos de QoS do BEO cliente são então calculados e passados ao Objeto de Protocolo na forma de um pedido de reserva de recursos. Esse pedido é atendido em parte pelo Objeto de Protocolo e em parte pelo Objeto Proxy, de acordo com a métrica. Um "handler" é retornado ao BEO cliente, que o utiliza para transferir os dados. O BEO servidor também pode fazer uma reserva de

QoS, já que a QoS só é reservada em uma direção.

5.4 O Objeto de Negociação de QoS

O Objeto de Negociação de QoS, como o próprio nome indica, tem por função intermediar a negociação de QoS em um ambiente distribuído aberto. Pelas suas características podemos dizer que, como o Trader, o ONQoS é uma extensão do serviço de nomeação (naming), podendo inclusive ser implementado como parte dele. A interface do ONQoS é bastante simples, e será detalhada no próximo capítulo. A interface é composta de métodos para recuperar, incluir, modificar ou excluir registros de QoS. Os registros de QoS são compostos dos requisitos e métricas apresentados, indicando os valores limítrofes para as duas direções da comunicação. Em alguns casos somente uma direção precisa ser indicada, por exemplo, no caso de servidores de vídeo que somente suportem a distribuição multicast em uma direção, recebendo somente os controles por parte do cliente.

Pode-se também imaginar uma interface mais complexa para o ONQoS, que inclua métodos de pesquisa. Porém, conforme explicamos anteriormente, esta função já é exercida pelo Objeto Trader [38]. Outro motivo pelo qual não haveria necessidade de uma interface complexa é que, nesta etapa, o Objeto Cliente já possui a referência do Objeto Servidor, e portanto já escolheu o parceiro da comunicação.

Outra alternativa ao modelo seria que o próprio ONQoS fizesse o pedido de reserva de recursos da rede. Esta alternativa não foi adotada por causa da possível localização dos objetos. O ONQoS pode estar em qualquer nó da rede. Para que ele mesmo realizasse o pedido de reserva de recursos, seria necessário que o Objeto de Protocolo (que é responsável pela interface com a rede) aceitasse chamadas ORB. Como podem ser feitos vários pedidos de reserva, indo de um valor mais alto para um valor mais baixo, o número total de chamadas ORB e de possíveis erros de comunicação seria muito elevado. Além disto, a interface do Objeto de Protocolo é grandemente simplificada pelo fato das chamadas serem feitas localmente ao nó.

5.5 O Objeto de Protocolo

O Objeto de Protocolo tem o objetivo de ser um protocolo de transporte com funções de tempo real e com requisitos de QoS. As interfaces de QoS acrescentadas, que serão descritas em detalhes no capítulo 6, permitem que o BEO Cliente e o Objeto Servidor façam pedidos de reserva de recursos, utilizando os requisitos de canal. Os pedidos incluem somente os valores limítrofes para cada métrica. Caso a rede não consiga fornecer o nível de QoS desejado, um indicativo de erro é retornado.

Uma alternativa a esta abordagem seria especificar um intervalo de valores de QoS aceitáveis, os quais o Objeto de Protocolo tentaria reservar, indo do valor mais alto para o valor mais baixo. Porém, nem todas as métricas aceitam intervalos, e a combinação dos intervalos de várias métricas geraria valores difíceis de rotular como "aceitáveis" ou "não aceitáveis". Em vista desta dificuldade, decidiu-se que seria uma tarefa da aplicação requisitar vários pedidos de QoS caso necessário. A aplicação possui meios de concluir mais acertadamente quais combinações de valores são aceitáveis. Além disto, como as chamadas de reserva de recursos são locais, pode-se facilmente gerar várias chamadas por segundo. Em testes realizados, obteve-se um tempo médio de 40 milissegundos entre o pedido de reserva e o resultado [39].

Como descrito na Figura, um Objeto de Protocolo deve existir no mesmo nó que o BEO cliente. De acordo com a localização e a forma de implementação do Objeto de Protocolo, há varias formas de efetivar a comunicação com o BEO. Caso o Objeto de Protocolo resida na mesma cápsula que o BEO, pode-se utilizar uma forma de comunicação por memória compartilhada e chamadas de função. Caso o Objeto de Protocolo esteja em uma cápsula distinta, deve-se utilizar alguma forma de IPC (Inter-Process Communication). O mesmo se aplica à comunicação entre o Objeto de Protocolo e o Objeto Proxy. A única restrição aplicada pelo modelo é que a comunicação não utilize chamadas ORB, por serem muito lentas e poderem gerar vários erros, que complicariam a interface do Objeto de Protocolo.

Por ser um protocolo de tempo real, o Objeto de Protocolo deve comunicar ao BEO sempre que os requisitos de tempo real não puderem ser atendidos. O BEO, por sua vez, deve prever este caso, e decidir que ação será tomada. A forma como o Objeto de Protocolo sinalizará a mudança na QoS obtida depende do protocolo adotado. Conforme visto no capítulo 2, em face a uma degradação da QoS requisitada, uma aplicação pode adaptar-se ao novo padrão de QoS, tentar renegociar novos níveis de QoS, ou simplesmente cancelar a conexão.

5.6 O Objeto Proxy

O Objeto Proxy tem duas funções. A primeira é ser um protocolo de rede genérico, com funções para transferir unidades de dados do usuário, informações de controle, etc. A interface de rede do Objeto Proxy é usada pelo Objeto de Protocolo para implementar a camada de transporte. A segunda função é mapear o pedido de reserva de recursos nos parâmetros de tráfego das redes abaixo dele. O objetivo disto é que o Objeto de Protocolo trate exclusivamente com o nível de transporte e níveis superiores, usando chamadas genéricas. O Objeto Proxy, por sua vez, seria específico para a arquitetura de rede utilizada. Caso a arquitetura de rede possua suporte a QoS, o Proxy também pode incluir as chamadas específicas de reserva e monitoração dos recursos. Caso não haja

suporte a QoS na arquitetura, o Objeto Proxy pode utilizar outros mecanismos, como o controle de admissão de chamadas e a reserva de buffers de transmissão e recepção.

De acordo com o modelo, poderiam existir vários Proxys, cada um traduzindo as APIs (Application Programming Interface) específicas da arquitetura em chamadas "canônicas", independentes da arquitetura. Caso fosse trocada a infra-estrutura da rede, o Proxy poderia ser trocado sem a necessidade de modificar nenhum outro objeto. Porém, devido à grande discrepância de parâmetros de QoS entre as redes de alta velocidade hoje existentes, poucas arquiteturas poderão ser adaptadas sem modificação. Outro possível modelo de Objeto Proxy seria para um protocolo de rede que interagisse diretamente com os roteadores para fazer a reserva de recursos, como é o caso dos protocolos ST2 e RSVP, descritos no capítulo 3. Além de reservar recursos nos roteadores, outra função bastante útil deste tipo de Proxy seria fazer roteamento baseado em QoS, onde as rotas seriam escolhidas com base nas características de QoS escolhidas.

Conforme explicado anteriormente, adotou-se o ATM como arquitetura de referência para a implementação do Modelo. Assim sendo, o Objeto Proxy implementado trabalha com redes ATM, utilizando a AAL5 como meio de transporte (vide capítulo 6). Além de suportar diretamente a reserva de recursos e vários parâmetros de QoS, a arquitetura ATM prevê a função de rotemento baseado em QoS, o que representa um grande impacto positivo no uso dos recursos. O protocolo PNNI fase 1 (Private Network-to-Network Interface), que é parte da arquitetura ATM e que está sendo padronizado, suporta roteamento baseado em QoS. Nenhuma outra arquitetura de rede atual tem tantos recursos de suporte a Qualidade de Serviço.

5.7 Objetos Acessórios

O Modelo proposto inclui ainda alguns objetos acessórios, que são o Repositório de Implementação, o Repositório de Dados e o Objeto Trader. O Repositório de Implementação é um elemento da arquitetura CORBA responsável por guardar dados sobre a implementação de um objeto servidor, como o código executável, informações de depuração, informações sobre versões, etc. Este seria, portanto, o local ideal para armazenar os requisitos de QoS do Objeto Servidor, quando o mesmo estivesse inativo. Porém, nas implementações de CORBA disponíveis atualmente, não se tem acesso direto ao Repositório de Implementação. Em um modelo ideal, quando o objeto servidor fosse ativado, um método de inicialização leria os requisitos de QoS e registraria os requisitos de ambiente com o Objeto Trader. Os requisitos de canal do Objeto Servidor seriam requisitados quando houvesse um pedido de conexão vindo de um BEO cliente. Neste momento o Adaptador de Objetos realizaria o pedido de recursos através do Objeto de Protocolo. Novamente lembrando, a reserva de recursos é feita individualmente em cada uma das

direções, daí a necessidade de ser feita também pelo Objeto Servidor. O Adaptador de Objetos é um outro elemento da arquitetura CORBA responsável por ativar e desativar os objetos servidores, de acordo com as políticas de ativação, dentre outras funções.

O Repositório de Dados, como o próprio nome indica, armazena dados genéricos em um ambiente orientado a objetos. No modelo, um repositório de dados seria necessário somente para tornar persistentes os registros guardados pelo ONQoS. Caso este objeto tenha uma interface inteligente, pode-se deixar a seu cargo o controle de invalidação dos registros. Como os requisitos de QoS relativos ao canal podem mudar de acordo com a localização do Objeto Servidor, pode-se determinar um tempo máximo de validade para um registro, depois do qual ele seria removido, forçando um novo registro por parte do Objeto Servidor. Isto seria feito para prevenir que os BEOs Clientes recebam informações desatualizadas. Por exemplo, um Objeto Servidor de vídeo pode migrar de uma conexão ATM de alta velocidade para uma conexão Frame Relay, de velocidade menor. Isto forçaria uma redefinição dos requisitos de canal por parte do Objeto Servidor.

O Objeto Trader utiliza as interfaces de exportação e importação para negociar os requisitos de ambiente. Um Objeto Trader Federado já foi implementado para a Plataforma Multiware, e foi adotado como parte do modelo [40]. O Repositório de Implementação, o Repositório de Dados e o Adaptador de Objetos já existem como parte das plataformas ORB, como a Orbix ou o Visibroker, e por isto não houve a necessidade de implementá-los. Cabe ressaltar que os Repositórios de Implementação e de Dados e os Adaptadores de Objeto atualmente presentes nas plataformas ORB citadas são pouco flexíveis, não permitindo sua manipulação direta pelo usuário.

Capítulo 6

Aspectos de Implementação

6.1 Principais Aspectos de Redes ATM

Nesta seção descreveremos os principais aspectos das redes ATM, dando especial ênfase àqueles que serão necessários à compreensão deste capítulo.

O ATM é um protocolo de transporte, comutação e multiplexação de células. Ele utiliza estruturas de 53 bytes chamadas células, assim conhecidas por serem de tamanho fixo, em contraste com os pacotes, que têm tamanho variável. O ATM é um dos protocolos padronizados para a B-ISDN (Broadband Integrated Services Digital Network), destinado a transportar dados, voz, imagens e vídeo de forma transparente entre entidades de nível superior [68, 52]. É um protocolo bastante simples, e que utiliza meios bastante confiáveis para o transporte de células. A conjugação desses dois fatores faz com que o ATM atinja velocidades da ordem dos gigabits por segundo.

Vários fatores fizeram com que o ATM se tornasse uma tecnologia revolucionária. Primeiramente, o aumento da qualidade dos meios de transmissão. A utilização de fibra ótica proporciona velocidades de transmissão só limitadas pela velocidade da luz, além de possibilitar taxas de erro muito baixas, da ordem de um erro a cada bilhão de pulsos transmitidos. Este fato fez com que se pudesse desenhar um protocolo "enxuto", retirando todo o "overhead" de controle de erros.

O ATM é um protocolo baseado na multiplexação e na comutação de células. Ao contrário do STM (Synchronous Transfer Mode), que divide a banda em slots de tempo fixos, o ATM insere assincronamente as células no meio sempre que necessário. Desta forma, o que é assíncrono no ATM é a alocação de banda, e não a técnica de sinalização do meio, que é síncrona na maioria dos casos [52].

A comutação de células é feita pelos elementos intermediários da rede ATM, os comutadores (switchers). Esses elementos, graças à simplicidade do protocolo, são também bastante simples, podendo ser implementados inteiramente em hardware, ou em uma con-

jugação de hardware e software. Uma vez estabelecido um circuito entre fonte e destino, as células seguem por esse caminho.

O protocolo ATM fornece serviços que equivalem àqueles fornecidos pela camada física do modelo OSI. A camada AAL (ATM Adaptation Layer) fornece serviços que equivalem àqueles fornecidos pela camada de enlace do modelo OSI [52].

6.1.1 As células ATM

A célula ATM é composta de 53 bytes, sendo que 48 deles formam o payload (campo de informação) e cinco bytes formam o cabeçalho da célula. A célula ATM tem pequenas diferenças no seu formato na UNI (User-Network Interface) e na NNI (Network-Network Interface). As Figuras 6.1 e 6.2 mostram o formato do cabeçalho da célula para a UNI e para a NNI, respectivamente. Neste projeto, onde utilizamos um comutador formando uma rede privada, somente as células no formato UNI serão usadas. A seguir descreveremos a função de cada campo do cabeçalho.

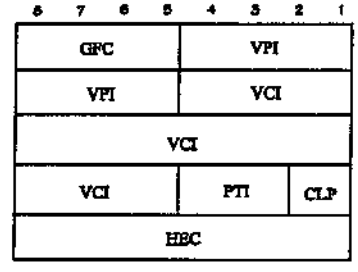


Figura 6.1: Célula ATM na UNI

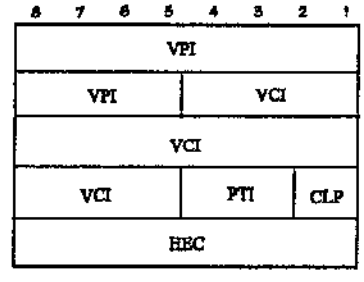


Figura 6.2: Célula ATM na NNI

O campo GFC (Generic Flow Control) é composto de 4 bits. Este campo é utilizado para prover um mecanismo de controle de fluxo. Foram definidos dois modos de operação: o modo de acesso controlado (controlled access) e o modo de acesso não controlado (uncontrolled access). No modo de acesso controlado, o host é obrigado a modificar seu padrão de transmissão de acordo com os bits GFC, que informariam a ocorrência

de congestionamento. O padrão da UNI é o acesso não controlado, onde esses bits são simplesmente ignorados.

Os campos VPI (Virtual Path Identifier) e VCI (Virtual Channel Identifier) são interpretados conjuntamente. O campo VCI identifica conexões do usuário com a rede. O campo VPI é usado para identificar um conjunto de conexões virtuais que seguem pelo mesmo caminho. O tamanho dos campos VPI e VCI é diferente nas células UNI e NNI. O tamanho de cada campo também pode ser negociado no estabelecimento das sessões. Quando um usuário local deseja estabelecer uma sessão com um usuário remoto, a rede retorna um número de VCI, que é usado pelo usuário local para a transferência de dados. Cada VCI pode estar associado a diferentes características de tráfego, Qualidade de Serviço, velocidade, etc.

O campo PT (Payload Type) possui três bits, que são usados para indicar se o dado sendo transportado pertence ao plano de usuário ou ao plano de gerenciamento. O campo CLP (Cell Loss Priority) possui um único bit, que é usado pelo usuário para indicar a prioridade das células. Células com CLP=0 são consideradas de alta prioridade. Células com CLP=1 são de baixa prioridade, e podem ser descartadas em um evento de congestionamento. O campo HEC (Header Error Control) possui um byte, contendo um cálculo de CRC para a célula, que é regenerado em cada nó comutador (os cabeçalhos das células são modificadas em cada nó).

6.1.2 Circuitos Virtuais

Um VCC (Virtual Channel Connection), ou conexão de canal virtual, é o menor recurso alocado na rede pelo usuário. Um canal virtual tem por função prover um "tubo" bidirecional de transferência de dados, entre emissor e receptor. A rede ATM cuida de fazer o mapeamento de canais virtuais entre os comutadores intermediários, de forma que o usuário só precise conhecer um número de canal virtual. Na fase de estabelecimento da chamada, o usuário, através do endereçamento ATM, identifica o usuário destino. A rede então, através das tabelas de roteamento, determina uma rota até o destino. Caso o usuário destino queira estabelecer a conexão, um número de VCC é retornado a ele. Uma vez estabelecida a rota, ela é usada para todas as células daquela conexão.

Um VPC (Virtual Path Connection), ou conexão de caminho virtual, é um agrupamento de canais virtuais que seguem pela mesma rota. O objetivo de um canal virtual é agrupar vários VCCs, facilitando assim as funções da rede, no tocante a gerenciamento de recursos, controle de congestionamento, policiamento ou mesmo tarifação pelo uso de recursos (Figura 6.3).

Conforme explicado anteriormente, através do mecanismo de VCC/VPC, o usuário consegue estabelecer circuitos virtuais entre a entidade fonte e a entidade destino. Um PVC (Permanent Virtual Circuit) é um circuito que é estabelecido de forma permanente.

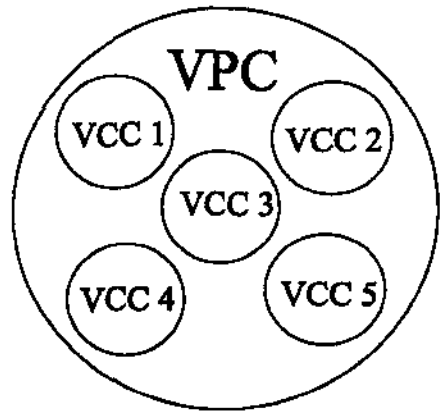


Figura 6.3: Conexões de Canal Virtual e Caminho Virtual

Em termos de arquitetura ATM, esses circuitos podem ser estabelecidos usando um software de controle no centro de operação da rede. O problema deste tipo de circuito é quando se tem um número muito grande de nós na rede. Se for estabelecido um PVC entre cada um dos "n" nós, teremos um número muito grande de conexões abertas simultaneamente, gastando recursos excessivos e dificultando a manutenção.

O outro tipo de conexão, chamado de SVC (Switched Virtual Circuit), é usado quando se quer estabelecer um circuito temporário. O melhor exemplo dessa necessidade é o tráfego de redes locais, que é inerentemente "connectionless" (modo datagrama). Quando uma estação quer manter comunicação com outra, um circuito temporário é estabelecido, e posteriormente desfeito quando não houver mais comunicação durante um determinado período.

6.1.3 Camadas de Adaptação do ATM

Conforme dissemos anteriormente, o ATM é um protocolo bastante simples, o que o torna mais rápido que outros protocolos. Porém, para que serviços de mais alto nível sejam disponibilizados, tornou-se necessário prover uma camada de adaptação acima do protocolo ATM, que é chamada de AAL (ATM Adaptation Layer). O ATM Forum dividiu a AAL em diversas classes, e cada classe de serviço tem uma ou mais AALs associadas. A Tabela 6.1 mostra as classes definidas pelo ATM Forum [68].

Entre os serviços de classe A estão a emulação de circuitos e a transmissão de vídeo a taxa constante. A emulação de circuitos consiste em propiciar a transmissão transparente de dados digitais, da mesma forma que as linhas digitais da N-ISDN. Na classe B estão os serviços de áudio e vídeo a taxas variáveis, como as aplicações multimídia. A diferença para a classe A é que este serviço funciona sob demanda e por um menor período. Na classe C estão as aplicações de transferência de dados orientada a conexão. Nesta categoria

Classe	Sincronismo	Paradigma	Emissão de bits
A	Isócrono	conexão	constante
B	Isócrono	conexão	variável
C	Não isócrono	conexão	contante
D	Não isócrono	datagrama	variável

Tabela 6.1: Classes de Serviço em ATM

se encaixa o tráfego dos outros protocolos transportados pelo ATM, como Frame Relay ou SNA. A classe D inclui os serviços de transferência de dados orientados a datagrama, sendo o maior exemplo o tráfego de redes locais.

A AAL é dividida em duas sub-camadas: SAR (Segmentation Assembly and Reassembly) e CS (Convergence Sublayer). A sub-camada SAR é responsável pela segmentação e remontagem das SDUs (Service Data Units), que são as unidades de dados do usuário, subdividindo-as em células ATM e reconstituindo a unidade de dados do outro lado da conexão. Esta sub-camada existe em praticamente todas as aplicações do ATM. A sub-camada CS é responsável pela adaptação do serviço prestado pelo ATM às necessidades do usuário, e pode não existir em alguns casos. O ATM Forum definiu vários tipos de AAL, cada um adaptado para uma determinada classe de serviço. Veremos a seguir cada um deles.

A AAL tipo 1 foi desenhada para cumprir os requisitos dos serviços classe A, isto é, serviços isócronos, orientados a conexão e com taxa de bits constante. Para isto foi [illegible] SAR e várias sub-camadas CS, uma para cada tipo de conexão.

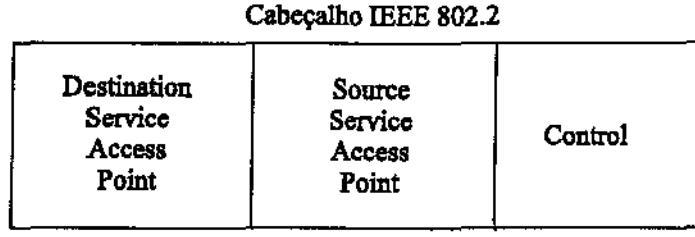


Figura 6.6: Header LLC para encapsulamento em AAL5

O campo DSAP (Destination Service Access Point) identifica o destino da PDU no outro lado da conexão. O campo SSAP (Subnetwork Service Attachment Point) identifica sub-redes no destino, de acordo com a especificação IEEE 802.1. O campo de controle sempre tem o valor fixo 0x03.

Dentro desta especificação é possível encapsular protocolos roteáveis, como IP e IPX, e também protocolos de enlace, que são transportados como uma "bridge", como o ethernet, por exemplo. O encapsulamento de protocolos de enlace em AAL5 é conhecido como "LAN Emulation".

No método de encapsulamento baseado em VC, nenhuma informação de controle é necessária para transportar vários protocolos, já que a multiplexação seria baseada no número do VC. O protocolo de sinalização do ATM, o Q.2931, é capaz de criar conexões de forma estática ou dinâmica para atender aos diversos protocolos encapsulados.

6.2 Ambiente de Implementação

Conforme dissemos na introdução, o modelo proposto no capítulo 5 foi desenvolvido tendo-se em mente sua implementação em uma rede ATM, por causa de suas características de alta velocidade e da existência de mecanismos de controle de tráfego. A implementação aqui descrita segue este Modelo proposto. Há na literatura algumas implementações de protocolos de tempo real e modelos de QoS baseados em redes ATM [19, 20, 18, 6, 3].

A implementação foi realizada no Laboratório de Redes de Alta Velocidade do Departamento de Ciência da Computação da UFMG. O ambiente utilizado foi uma rede ATM, composta de um comutador CISCO Hyperswitch A100, com 15 portas de 155 Mbps (OC-3), usando cabeamento de fibra ótica multimodo e par trançado categoria 5 (UTP cat 5). Ao comutador estão ligadas duas SPARC Ultra 1, de 145 Mhz, com 64 Mb de RAM, além de uma SPARC 5 com 32 Mb de RAM. Todas as estações possuem placas SunATM 2.0 e sistema operacional Solaris 2.5. As duas SPARC Ultra foram usadas na implementação, já que a transferência de dados em alta velocidade exige CPUs e barramentos rápidos. Em testes de desempenho, observamos que o barramento da SPARC 5 não foi capaz de transferir os dados na velocidade máxima da interface ATM. Já as SPARC Ultra conseguiram

aproximar-se da velocidade máxima da interface.

Este ambiente proporciona o uso das camadas de adaptação do ATM nos níveis AAL0, AAL1 e AAL5. A AAL0 equivale a não utilizar a camada de adaptação, onde os dados são transferidos diretamente da camada ATM para a interface STREAMS, que será explicada a seguir. A AAL1, conforme explicado na seção anterior, é adequada para serviços que possuem uma taxa de emissão de bits constante, o que não é o caso do tráfego de redes locais. A AAL5 é voltada para a transferência de dados orientados a conexão e assíncronos, sendo o padrão usado atualmente para transmissão de dados em redes locais. Por este motivo, escolhemos a AAL5 como padrão para a implementação dos protocolos de nível superior.

6.3 O SandiaXTP

Como base para nossa implementação, utilizamos a implementação do XTP conhecida como SandiaXTP, do Sandia National Laboratories [60, 61]. Esta é a implementação mais utilizada do XTP, por ter um código bem estruturado e por ser constantemente atualizada, já sendo compatível com a versão 4.0 do protocolo. O SandiaXTP foi totalmente implementado em C++, o que o torna mais simples de adaptar.

O SandiaXTP é uma implementação orientada a objeto da versão 4 do protocolo XTP. O núcleo das classes utilizadas pelo XTP é baseada na MTL (Meta Transport Library) [59]. A MTL é responsável por prover as primitivas de comunicação relativas ao nível de rede. A MTL pode utilizar IP ou UDP como protocolo de entrega de dados. Neste trabalho implementamos a entrega de dados via AAL5.

A comunicação no SandiaXTP é inteiramente full-duplex. Cada elemento comunicante é, ao mesmo tempo, transmissor e receptor. A máquina de estados do SandiaXTP o define como um protocolo orientado a emissor. Assim sendo, o transmissor define as características da comunicação, envia estas características ao receptor, que pode concordar, concordar com modificações ou rejeitá-las. A partir deste ponto, o receptor passa a reagir de acordo com os pacotes de controle recebidos.

O SandiaXTP é implementado como um "daemon" em nível de usuário. O daemon recebe requisições dos usuários, através de rotinas de interface, e comunica-se com o daemon remoto para trocar pacotes de dados e de controle. Os pacotes de dados são entregues aos usuários através de uma área de memória compartilhada. Alguns dos pacotes de controle são interpretados diretamente pelo deamon, e outros são entregues aos usuários, também utilizando memória compartilhada. Na inicialização do daemon pode-se escolher qual protocolo de entrega de dados (IP, UDP, AAL5) será utilizado. Nesse momento pode-se também definir várias características de operação do protocolo, como o número máximo de contextos permitidos, o tamanho da área de recebimento de pacotes, etc [60, 61].

A interface dos usuários com o SandiaXTP é feita através de uma interface de programação baseada em uma classe (xtpif). O usuário instancia um objeto a partir da classe xtpif, que quando inicializado, contém o identificador da associação, que é associado a um contexto XTP. Todas as operações disponíveis aos usuários são métodos da classe xtpif. O código gerado deve incluir um módulo de definições do SandiaXTP (diretiva #include), e deve ser ligado à biblioteca que contém as definições da MTL (libmtl.a). A definição da interface da classe mtlif é mostrada a seguir.

```
class xtpif:public mtlif {
public:
   xtpif();
   ~xtpif();
   int config(word32 cmd, xtp_config* xcf);
   int reg();
   int bind(address_segment* addr);
   int bind(address_segment* addr, void* tspec);
   int getaddr(address_segment* addr);
   int gettspec(traffic* tspec);
   int puttspec(traffic* tspec);
   int getstate(XTPstate_machine* st);
   int listen(int block, short16* options);
   int listen(int block, short16* options, traffic* tspec);
   int reject(short16 why);
   int send(void* p, int* length, int block, short16* options);
   int send(int block, short16* options);
   int receive(void* p, int* length, int block, short16* options);
   int receive(int* length, int block, short16* options);
   int sendcntl(int block, short16* options);
   int release();
   int write(void* p, int len, short16 options=0);
   int read(void* p, int len);
   void print_options(short16 options, char* s = "");
   void fprint_options(FILE* fd, short16 options, char* s = "");
   void perror(int res, char* usr_msg = (char*)NULL);
}
```

O construtor da classe, que não tem argumentos, aloca as áreas de memória necessárias para a associação e estabelece uma área de memória compartilhada, por onde será feita a troca de dados entre o usuário e o deamon XTP. O destrutor da classe libera os recursos

da associação. A função config() permite que o usuário especifique vários parâmetros da associação, como o tamanho da PDU (Protocol Data Unit), alguns temporizadores e o tamanho de alguns buffers. A função reg() registra o usuário com o daemon XTP. Isto é necessário para que o deamon mantenha o controle sob quais associações estão ativas. A função bind() associa um endereço (que possui vários formatos) ao usuário da conexão. Se o chamador de bind() for o originador da conexão, este endereço especificará o destino da chamada. Se o chamador for o receptor, o endereço será um filtro indicando de qual endereço deverá originar a chamada, para que seja aceita.

As funções getaddr(), gettspec(), puttspec() e getstate() são usadas para informar e modificar os parâmetros da conexão. Alguns desses parâmetros somente podem ser modificados antes de iniciada a transferência de dados, e outros podem ser modificados durante a comunicação. A função listen() faz com que o usuário seja bloqueado à espera de um pacote FIRST (vide seção 3.2.1), que encerra a fase de estabelecimento de conexão e inicia a transferência de dados. A função send(), que possui duas formas, permite que o usuário envie dados já anteriormente transferidos para o buffer de transmissão, ou ainda indicando um ponteiro para o início dos dados. Da mesma forma, a função receive() possibilita o recebimento de dados mantendo-os no buffer de recepção, ou ainda transferindo-os diretamente para uma área do usuário, também especificada por um ponteiro.

A função sendcntl() permite que um usuário envie um pacote de controle ainda não solicitado, por exemplo para testar se o outro lado da conexão ainda está ativo. A função release() libera todos os recursos associados a uma conexão, e retorna a máquina de estados para o estado inicial (quiescent). As funções write() e read() permitem escrever e ler dados nos buffers de transmissão e recepção, respectivamente. Isto permite que o usuário manipule os dados antes de efetivamente acionar o protocolo de transmissão, permitindo uma razoável economia de tempo, principalmente se há vários usuários compartilhando o acesso ao protocolo e à CPU. As funções print_options() e fprint_options() permitem que o usuário visualize os parâmetros da conexão em uma forma textual. A função perror() imprime a descrição de um código de erro retornado pelo protocolo.

O SandiaXTP já passou por várias atualizações e revisões de implementação. Notou-se porém dois problemas com o código. O primeiro deles é o fato de ser um pouco lento, em virtude do número de chamadas de função. Partindo do mais alto nível, até 18 chamadas de função são necessárias para chegar até o driver de rede. O outro problema é a falta de threads. Por executar em um único processo, o protocolo pode ter seu desempenho comprometido quando há vários clientes fazendo chamadas a ele.

6.4 A interface ATM da Sun

A interface com as placas SunATM é feita através de um padrão de interface da Sun, chamado STREAMS. Nesse padrão, temos um driver comum (/dev/sa0), que suporta chamadas IOCTL. Através das chamadas IOCTL é possível enviar e receber dados, além de especificar os parâmetros do tráfego. O driver é compatível com a especificação ATM Forum UNI 3.0, e implementa o protocolo Q.2931 para sinalização.

O driver tem dois modos de operação: raw mode e DLPI mode, que indicam o tipo de encapsulamento usado. No modo raw um único bloco de dados é enviado, contendo o número do VCI (4 bytes) seguido dos dados. A multiplexação no receptor é feita com base no VCI. No modo DLPI (Data Link Provider Interface) são enviados dois blocos, o primeiro contendo o tipo de mensagem DLPI, e o segundo contendo um cabeçalho LLC (Logical Link Control) seguido dos dados. A multiplexação é feita com base no cabeçalho LLC. Por não ser necessária a multiplexação por LLC, optou-se por utilizar o modo raw, definindo-se um VCI padrão para ser usado no XTP (VCI 120).

Os parâmetros de tráfego permitidos pelo driver são: o tipo de conexão (CBR ou VBR), a unidade de alocação de banda (64k ou 1 Mbps), a taxa de pico, a taxa média, o número de bytes em modo burst e a prioridade. Pode-se especificar também se a alocação é para um VCI específico ou para qualquer VCI que usar a interface. Porém, no driver das estações Ultra há um problema com a alocação por VCI, ainda não corrigido na versão 2.0. Para contornar este problema, elaborou-se um esquema com várias aberturas simultâneas do driver, que fazem com que, mesmo usando um só VCI, o tráfego dos diversos clientes não seja "misturado". Há ainda outros problemas na versão 2.0 do driver, que ainda não foram corrigidos. Usando uma dessas falhas, um usuário comum (sem direitos especiais) pode "rebootar" qualquer estação que tenha a placa SunATM. A versão do driver utilizada é baseada em PVC (Permanent Virtual Calls), embora já haja uma atualização para torná-lo compatível com SVC (Switched Virtual Calls).

O maior throughput possível no nível STREAMS é de 134 Mbps. Dos 155 Mbps, 2 Mbps são reservados para controle, e o restante é perdido com cabeçalhos de células e da AAL5. Desses 134 Mbps, a MTLQoS reserva 2 Mbps para controle do protocolo, restando 132 Mbps para os usuários.

6.5 Portando o XTP para ATM

Conforme dissemos anteriormente, o SandiaXTP é baseado na MTL, que trata dos protocolos de rede nos quais o XTP está baseado. Para portar o XTP para ATM, reescrevemos a MTL para torná-la compatível com a AAL5. Chamaremos essa implementação modificada de MTLQoS.

A MTL (Meta Transport Library) é uma biblioteca de classes de comunicação que pode ser utilizada para a implementação de protocolos genéricos, e não somente o XTP. A MTL inclui primitivas para o envio e recebimento de dados, e também para o controle de buffers de transmissão e recepção. A criação de protocolos específicos é feita implementando-se classes derivadas das classes-base definidas. Na MTL também estão definidas funções virtuais, nos casos em que não há informação suficiente para a implementação completa da funcionalidade, que deve ser definida de acordo com o protocolo implementado acima da MTL. Há também funções puras virtuais (pure virtual), que devem ser implementadas pelos protocolos derivados [59].

A comunicação entre o daemon que controla a comunicação na MTL e os usuários é feita através de áreas de memória compartilhada, de acordo com a Figura 6.7. A interface com os usuários é feita através de uma biblioteca, que é incluída em tempo de compilação. Cada contexto controla uma conexão de usuário. O serviço de entrega de dados trata da interface de comunicação com os protocolos nos quais a MTL está baseada, como IP, UDP ou AAL5.

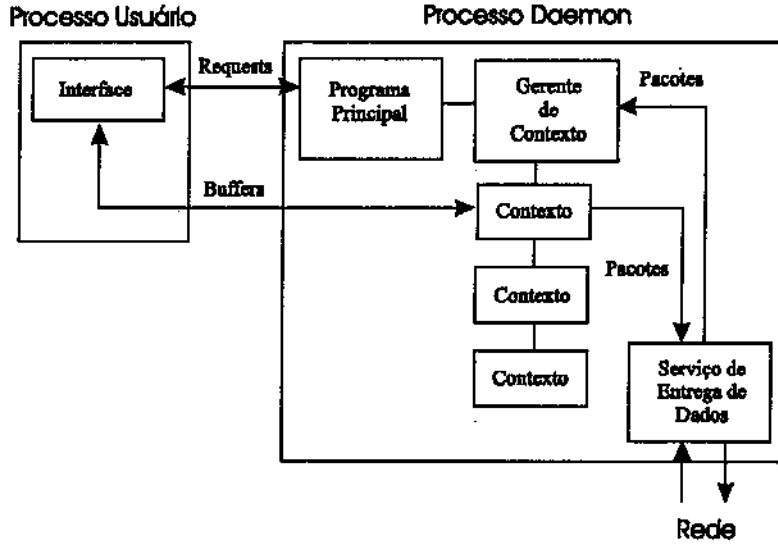


Figura 6.7: Arquitetura da MTL

A primeira decisão de projeto foi estabelecer o tamanho da unidade de transferência do protocolo (MTU). Através da interface STREAMS é possível alocar um buffer de transferência de até 64 Kbytes, mas a MTL originalmente limita o tamanho da MTU em 8 Kbytes. A Figura 6.8 mostra a relação entre o tamanho da MTU e o throughput no nível do driver. Conforme vemos no gráfico, acima de 4096 bytes praticamente não há mudança no throughput, que se estabiliza em aproximadamente 128 Mbps. Porém, no nível do XTP, a MTU guarda uma relação estreita com outros parâmetros, conforme veremos na análise de desempenho. Analisando esses dados em conjunto, definimos uma

MTU variável de 1 a 32 Kbytes, com valor default de 8 Kbytes.

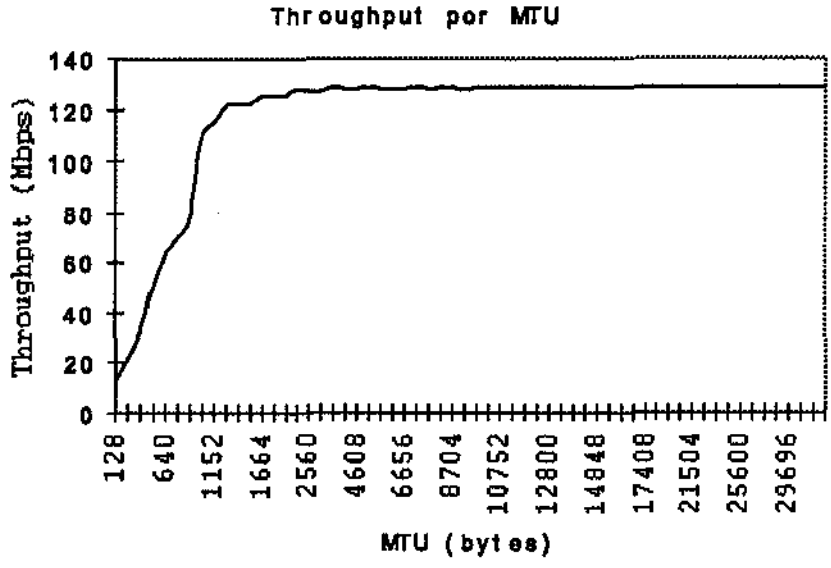


Figura 6.8: Gráfico de throughput por MTU

Outra decisão foi quanto ao endereçamento. O XTP possui um esquema de endereçamento flexível, que permite definir novos formatos. Os formatos de endereço aceitos pelo XTP são mostrados na Tabela 3.1. Desta forma, seria possível criar um novo formato utilizando os endereços NSAP (Network Service Access Point) (ISO 8348 e ITU X.213) implementados no driver, de 20 bytes. Por outro lado, se usarmos endereços IP, ganharemos em portabilidade e facilidade. Por tratar-se de um projeto piloto, e pelo ambiente ser restrito a uma rede local, decidiu-se utilizar endereçamento IP.

Outro problema seria como fazer o mapeamento entre os VCIs utilizados para a transmissão na rede ATM e o endereçamento IP usado no SandiaXTP. Para isto, criamos um esquema de ARP (Address Resolution Protocol) manual. Em cada estação ATM existe um arquivo de configuração, que lista em cada linha o endereço IP seguido do VCI correspondente. Na inicialização do protocolo esta lista é colocada em memória, e usada sempre que se faz um "bind" no XTP.

6.6 Introduzindo QoS no XTP

Todo o mecanismo de QoS introduzido no XTP foi feito através de polimorfismo. Isto é, foram acrescentadas implementações de métodos que incluem parâmetros de QoS. Desta forma, a introdução de parâmetros de QoS não modificou a operação do XTP.

As aplicações já desenvolvidas para o XTP não precisam ser modificadas. Cada aplicação opta por utilizar ou não os parâmetros de QoS. Mostraremos a seguir a estrutura definida para controlar a QoS vinda da rede.

```
typedef struct {
   int handle;        // handle para transmissao
   byte8 type;        // CBRxMEGA, CBRx64, VBRxMEGA, VBRx64
   short16 peak;      // throughput de pico
   short16 average;   // throughput medio
   short16 burst;     // bytes em modo burst
   byte8 priority;    // prioridade (alta ou baixa)
} qos_type;
```

O parâmetro "type" indica se a transmissão será em modo CBR ou VBR, e se a unidade de alocação será de 64 Kbytes ou em megabytes. O parâmetro peak indica o throughput de pico especificado para a conexão, enquanto o parâmetro average indica o throughput médio esperado. O parâmetro burst indica quantos bytes deve-se esperar em cada rajada de tráfego acima da média. O parâmetro priority equivale ao bit CLP das células ATM. Este parâmetro somente suporta os valores 0 (alta prioridade) e 1 (baixa prioridade).

O "bind" de uma conexão XTP é feito passando-se os seguintes parâmetros:

```
int bind(address_segment* addr, qos_type* qos_spec);
```

Se efetuado com sucesso, o "bind" retorna na estrutura um "handler", a ser usado para o envio de dados. A função "send" possui quatro formas, onde duas recebem o "handler" como parâmetro. Mostramos a seguir uma delas.

```
int send(int block, short16* options, int shandle);
```

A função "release" libera a QoS alocada, e também encerra a conexão. Ela tem o seguinte protótipo:

```
int release(int rhandle);
```

Além desses parâmetros de QoS, garantidos pela rede ATM, há outros, que correspondem a parâmetros do protocolo XTP. A estrutura proposta e implementada para controlar todos os parâmetros de QoS é mostrada a seguir.

```
struct qos_param {
     unsigned char InSyncType;
     unsigned short InPeakRate;
```

```
    unsigned short InAverageRate;
    unsigned short InBurst;
    unsigned char InPriority;
    unsigned char InTypeOfService;
    unsigned char InErrorControl;
    unsigned char InFlowControl;
    unsigned char InCheckControl;
    unsigned char InRateControl;
    unsigned short InMaxData;
    unsigned char OutSyncType;
    unsigned short OutPeakRate;
    unsigned short OutAverageRate;
    unsigned short OutBurst;
    unsigned char OutPriority;
    unsigned char OutTypeOfService;
    unsigned char OutErrorControl;
    unsigned char OutFlowControl;
    unsigned char OutCheckControl;
    unsigned char OutRateControl;
    unsigned short OutMaxData;
};
```

A Tabela 6.2 mostra os valores permitidos para cada um dos parâmetros. Para cada um dos parâmetros de entrada existe um correspondente no sentido inverso, com a mesma função. Os primeiros cinco parâmetros (InSyncType, InPeakRate, InAverageRate, InBurst, InPriority) correspondem aos parâmetros que controlam a QoS vinda da rede, já explicados anteriormente.

O parâmetro InTypeOfService controla o paradigma de comunicação a ser utilizado na associação. O modo "connection-oriented" (CONN) implica em usar "three-way handshaking" para estabelecer uma conexão entre as entidades. Além deste modo há um modo de datagrama, onde os pacotes são enviados de forma individual, sem nenhum estabelecimento de sessão. Este modo pode ser feito sem confirmação (UDAT - Unacknowledged Datagram) ou com confirmação (ADAT - Acknowledged Datagram) dos pacotes enviados. Há ainda dois outros modos que não possuem correspondentes nos protocolos convencionais. O modo "transaction" (TRANS) envia transações, que são confirmadas e registradas pelo protocolo, sendo ideal para ambientes de banco de dados. O modo "bulk" é direcionado à transferência de grandes volumes de dados, onde utiliza-se a maior PDU possível, e as confirmações são feitas a cada bloco de dados transmitidos. É ideal para a transmissão de grandes arquivos, como em backup remoto.

Parâmetro	Valores
InSyncType	CBRxMEGA, CBRx64, VBRxMEGA, VBRx64.
InPeakRate	[1-132] Mbps
InAverageRate	[1-132] Mbps
InBurst	número de bytes
InPriority	[0 ou 1]
InTypeOfService	CONN,TRANS,UDAT, ADAT, bulk.
InErrorControl	go-back-n, FASTNAK, selective.
InFlowControl	use-it, dont-use.
InCheckControl	use-it, dont-use.
InRateControl	use-it, dont-use.
InMaxData	[1-32] Kb

Tabela 6.2: Parâmetros de QoS introduzidos no XTP

O parâmetro InErrorControl especifica o tipo de controle de erro a ser utilizado. Pode-se optar entre três técnicas: go-back-n, selective retransmission ou fast negative acknowledgement (FASTNAK). A técnica go-back-n (retorne ao "n") consiste em, quando forem detectados erros nos dados, enviar um sinal informando o último número de pacote que foi recebido sem erro. Ao receber esse sinal, o emissor reinicia a transmissão de todos os pacotes a partir do ponto especificado. Esta técnica não dintingüe se os pacotes foram recebidos com erro ou simplesmente perdidos. Na técnica "selective retransmission" (retransmissão seletiva) é enviado um pacote de controle indicando exatamente quais pacotes foram recebidos com erro, e somente estes são reenviados. Esta técnica envia menos dados em caso de erro, porém implica em um maior processamento por parte do protocolo. A técnica de "Fast Negative Acknowledgement", que é uma técnica de controle agressivo de erros, implica que o receptor vai notificar o emissor imediatamente após perceber que um pacote foi perdido ou recebido com defeito. Nas duas técnicas anteriores, a ocorrência de erros só é informada quando requisitada através dos bits SREQ e DREQ. Pode-se também optar por não utilizar nenhuma técnica de controle de erros, deixando esta tarefa para os níveis superiores.

O parâmetro InFlowControl especifica se o controle de fluxo será utilizado. No XTP o controle de fluxo é baseado em janelas deslizantes controladas por números de seqüência, e é realizado fim-a-fim. Um número de seqüência é associado a cada byte transmitido, e o controle das janelas é feito no receptor. O parâmetro InCheckControl especifica se será utilizado um checksum no campo de dados dos pacotes (o cabeçalho é sempre protegido por um checksum). Pode-se optar por não utilizar o checksum caso o tipo de dado sendo transmitido não exija um controle estrito, por exemplo na transmissão de

áudio sem compactação. O parâmetro InRateControl especifica se o controle de taxa será utilizado. No XTP o controle de taxa é feito através de fichas (tokens). Um número máximo de bytes por segundo a transmitir é negociado no estabelecimento da sessão, e o valor correspondente em fichas é gerado a intervalos regulares.

Todos esses parâmetros de QoS são passados ao XTP através de duas funções: "configure_session" e "configure_traffic". A primeira controla os parâmetros de sessão, que são estabelecidos antes do bind, e a segunda controla os parâmetros de tráfego, que são estabelecidos depois do "bind" e antes da transferência dos dados.

6.7 O Objeto de Negociação de QoS

O Objeto de Negociação de QoS (ONQoS) foi implementado usando-se RPC (Remote Procedure Call), com uma interface compatível com a CORBA-IDL. O ONQoS possui uma interface simples, com funções para incluir, modificar e retirar serviços. A chave de busca é sempre a referência do objeto.

Um ONQoS deve existir em cada host servidor. O objeto servidor cadastra seus serviços no ONQoS, que podem ser modificados, conforme são criadas novas versões. As tabelas de serviços são mantidas em memória. Nenhuma função de persistência foi criada, embora sua implementação seja simples. A interface do ONQoS é feita através das seguintes funções:

```
int   *register_service(struct srecord *ptr);
int   *drop_service(unsigned long *objref);
int   *change_service(struct srecord *ptr);
struct srecord *give_qos(unsigned long *objref);
struct srecord {
   unsigned long   ObjRef;
   unsigned char   deleted;
   qos_param       rep_qos;
}
```

Essas funções incluem, removem, modificam e pesquisam serviços, respectivamente. É mostrada também a estrutura "srecord". O campo ObjRef guarda a referência do objeto, que varia de formato de acordo com a plataforma ORB utilizada. Ela foi definida como um "long" sem sinal (64 bits). O campo "deleted" ajuda a remover os serviços da memória. A estrutura "qos_param", como vimos anteriormente, guarda os parâmetros de QoS.

6.8 Análise de Desempenho

Nesta seção analisaremos o desempenho do XTP com QoS (XTPQoS) em uma rede ATM. Conforme dissemos anteriormente, o protocolo XTP é bastante complexo. A implementação SandiaXTP é composta de mais de 20.000 linhas de código C++, e com muitas chamadas de função. O fato do protocolo executar em um único processo também contribui para os problemas. Apesar disto, como veremos adiante, o desempenho mostrou-se satisfatório, principalmente por causa das estações de alta velocidade utilizadas. Comparando-se os resultados com os requisitos para transmissão de aplicações multimídia, como vídeo MPEG, vemos que é perfeitamente possível sua utilização. Veremos a seguir alguns gráficos de desempenho.

A Figura 6.9 mostra a relação entre a banda alocada e o "throughput" efetivamente obtido no XTPQoS. São mostrados uma conexão em modo não-bloqueante, onde o envio de dados é assíncrono em relação às confirmações de recepção, e uma conexão bloqueante, onde cada pacote só é enviado depois do pacote anterior ter sido confirmado. Utilizamos no exemplo pacotes de 32 bytes. Vemos que no exemplo não-bloqueante o overhead do protocolo é mínimo, e o throughput cresce linearmente até um máximo aproximado de 110 Mbps. No caso bloqueante, o limitante é a espera pelas confirmações, que restringe o throughput a aproximadamente 40 Mbps.

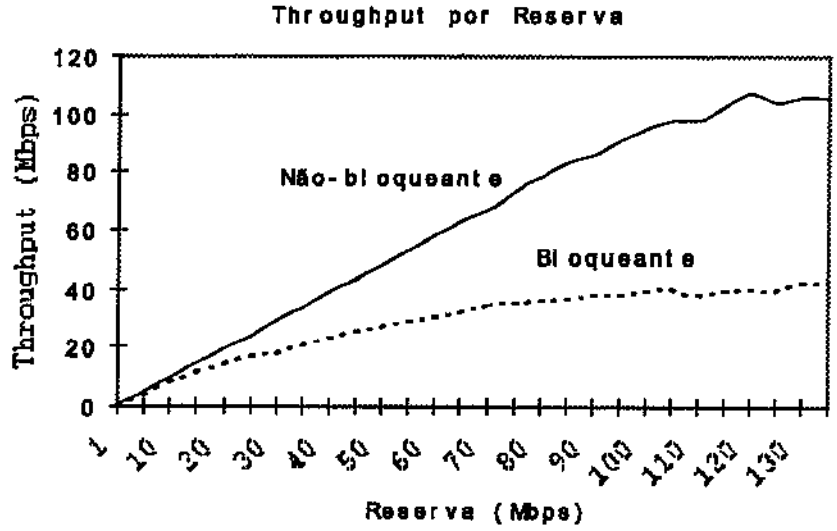


Figura 6.9: Gráfico de Reserva por Throughput

Na Figura 6.10 vemos a relação entre o throughput e a MTU. Vemos que, quanto maior a MTU, maior é o throughput. Com uma MTU pequena, o overhead de troca de

mensagens e de processamento de protocolo passa a ser mais significativo, e temos um menor throughput.

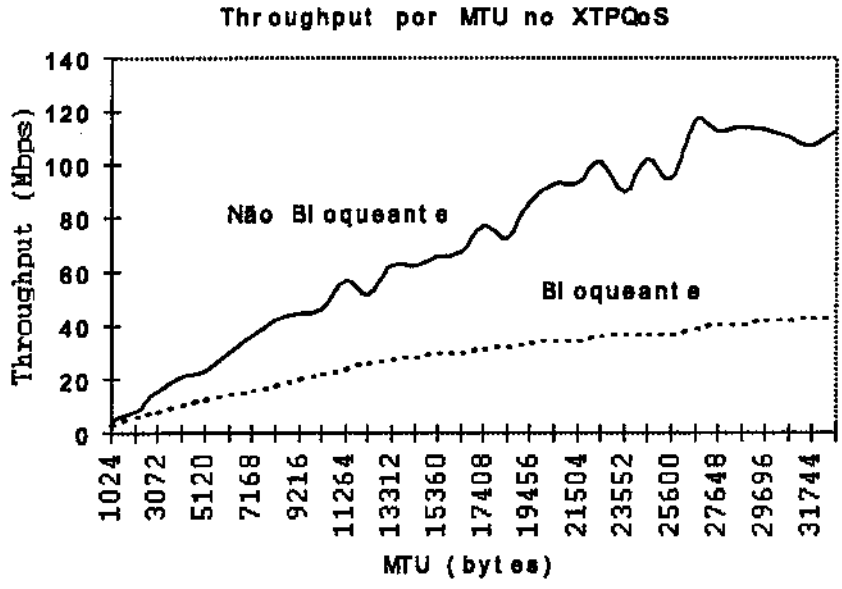


Figura 6.10: Gráfico de Throughput por MTU

A Figura 6.11 mostra a relação entre o atraso e o tamanho dos pacotes no modo bloqueante. Aqui vemos que o atraso cresce com o tamanho dos pacotes. O atraso máximo apresenta uma grande variação, que atribuimos ao escalonamento do processador durante o período de transmissão. Porém, o atraso médio cresce linearmente, chegando ao máximo de 5 ms. No modo não-bloqueante o atraso médio é ainda menor, não passando de 2 ms (Figura 6.12).

Na Figura 6.13 vemos a relação entre o jitter e o tamanho do pacote em modo bloqueante. Podemos ver que, da mesma forma que o atraso (e pela mesma causa), a variação do jitter máximo é bem grande, porém o valor médio é baixo, menos de 0.2 ms. A variação do jitter pode ser facilmente compensada usando-se buffers de tamanho compatível. No modo não-bloqueante, mostrado em seguida, vemos que o jitter médio é ainda menor, com o jitter máximo também apresentando uma grande variação (Figura 6.14).

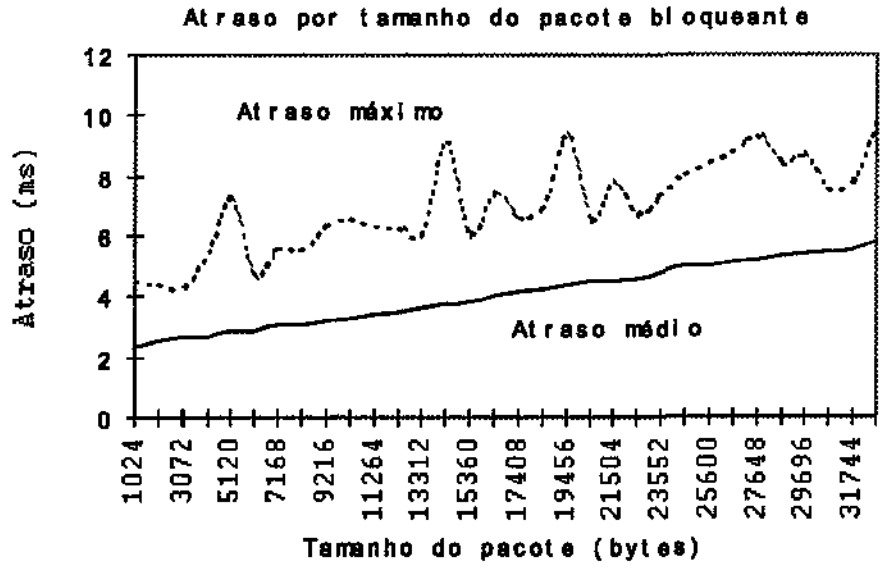


Figura 6.11: Gráfico de Atraso por tamanho do pacote bloqueante

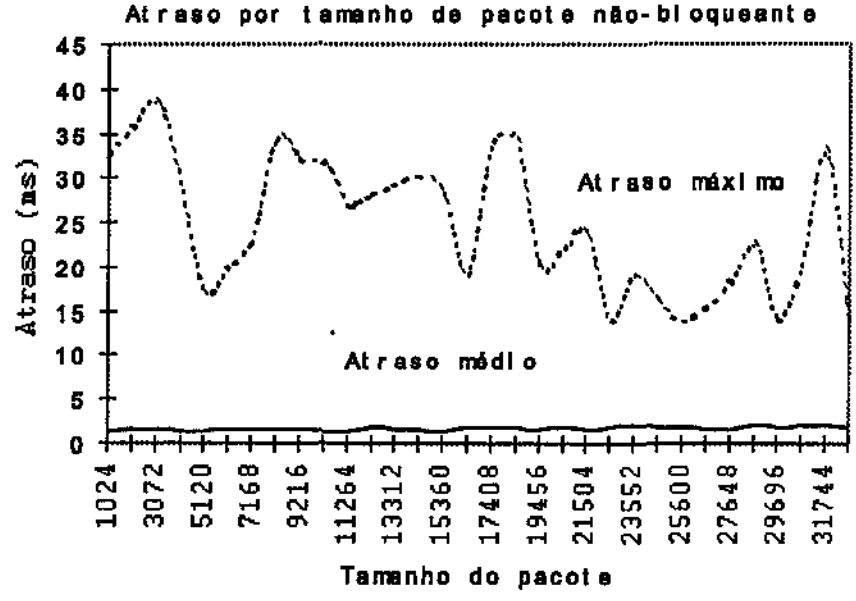


Figura 6.12: Gráfico de Atraso por tamanho do pacote não-bloqueante

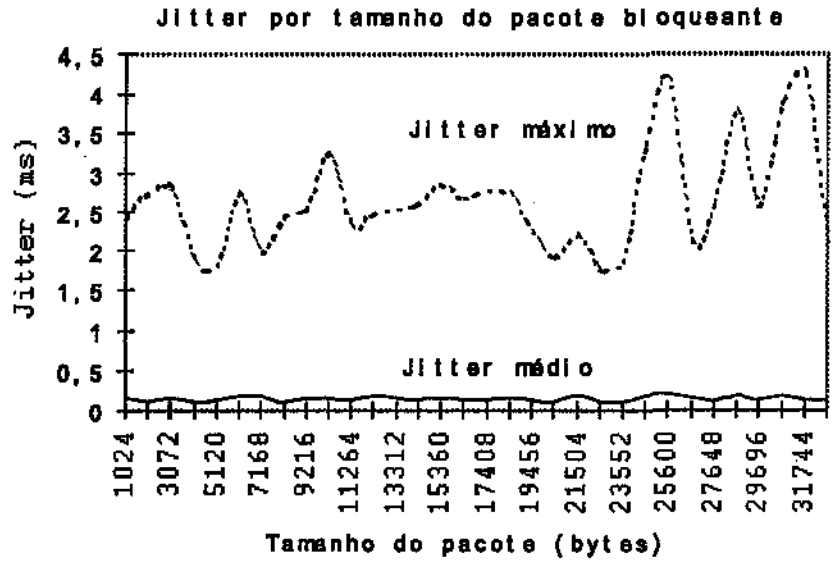


Figura 6.13: Gráfico de Jitter por tamanho do pacote bloqueante

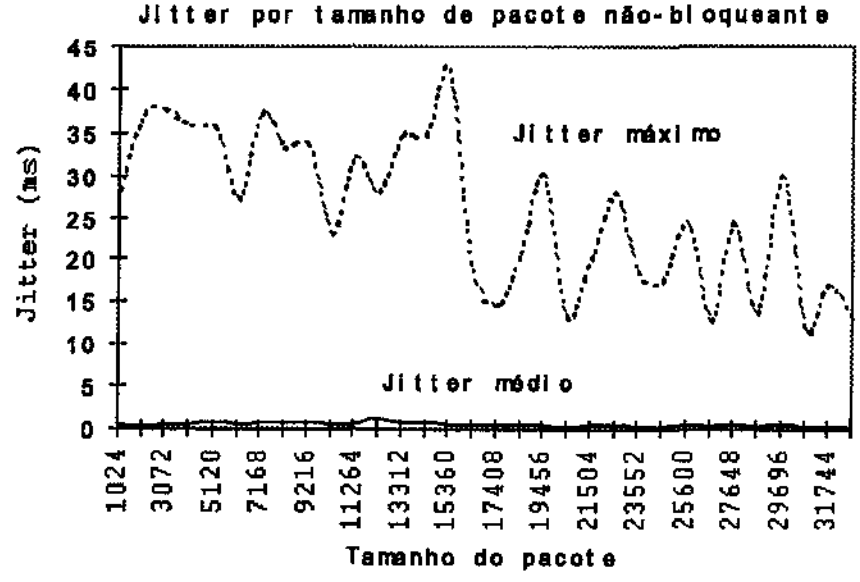


Figura 6.14: Gráfico de Jitter por tamanho do pacote não-bloqueante

Capítulo 7

Conclusão

O desenvolvimento de sistemas computacionais distribuídos tem passado por grandes mudanças nos últimos anos. O aparecimento de redes de computadores cada vez mais rápidas propiciou a criação de aplicações complexas, que lidam com diversas mídias e possuem requisitos bastante estritos em relação ao ambiente em que executam. Com isto, além da preocupação óbvia quanto à funcionalidade da aplicação, cresceu a preocupação com os aspectos não-funcionais, que formam a Qualidade de Serviço. De acordo com esse novo ambiente surgiram modelos de computação distribuída baseada em objetos, como o RM-ODP e a arquitetura CORBA, que endereçam esses requisitos, embora ainda de maneira abstrata. Nesse contexto surgiu a plataforma Multiware, como um esforço para criar uma plataforma operacional, compreendendo múltiplas camadas de protocolos e baseada em padrões fortes, que auxiliasse o desenvolvimento de aplicações distribuídas baseadas em objetos.

Este trabalho compreendeu primeiramente um estudo sobre a Qualidade de Serviço em suas diversas nuances. A primeira parte desse estudo foi feita nos modelos e arquiteturas nas quais a Multiware está baseada (RM-ODP e CORBA). Como resultado disto, constatou-se a inexistência de padrões para muitos aspectos da QoS, particularmente nos requisitos e na negociação de QoS em ambientes distribuídos baseados em objetos. Nesse ponto foi criada uma base conceitual na qual o restante do trabalho está assentado, utilizando para isto os conceitos dos trabalhos em andamento do ISO e da ITU, e juntando-se a eles conceitos e contextualizações próprias, onde não havia nada padronizado.

Uma segunda parte deste estudo focou nos protocolos de comunicação, de reserva de recursos e de roteamento, e também na infra-estrutura das redes hoje existentes, identificando e analisando os mecanismos de QoS implementados, com o objetivo de selecionar um ambiente alvo onde situar as etapas seguintes do projeto. Este estudo mostrou haver inúmeras propostas e muito pouco consenso para uma implementação de mecanismos de QoS em larga escala. Parte dessa falta de consenso deve-se à administração da Internet.

Por sua importância indiscutível, qualquer padrão que queira tornar-se "utilizável" deve passar pela Internet, onde há vários organismos criando padrões, muitas vezes conflitantes. Outra constatação é que os novos protocolos com mecanismos de QoS têm sido associados às redes de alta velocidade, e pode-se prever um aumento em sua disseminação associado ao aumento das redes ATM, por exemplo. Por este motivo, a adoção de mecanismos de QoS em larga escala ainda tem sido restrita a projetos específicos, como os projetos ESPRIT (European Strategic Programme for R&D in Information Technology), BERKOM (Berlin Communication System), RACE (R&D Advanced Communications Technologies in Europe) e BAGNet (Bay Area Gigabit Network).

Como resultado dessa etapa de estudos, além de um "arcabouço" de definições e contextualizações, obteve-se um ambiente-alvo para as próximas etapas, com a escolha de uma infra-estrutura de rede (arquitetura ATM), um protocolo onde implementar os mecanismos de QoS (XTP) e um ambiente de computação distribuída (plataforma Multiware, baseada em CORBA e ODP).

A etapa seguinte consistiu na criação de um modelo de Qualidade de Serviço para esse ambiente. A primeira parte do modelo consistiu em definir os requisitos e as métricas de QoS adequadas. O passo seguinte consistiu em criar um protocolo de negociação de QoS, que foi dividido em duas partes, distingüindo assim os requisitos de alto nível (semi-estáticos, associados ao ambiente) dos requisitos de baixo nível (dinâmicos, associados à infra-estrutura de rede). Para esses dois conjuntos de requisitos foram associados dois elementos negociadores, o Trader e o Objeto de Negociação de QoS, e foram definidos os protocolos de negociação. Em seguida foram definidos os mecanismos de tradução dos requisitos e a interação com os protocolos de transporte, de rede e de enlace.

Após criado o modelo, seguiu-se uma etapa de implementação. Esta etapa abrangeu desde a configuração dos switchers ATM até a implementação dos objetos distribuídos, passando por protocolos de enlace, de rede e de transporte. Partindo-se de uma implementação genérica de classes de protocolo em nível de rede, adaptou-se a mesma à utilização com AAL5 e com os mecanismos de QoS das redes ATM. Uma implementação do protocolo XTP foi estendida (sem ferir a especificação do protocolo), acrescentando-se a ela a possibilidade de trabalhar com redes ATM, os mecanismos e o protocolo de negociação de QoS. Como última etapa de implementação, um objeto de negociação de QoS foi implementado. Finalmente, foram criados alguns exemplos de aplicações típicas (transferência de arquivos, transmissão de frames isócronos, etc) para testar a viabilidade do protótipo.

7.1 Resultados obtidos

Como principais contribuições deste projeto pode-se destacar:

- Um modelo de especificação, tradução, negociação e monitoração de Qualidade de Serviço para ambientes distribuídos orientados a objeto, baseado no modelo ODP e na arquitetura CORBA;
- Uma implementação do protocolo XTP para redes ATM baseadas em AAL5, incluindo a reserva de recursos;
- A criação de um protótipo operacional que implementa a Qualidade de Serviço em todos os níveis de protocolos especificados, desde a interface de aplicação até a rede ATM, passando pelos protocolos de rede e de transporte.

7.2 Extensões ao trabalho

Devido às restrições de tempo sempre presentes em trabalhos desta natureza, e devido também às restrições de ambiente, alguns aspectos não foram suficientemente trabalhados, e serão deixados como possíveis extensões.

A primeira, e uma das mais importantes, é a renegociação dinâmica de QoS no ambiente distribuído. Conforme vimos em muitos pontos do texto, há vários motivos pelos quais tanto o ambiente quanto a aplicação podem necessitar rever o contrato de QoS no decorrer da comunicação, abrindo um processo de renegociação. Em um ambiente distribuído, e que planeja suportar aplicações de multimídia e de tempo real, a renegociação envolve muitos aspectos complicados. Tem-se que considerar a interrupção da aplicação, o impacto no sincronismo da comunicação, a interface de renegociação, como modificar as reservas de recursos, e vários outros aspectos. Até o momento não há nenhuma padronização neste tópico.

Outro importante aspecto não suficientemente estudado é a monitoração de QoS. Em um ambiente distribuído, onde são providas transparências, como de localização e de relocação, a determinação de pontos de monitoração se torna uma tarefa não trivial. Outro problema é que protocolo utilizar para a monitoração, se um protocolo já padronizado, como SNMP ou RMON, ou um protocolo especialmente desenhado para ambientes distribuídos orientados a objeto. Outra alternativa que se apresenta é o uso do modelo clássico agente-gerente, em oposição a um modelo de entidades pares. Outra questão é se a coleta de dados deve ser feita no núcleo da ORB, no adaptador de objetos, nos elementos do canal de comunicação ou em uma interface da própria aplicação. Todas essas são questões que merecem ser estudadas. Novamente, não há ainda padrões concretos neste tópico.

Ainda outra questão diz respeito ao uso do Trader para a negociação de requisitos de QoS de alto nível. Isto foi proposto neste trabalho, porém utilizando interfaces genéricas do Trader, e não uma interface específica. Outro trabalho dentro do projeto Multiware

implementou um Trader, porém não se discutiu seu uso específico para armazenar, traduzir e negociar parâmetros de QoS.

Outra possível extensão do trabalho é em relação ao seu emprego nas redes ATM. Simultaneamente à implementação do protótipo, que utiliza a UNI 3.0, surgiu a especificação UNI 4.0, que complementa muitos aspectos antes não inteiramente definidos, como o protocolo de sinalização e a P-NNI (Private Network-to-Network Interface). Esta nova especificação, quando adotada, disponibilizará muitos mecanismos no nível ATM antes inexistentes, e com interoperabilidade entre diversos fabricantes. Uma possível extensão do protótipo seria a implementação da UNI 4.0, e sua implantação em um ambiente ATM heterogêneo e de maior escala. Nesse ambiente seria possível testar os limites dos protocolos e das plataformas distribuídas, testando aplicações reais, de alto tráfego e com muitos participantes, como a distribuição de vídeo interativo.

7.3 Considerações finais

O ambiente envolvido nesta pesquisa, particularmente a interação entre Qualidade de Serviço, plataformas distribuídas e redes de alta velocidade apresenta enormes desafios, principalmente por serem temas novos, onde estão surgindo novos padrões a todo momento. Acredita-se que nos próximos anos serão vistas muitas pesquisas visando uma integração entre esses componentes, que juntos representam algumas das principais tendências na área de redes de computadores, e por isso atraem a atenção de toda a indústria de informática. Esses temas devem ser alguns dos principais componentes da "super-estrada da informação", que está sendo moldada a partir da atual Internet, e que promete revolucionar a forma como as pessoas interagem, aprendem, trabalham e se divertem.

Bibliografia

[1] Cristina Aurrecoechea, Andrew Campbell e Linda Hauw. A Survey of QoS Architectures. 4th IFIP International Workshop on Quality of Service, 1996.

[2] M. Bogen, H. Hausen e Rainer Worst. Handling of QoS Characteristics.

[3] M. Borden, E. Crawley, B. Davie e S. Batsell. Integration of real-time services in an IP-ATM network architecture. RFC 1821, Agosto de 1995.

[4] M. Borden et al. Integration of Real Time Services in an IP-ATM Network Architecture. RFC 1821, Agosto de 1995.

[5] R. Braden, D. Clark e S. Shenker. Integrated services in the Internet architecture: an overview. RFC 1633, Junho de 1994.

[6] Andrew T. Campbell. A Quality of Service Architecture. Tese de Doutorado, Lancaster University, 1996.

[7] Andrew Campbell e Geoff Coulson. Experiences with an Adaptive Multimedia Transport System in a QoS Architecture. University of Columbia, New York, NY, Janeiro de 1996.

[8] Andrew Campbell e Geoff Coulson. Implementation and Evaluation of the QoS-A Transport System. Proc. 5th IFIP International Workshop on Protocols for High Speed Networks, 1996.

[9] Andrew Campbell, Geoff Coulson e David Hutchison. A Quality of Service Architecture. IEEE INFOCOM, 1993.

[10] Andrew Campbell, Geoff Coulson e David Hutchison. Supporting Adaptive Flows in a Quality of Service Architecture. Multimedia Systems Journal, Special Issue on QoS Architecture, 1998.

[11] Andrew Campbell, Geoff Coulson, Francisco Garcia, David Hutchison e Helmut Leopold. Integrated Quality of Service for Multimedia Communications. IEEE INFOCOM, 1993.

[12] Douglas E. Comer. Internetworking with TCP/IP volume I. Prentice Hall, Janeiro de 1995.

[13] Ladislau Conceição. Serviços de Comunicação para a Plataforma Multiware. Tese de Mestrado, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica, 1995.

[14] Fábio Moreira Costa. Suporte a Grupos Cooperativos em Ambiente Distribuído Aberto. Tese de Mestrado, Universidade Estadual de Campinas, Instituto de Computação, 1996.

[15] F.M. Costa e E.R.M. Madeira. An Object Group Model and its Implementation to Support Cooperative Applications on CORBA. IFIP/IEEE ICDP, Fevereiro de 1996.

[16] F.M. Costa e E.R.M. Madeira. Suporte a grupos cooperativos na plataforma Multiware. XXII Software and Hardware Symposium / PANEL 95, Agosto de 1995.

[17] L. Delgrossi e L. Berger. Internet Stream Protocol Version 2 (ST2). RFC 1819, Junho de 1994.

[18] Sudheer Dharanokota e Kurt Maly. Design of QUANTA and an Evaluation Methodology. Old Dominion University, Old Dominion University technical report, 1995.

[19] Sudheer Dharanokota e Kurt Maly. Providing predictable performance to a network-based application over ATM. Old Dominion University, Old Dominion University technical report, 1995.

[20] Sudheer Dharanokota e Kurt Maly. QUANTA: Quality of Service Architecture for Native TCP/IP over ATM networks. Old Dominion University, Old Dominion University technical report, 1995.

[21] Linda Fedaoui, Aruna Seneviratne, Anne Fladenmuller e Eric Horlait. Implementation of a End-to-End Quality of Service Management Scheme, 1994.

[22] Domenico Ferrari et al. Network Support for Multimedia. University of California at Berkeley, Berkeley, CA.

[23] Anne Fladenmuller, Aruna Seneviratne e Eric Horlait. A Hybrid QoS Management Scheme for Distributed Multimedia Applications. Second workshop on PROMS, 1995.

[24] Anne Fladenmuller, Aruna Seneviratne e Eric Horlait. A Hybrid QoS Management Scheme. Second Workshop on High Performance Protocol Architecture, HIPPARCH95, 1995.

[25] Anne Fladenmuller, Eric Horlait e Aruna Seneviratne. QoS Management Scheme for Multimedia Applications in Best Effort Environments. Journal of Electrical and Electronics Engineering, 1996.

[26] Patricia G.S. Florissi. QoSME: QoS Management Environment. Tese de Doutorado, Columbia University, 1996.

[27] Patricia G.S. Florissi e Yechiam Yemini. Management of Application Quality of Service. Columbia University, New York, NY, 1994.

[28] A. Hafid e G. V. Bochmann. An Approach to Quality of Service Management for Distributed Multimedia Applications. Université de Montreal, Montreal, Canada.

[29] A. Hafid e Jan de Meer. Quality of Service Verification Experiments. MMNet95 - International IEEE Conference on Multi Media and Networking, 1995.

[30] Juha Heinanen. Multiprotocol Encapsulation over ATM Adaptation Layer 5. RFC 1483, Julho de 1993.

[31] David Hutchison, Geoff Coulson, Andrew Campbell e Gordon S. Blair. Quality of Service Management in Distributed Systems. Network and Distributed Systems Management, 1995.

[32] International Standardization Organization. SC21/N 10339 QoS Framework (DIS 13236). Draft International Standard, 1997.

[33] International Standardization Organization. SC21/N 10773 QoS Methods and Mechanisms DTR 13243. Draft International Standard, 1997.

[34] ISO/IEC JTC1/SC21. Basic Reference Model ODP - Part 1: Overview and Guide to Use, Junho de 1995.

[35] ISO/IEC JTC1/SC21. Basic Reference Model ODP - Part 2: Descriptive Model, Janeiro de 1995.

[36] ISO/IEC JTC1/SC21. Basic Reference Model ODP - Part 3: Prescriptive Model, Janeiro de 1995.

[37] B. Kerherve, A. Vogel et al. On Distributed Multimedia Presentational Applications: Functional and Computational Architecture and QoS Negotiation. Fourth International Workshop on Protocols for High-Speed Networks, Honolulu, pp. 1-17, 1994.

[38] Flávio H. de S. Lima e Edmundo R.M. Madeira. ODP-based QoS Specification for the Multiware Platform. IV International Workshop on Quality of Service, pp. 45-54, Paris, Maro de 1996.

[39] Flávio H. de S. Lima e Edmundo R. M. Madeira. Qualidade de Serviço baseada em ATM para a Plataforma Multiware. XV Simpósio Brasileiro de Redes de Computadores - SBRC 97, pp. 366-382, São Carlos, Maio de 1997.

[40] L.A.L. Lima Jr e E.R.M. Madeira. A Model for a Federative Trader. IFIP ICODP '95, pp. 155-166, Brisbane, Australia, 1995.

[41] W.P.D.C. Loyolla, E.R.M. Madeira et al. Multiware Platform: an Open Distributed Environment for Multimedia Cooperative Applications. IEEE COMPSAC, Taipei, Taiwan, Novembro de 1994.

[42] W.P.D.C. Loyolla, I.R. Fontes, F.C. Sica e M.J. Mendes. Plataforma Multiware: Especificação da Camada de Suporte para Aplicações CSCW. 12o Simpósio Brasileiro de Redes de Computadores, Maio de 1994.

[43] Edmundo R. M. Madeira. Multiware Platform: Some Issues about the Middleware Layer. 7th IASTED - International Conference on Parallel and Distributed Computing and Systems, pp. 162-166, Outubro de 1995.

[44] Jan de Meer. On the specification of End-to-End QoS Control. 5th IWQoS - International IFIP Workshop on Quality of Service, 1997.

[45] Jan de Meer. QoS Control by Balancing Continuous MuMe Streams. GI/ITG FG - National GI Meeting on Formal Methods, 1997.

[46] Jan de Meer. Towards a Model of QoS Binding. Presentation at the Mannheim Workshop on QoS, 1998.

[47] M.J. Mendes e E.R.M. Madeira. Plataforma Multiware: Projeto e Desenvolvimento da Camada Middleware. 12o Simpsio Brasileiro de Redes de Computadores, Maio de 1994.

[48] Danny J. Mitzel et al. An architectural comparison of ST-II and RSVP.

[49] Object Management Group. Object Services Architecture, Janeiro de 1994.

[50] Object Management Group. Quality of Service (QoS) Green Paper, Junho de 1997.

[51] Object Management Group. The Common Object Request Broker: Architecture and Specification, 1993.

[52] Martin De Prycker. Asynchronous Transfer Mode - Solution for Broadband ISDN. Ellis Horwood, Janeiro de 1993.

[53] J.A.G. Queiroz e E.R.M. Madeira. Management of CORBA Objects Monitoring for the Multiware Platform. Livro: Open Distributed Processing and Distributed Platforms - Chapman & Hall (Proceedings IFIP/IEEE International Conference on Open Distributed Processing and Distributed Platforms), pp. 122-133, Toronto, Canadá, Maio de 1997.

[54] G. Raeder e S. Mazaher. Quality-of-Service Directed Targeting Based on the ODP Engineering Model. ICODP '95, 1995.

[55] J. Ramaekers e G. Ventre. Quality-of-Service Negotiation in Real-Time Communication Network. University of California at Berkeley, Berkeley, CA, Abril de 1992.

[56] K.A. Raymond. Reference Model of Open Distributed Processing: A Tutorial. ICODP 93, pp. 3-14, Fevereiro de 1993.

[57] Axel Rennoch, Jan de Meer e Joerg Burmeister. Formal Approach to QoS Specification and Verification. CASCON Conference, 1993.

[58] J. Runbaugh, M. Blara, W. Premerlani, F. Eddy e W. Lorensen. Object Oriented Modeling and Design. Prentice Hall, 1991.

[59] Sandia National Laboratory. Meta Transport Library User's Guide, Setembro de 1995.

[60] Sandia National Laboratories. SandiaXTP Reference Manual, Setembro de 1995.

[61] Sandia National Laboratories. SandiaXTP User's Guide, Outubro de 1995.

[62] A. Schill, C. Mittasch, T. Hutschenreuther e F. Wildenhain. A Quality of Service Abstraction Tool for Advanced Distributed Applications. Dresden University of Technology, Dresden, Alemanha.

[63] H. Schulzrinne et al. RTP: A Transport Protocol for Real Time Applications. RFC 1889, Janeiro de 1996.

[64] A. Seneviratne, M. Fry, V. Withana, V. Saparamdu, A. Richards e E. Horlait. Quality of Service Management for Distributed Multimedia Applications. IEEE Communication Society, Phoenix Conference on Computer and communications, 1994.

[65] S. Shenker, C. Partridge e R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212, Setembro de 1997.

[66] W.T. Strayer e Alfred C. Weaver. Is XTP Suitable for Distributed Real-Time Systems? University of Virginia, Charlottesville, Virginia, 1995.

[67] C. Szyperski e G. Ventre. A Characterization of Multi-Party Interactive Multimedia Applications. University of California at Berkeley, Berkeley, CA, Janeiro de 1993.

[68] The ATM Forum. User-Network Interface Specification 3.0. PTR Prentice Hall, Setembro de 1993.

[69] V. Tschammer, M.J. Mendes, E.R.M. Madeira, W.L. Souza e W.P.D.C. Loyolla. Processamento Distribuído aberto e o modelo RM-ODP / ISO. 11o Simpósio Brasileiro de Redes de Computadores, Julho de 1993.

[70] Steve Vinoski. Distributed Object Computing with CORBA. C++ Report Magazine, Julho de 1993.

[71] A. Vogel, B. Kerherve, G.V. Bochmann e J. Gecsei. Distributed Multimedia Applications and Quality of Service - a Survey. IEEE Multimedia, 2(2):10-19, 1995.

[72] A. Vogel, G.V. Bochmann et al. On QoS Negotiation in Distributed Multimedia Applications. Université de Montreal, Montreal, Canada.

[73] Nicholas Yeadon, Francisco Garcia, Andrew Campbell e David Hutchison. QoS Adaptation and Flow filtering in ATM Networks. 2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures, 1994.

[74] J. Wroclawski. The Use of RSVP with IETF Integrated Services. RFC 2210, Setembro de 1997.

[75] XTP Forum. Xpress Transport Protocol Specification revision 4.0. Maro de 1995.

[76] Lixia Zhang et al. RSVP: A New Resource Reservation Protocol. IEEE Network magazine.

[77] L. Zhang, S. Berson, S. Herzog e J. Wroclaswki. Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification, 1995.

[78] Simon Znaty, Spiridon Asenis e Jean Sclavos. ODP Viewpoints of QoS Management Application. France TELECOM, Paris, Frana, 1995.