

UNIVERSIDADE ESTADUAL DE CAMPINAS

**FACULDADE DE ENGENHARIA ELÉTRICA E DE
COMPUTAÇÃO**

**DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
E AUTOMAÇÃO INDUSTRIAL**

Arquitetura de Conexão de Dispositivos Multimídia em Ambientes Distribuídos

Lucimara Desiderá

Orientador: Prof. Dr. Maurício F. Magalhães

Dissertação de Mestrado

Este exemplar corresponde a redação final da tese defendida por <u>Lucimara Desiderá</u> e aprovada pela Comissão Julgada em <u>16 / 07 / 97</u> <u>Maurício F. Magalhães</u> Orientador

Campinas - SP Brasil

1997



016023

UNIDADE	BC
N.º CHAMADA:	UNICAMP
	D46a
V	Ex.
TUNDO BC/	34632
PROC.	395/9.8
C	<input type="checkbox"/>
D	<input checked="" type="checkbox"/>
PREÇO	R\$ 11,00
DATA	04/08/98
N.º CPD	

CM-00113977-9

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

D46a

Desiderá, Lucimara

Arquitetura de conexão de dispositivos multimídia em ambientes distribuídos / Lucimara Desiderá.--Campinas, SP: [s.n.], 1997.

Orientador: Maurício F. Magalhães.

Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento eletrônico de dados - Processamento distribuído. 2. Programação orientada a objetos (Computação). 3. Redes de computação. I. Magalhães, Maurício F. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

UNIVERSIDADE ESTADUAL DE CAMPINAS

**FACULDADE DE ENGENHARIA ELÉTRICA E DE
COMPUTAÇÃO**

**DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
E AUTOMAÇÃO INDUSTRIAL**

Arquitetura de Conexão de Dispositivos Multimídia em Ambientes Distribuídos

Lucimara Desiderá

Orientador: Prof. Dr. Maurício F. Magalhães

Dissertação de Mestrado apresentada
à Faculdade de Engenharia Elétrica
como parte dos requisitos exigidos
para obtenção do título de Mestre em
Engenharia Elétrica, Área de
concentração: Automação Industrial

Campinas - SP - Brasil
1997

*Dedico este trabalho aos meus pais,
Sivanil e Dusolina, e à minha irmã
Cassiane, pela presença constante e
por acreditarem sempre em mim.*

*"Deus é grande, Deus é forte. Quando
Ele quer, não tem que não queira."*

Ayrton Senna

Agradecimentos

Aos meus pais, por me compreenderem e me apoiarem em todos os momentos.

À minha irmã, Cassiane, pelo carinho e cuidado no difícil começo.

Ao Prof. Maurício, pela orientação e confiança em meu trabalho.

Ao Valter, por ser um amigo verdadeiro.

Ao Francisco, por ter me ajudado a abrir horizontes profissionais.

Ao Marcelo, por sua compreensão, sua calma, por seu entusiasmo e por seu amor.

À Adela, pelo companheirismo sem igual.

Ao Renato, pelas dicas nos momentos certos.

À Sílvia, pela prontidão e por todas as caronas.

Aos amigos do LCA, Cláudia, Paulinho, Dino, Lucci, Jussara, Gonzaga, Raquel, Bruno, Letícia, Fabiano, entre tantos outros que fizeram de minha passagem por aqui muito mais do que puro trabalho. Vocês me ensinaram uma lição de vida.

Ao CNPq pelo apoio financeiro.

Sumário

AGRADECIMENTOS.....	I
SUMÁRIO	I
LISTA DE FIGURAS.....	III
LISTA DE ABREVIACÕES	IV
RESUMO	V
ABSTRACT	VI
1. INTRODUÇÃO.....	1
1.1. CONTEXTO	1
1.2. MOTIVAÇÃO	3
1.3. OBJETIVOS.....	3
1.4. ORGANIZAÇÃO.....	4
2. COMPUTAÇÃO MULTIMÍDIA.....	5
2.1. O QUE É MULTIMÍDIA	5
2.2. APLICAÇÕES MULTIMÍDIA	5
2.3. TIPOS DE MÍDIA	7
2.4. SISTEMAS MULTIMÍDIA DISTRIBUÍDOS	7
2.4.1. Plataformas Para Sistemas Distribuídos.....	7
2.4.2. Requisitos das Aplicações Multimídia Distribuídas	8
2.5. SOFTWARE MULTIMÍDIA.....	11
2.5.1. Multimídia Orientada a Objetos	11
2.5.2. Frameworks para Multimídia	12
2.6. CORBA	15
2.6.1. Introdução.....	15
2.6.2. Estrutura de um ORB na Arquitetura CORBA	16
2.6.3. Requisições em CORBA	18
2.6.4. Adaptador de Objeto	19
2.6.5. IDL: Interface Definition Language	21
2.6.6. O Processo de Desenvolvimento em um Ambiente CORBA	22
3. SERVIÇOS DE SISTEMA MULTIMÍDIA	24
3.1. INTRODUÇÃO	24

3.2. OBJETIVOS E MOTIVAÇÕES DO MSS.....	25
3.3. A ARQUITETURA MSS	26
3.3.1. <i>Visão do Cliente</i>	26
3.3.2. <i>Diagrama de subtipos</i>	28
3.3.3. <i>Ciclo de Vida dos Objetos</i>	30
3.4. DESCRIÇÃO DOS COMPONENTES	31
3.4.1. <i>Objetos MSS</i>	31
3.4.2. <i>Objetos de Configuração</i>	32
3.4.3. <i>Streams</i>	33
3.4.4. <i>Recursos Virtuais</i>	35
3.5. CENÁRIO TÍPICO DE USO DO MSS	38
4. A CONEXÃO VIRTUAL	41
4.1. INTRODUÇÃO	41
4.2. DESCRIÇÃO.....	42
4.2.1. <i>Objetos inclusos</i>	42
4.2.2. <i>O acordo de conexão</i>	42
4.3. INTERFACE	44
4.3.1. <i>Unicast</i>	44
4.3.2. <i>Multicast</i>	44
4.4. PROPOSTA DE IMPLEMENTAÇÃO.....	44
4.4.1. <i>Escopo</i>	44
4.4.2. <i>Metodologia de Desenvolvimento</i>	45
4.4.3. <i>A Hierarquia de Tipos da Conexão Virtual</i>	46
4.4.4. <i>Interação com outros objetos</i>	46
4.4.5. <i>Os Serviços da Conexão Virtual</i>	50
4.4.6. <i>O Modelo de Classes</i>	57
5. IMPLEMENTAÇÃO.....	60
5.1. INTRODUÇÃO	60
5.2. ESCOPO DO PROTÓTIPO	60
5.3. PROJETO E IMPLEMENTAÇÃO	61
5.3.1. <i>Tipos e Estruturas de Dados</i>	62
5.3.2. <i>Raiz da Hierarquia de Classes</i>	63
5.3.3. <i>Conexão Virtual</i>	64
5.3.4. <i>Dispositivos Virtuais</i>	66
5.3.5. <i>Adaptador de Conexão Virtual</i>	70
5.3.6. <i>Streams</i>	72
5.3.7. <i>Servidores</i>	73
5.4. EXEMPLO DE APLICAÇÃO	74
6. CONCLUSÃO.....	78
6.1. INTRODUÇÃO	78
6.2. AVALIAÇÃO.....	78
6.3. DIFICULDADES ENCONTRADAS	80
6.4. CONTRIBUIÇÕES.....	81
6.5. TRABALHOS FUTUROS	81
BIBLIOGRAFIA	82
ANEXO A : ESQUEMA DA VIRTUALCONNECTION.....	86

Lista de Figuras

FIGURA 1-1: PLATAFORMA MULTIWARE.	2
FIGURA 2-1: ESPECIALIZAÇÃO DE UM FRAMEWORK.....	13
FIGURA 2-2: INDEPENDÊNCIA DA PLATAFORMA.	14
FIGURA 2-3: SUBCONJUNTO DA HIERARQUIA DE CLASSES DE DISPOSITIVOS DO FRAMEWORKDAVE.	14
FIGURA 2-4: MODELO DE REFERÊNCIA OMA.	15
FIGURA 2-5: REQUISIÇÃO DE UM CLIENTE USANDO OORB.	16
FIGURA 2-6: COMPONENTES DE UM ORB-CORBA.....	16
FIGURA 2-7: INVOCÇÃO DE SERVIÇOS ATRAVÉS DOORB.....	18
FIGURA 2-8: SERVIÇOS DE UM ADAPTADOR DE OBJETO.	19
FIGURA 2-9: DESENVOLVIMENTO EM AMBIENTE CORBA..	22
FIGURA 3-1: INTERAÇÃO DO CLIENTE COM O MSS.....	27
FIGURA 3-2: INTERFACES INTERNAS DO MSS.	28
FIGURA 3-3: DIAGRAMA DE SUBTIPOS (HIERARQUIA DE INTERFACES)	29
FIGURA 3-4: CICLO DE VIDA DOS OBJETOS MSS.....	30
FIGURA 3-5: FAIXA DE VALORES PERTENCENTES A UMA CHAVE	31
FIGURA 3-6: HIERARQUIA DO TIPO STREAM.....	34
FIGURA 3-7: CONSTITUIÇÃO DO TIPO DISPOSITIVO VIRTUAL.....	36
FIGURA 4-1: EXTENSÃO PROPOSTA À HIERARQUIA DEVIRTUALCONNECTION.....	46
FIGURA 4-2: INTERFACE DA PORTA EM NOTÇÃO OMT.	48
FIGURA 4-3: A INTERFACE DO ADAPTADOR DE CONEXÃO VIRTUAL EM NOTÇÃOOMT.....	49
FIGURA 4-4: ESTABELECIMENTO DA CONEXÃO. DFD NÍVEL ZERO.	50
FIGURA 4-5: ESTABELECIMENTO DA CONEXÃO. DFD NÍVEL UM.	51
FIGURA 4-6: ESTABELECIMENTO DA CONEXÃO. DFD NÍVEL DOIS.....	52
FIGURA 4-7: INTERAÇÕES PARA O ESTABELECIMENTO DA CONEXÃO.	53
FIGURA 4-8: MECANISMO DE TROCA DE DADOS PARA CONEXÃO EM REDE.....	55
FIGURA 4-9: INTERFACE DO TIPO STREAM EM NOTÇÃO OMT.	56
FIGURA 4-10: DIAGRAMA DE TRANSIÇÃO DE ESTADOS PARA OSTREAM.....	56
FIGURA 4-11: DIAGRAMA DE INTERAÇÕES ENTRE OBJETOS DURANTE O FLUXO DE DADOS	57
FIGURA 4-12: MODELO DE CLASSES SEGUNDO A NOTÇÃOOMT.	58
FIGURA 4-13: MODELO DE CLASSES ESPECIALIZADO.....	59
FIGURA 5-1: FORMATO DE UMA STRING REFERÊNCIA DE OBJETO.....	63
FIGURA 5-2: ADAPTADOR REALIZANDO A TRANSFERÊNCIA DE DADOS.....	71
FIGURA 5-3: DISTRIBUIÇÃO DA APLICAÇÃO SOBRE OORB.....	74
FIGURA 5-4: UTILIZANDO O MSS NA LEITURA E EXIBIÇÃO DE ARQUIVO	75

Lista de Abreviações

API - Application Programming Interface
BOA - Basic Object Adapter
CASE - Computer Aided Software Engineering
CORBA - Common Object Request Broker
DFD - Diagrama de Fluxo de Dados
DMA - Direct Access Memory
IDL - Interface Definition Language
IEC - International Electrotechnical Commission
IMA - Interactive Multimedia Association
ISO - International Standard Organization
JDK - Java Development Kit
LAN - Local Area Network
MSP - Multimedia Stream Protocol
MSS - Multimedia System Services
ODP - Open Distributed Processing
OMA - Object Management Architecture
OMG - Object Management Group
OMT - Object Modeling Technique
ORB - Object Request Broker
PIO - Parallel Input Output
PREMO - Presentation Environment for Multimedia Objects
QoS - Quality of Service
WAN - Wide Area Network

Resumo

Desiderá, L., Arquitetura de Conexão de Dispositivos Multimídia em Ambientes Distribuídos. Campinas: DCA/FEEC/UNICAMP, 1997. (Dissertação de Mestrado)

O desenvolvimento de aplicações multimídia é uma atividade complexa devido à variedade, à sofisticação e às particularidades de tais aplicações, especialmente aquelas destinadas a operar em ambientes distribuídos. Neste contexto, foi proposta a arquitetura MSS (*Multimedia System Services*), com o objetivo de prover abstrações sobre características específicas de plataformas, de mídias e de sistemas de transporte, oferecendo suporte ao trabalho dos programadores de aplicações multimídia.

Este trabalho apresenta uma proposta de implementação para o serviço de conexão virtual definido na arquitetura MSS, empregando um ambiente baseado na plataforma CORBA (*Common Object Request Broker Architecture*) como infra-estrutura aos objetos distribuídos. Sua realização envolve o estudo e a interpretação da especificação do MSS, modelagem do sistema e implementação de um protótipo. Destaca-se a importância do emprego de uma arquitetura de serviços no suporte ao desenvolvimento de aplicações multimídia. As conclusões mostram que o MSS é completamente adequado para suporte às características deste tipo de aplicações.

Palavras-chave: Multimídia; Processamento eletrônico de dados - Processamento distribuído; Programação orientada a objetos (Computação); Redes de computação

Abstract

Desiderá, L., Multimedia Devices Connection Architecture in Distributed Environments. Campinas: DCA/FEEC/UNICAMP, 1997. (Masters)

Multimedia application development is a complex activity due to variety, sophistication and particularity of such applications, especially those intended for working on distributed environment. A framework called MSS (*Multimedia System Services*) has been proposed with the goal of providing abstractions with respect to specific features of platforms, media and transport systems to support the multimedia application developers' work.

This work presents an implementation proposal to the virtual connection service defined in MSS, using a CORBA (*Common Object Request Broker Architecture*) environment as infrastructure to distributed objects. It encompasses the MSS specification study and interpretation, system modeling and a prototype implementation. This work points out the importance of using a framework to support multimedia application development. Conclusions show that the MSS specification is completely adequated to support the features of this kind of applications.

Keywords: Multimedia; Distributed processing. Object-oriented programming (Computing); Computer Networks

1. Introdução

1.1. Contexto

A revolução tecnológica ocorrida durante este século provocou uma grande mudança na maneira como o homem se comunica. A convergência das telecomunicações com a computação, aliados à multimídia, transformou o computador em um veículo de comunicação, assim como a computação multimídia em uma das áreas de pesquisas mais intensas.

A mudança no papel desempenhado pelo computador foi fortemente impulsionada pela melhoria na performance da tecnologia digital, concomitantemente à redução de seu custo. À medida que o *hardware* é aprimorado, novos usos e formas de mídias tornam-se possíveis. Além disso, as redes digitais de serviços integrados estão influenciando fortemente no caráter distribuído das aplicações multimídia.

A pesquisa em computação multimídia distribuída envolve questões em diversas áreas do desenvolvimento científico e tecnológico. [Blair93] avalia pontos concernentes à tecnologia de comunicação e de sistemas distribuídos em aplicações multimídia, especificamente em redes multi-serviços, interfaces de rede, suporte de comunicação e de sistema operacional, plataformas de sistemas distribuídos e gerência de qualidade de serviço. [Gibbs94] destaca também algumas questões relativas a redes de comunicação, sistemas operacionais e sistemas de banco de dados.

Outro setor de grande atividade de pesquisa influenciado pela computação multimídia distribuída, é o de desenvolvimento de *software*. A complexidade e variedade das aplicações multimídia, aliadas à gama de conhecimentos envolvidos, torna o desenvolvimento de tais aplicações uma atividade árdua, especialmente quando se deseja generalidade. É neste sentido que [Gibbs94] ressalta a tecnologia de orientação a objetos, os ambientes multimídia e *frameworks*.

A combinação da computação distribuída com a orientação a objetos levou à definição de diversas plataformas para sistemas distribuídos, visando o provimento de serviços de gerência de objetos e de comunicação transparente entre eles. As mais conhecidas são CORBA (*Common Object Request Broker Architecture*) da OMG (*Object Management Group*) [CORBA93], [Orfali94], [Ben-Natan95] (também a

Este trabalho situa-se no contexto dos serviços de suporte ao desenvolvimento de aplicações multimídia distribuídas, os quais devem ser oferecidos pela camada *Middleware* da plataforma. Para isso devem ser utilizados o *framework* MSS como arquitetura de serviços e a plataforma CORBA como infra-estrutura de distribuição.

1.2. Motivação

As motivações para este trabalho encontram-se delineadas no contexto descrito acima. Entretanto, para maior clareza, elas são destacadas abaixo.

- o Projeto Temático, o qual compreende o desenvolvimento de uma plataforma distribuída com ênfase em multimídia (Plataforma *Multiware*) e possui um componente destinado ao oferecimento de serviços genéricos para programação de aplicações multimídia;
- a existência de um *framework* aberto como o MSS o qual serve de base para o desenvolvimento dos serviços a serem oferecidos pela plataforma *Multiware*;
- a disponibilidade de infra-estrutura de *hardware* e *software*, constituindo um ambiente distribuído com suporte ao desenvolvimento de aplicações distribuídas orientadas a objetos, segundo a especificação CORBA;

1.3. Objetivos

Devido à abrangência do contexto deste trabalho, tornou-se necessário delimitar seu escopo e estabelecer objetivos básicos para sua realização. Neste sentido, determinou-se como objetivos:

- estudar o *framework* MSS para compreender sua proposta, seus elementos componentes, as respectivas funcionalidades e o relacionamento entre eles;
- estabelecer uma arquitetura de implementação para o serviço de conexão virtual do *framework* MSS, baseando-se na plataforma CORBA como suporte à distribuição. Outros serviços, tanto do MSS (ex.: dispositivo virtual) como de outros componentes do PREMO (ex.: serviço de propriedades, ciclo de vida, etc.) serão implementados por outros participantes do projeto para posterior integração;
- realizar um protótipo que permita validar a arquitetura de implementação proposta no item anterior;
- avaliar a proposta de implementação com relação às arquiteturas MSS e CORBA.

mais difundida comercialmente) e ODP (*Open Distributed Processing*) da ISO [ODP95a], [ODP95b], [ODP95c] e [ODP95d].

Por outro lado, a conjunção da orientação a objetos com a multimídia promoveu a definição de diversos *frameworks* de objetos destinados à representação de características de sistemas multimídia. Dentre eles estão o DAVE (*Distributed Audio Video Environment*) [Friesen95], o *framework* de Simon Gibbs [Gibbs94], e o MSS (*Multimedia System Services*) da IMA (*Interactive Multimedia Association*) [IMA94], [PREMO96c].

No âmbito da UNICAMP, existe um Projeto Temático financiado pela FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) que visa a construção de uma plataforma distribuída com ênfase em aplicações multimídia. Esta plataforma é denominada *Multiware* [Madeira96] e constitui-se de três camadas (veja Figura 1-1):

- *Software/Hardware* - composta por sistemas operacionais, protocolos e *microkernels*. Não provê suporte à distribuição;
- *Middleware* - fornece suporte à distribuição e contém três partes:
 - ✓ subcamada ORB;
 - ✓ subcamada de Serviços de Objetos;
 - ✓ subcamada de Processamento Multimídia (em paralelo com as outras duas subcamadas);
- *Groupware* - camada de suporte a diferentes tipos de aplicações CSCW (*Computer Supported Cooperative Work*)

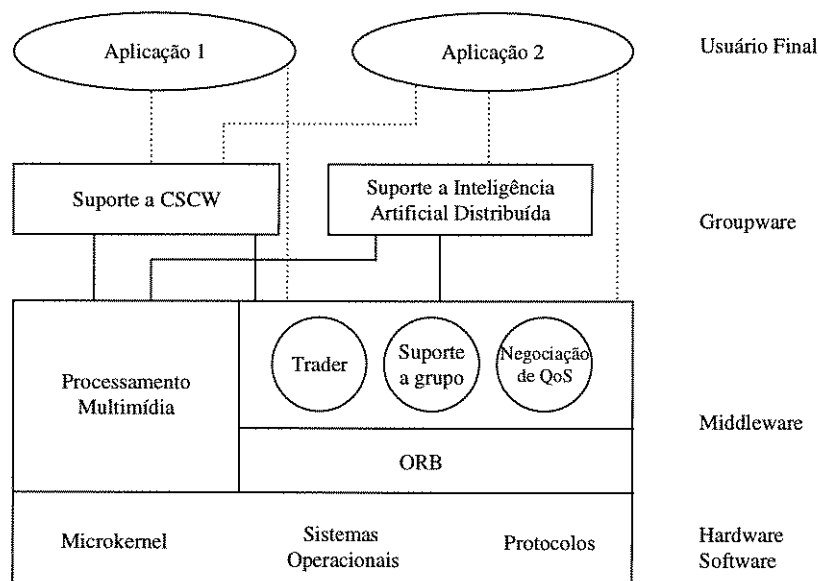


Figura 1-1: Plataforma Multiware.

1.4. Organização

Este trabalho está estruturado em 6 capítulos. Os capítulos 2 e 3 apresentam os conceitos teóricos que norteiam o desenvolvimento do trabalho. No capítulo 2 são destacadas as questões relativas à multimídia, aos sistemas distribuídos e ao desenvolvimento de *software* orientado a objetos, voltado às aplicações multimídia distribuídas. O capítulo 3 traz a descrição do *framework* Serviços de Sistema Multimídia (MSS) do padrão PREMO/ISO, sobre o qual se baseia este trabalho.

O capítulo 4 detalha um dos componentes do *framework* MSS, a conexão virtual, o qual constitui o cerne do trabalho de modelagem e implementação. Este capítulo apresenta também uma proposta de implementação para tal componente. O capítulo 5 trata de questões de implementação e descreve o protótipo realizado.

Finalizando, o capítulo 6 apresenta as considerações finais sobre o presente trabalho e sugestões para trabalhos futuros.

2. Computação Multimídia

2.1. *O Que é Multimídia*

Ao longo dos anos, tem-se observado uma contínua tendência para o desenvolvimento de formas cada vez mais ricas e variadas de mídia. Especialmente neste final de século, esta tendência tem-se acelerado e agora dispõe-se de uma enorme variedade de tipos de mídia para transmissão de informação [Williams91].

Do ponto de vista de sistema, a tecnologia de informação preocupa-se com o processamento e a comunicação da informação, combinando a tecnologia da computação com a infra-estrutura da comunicação, tendo por finalidade gerenciar informação para o usuário final.

Outra questão importante é a integração das várias formas de mídia, de onde pode-se dizer que **multimídia** é sinônimo de **variedade mais integração** [Williams91]. Um sistema de computação multimídia é aquele que permite ao usuário criar, armazenar e comunicar uma variedade de formas de informação de maneira integrada.

2.2. *Aplicações Multimídia*

A introdução da multimídia enriqueceu a utilidade de sistemas existentes em muitas áreas de aplicação e provocou o surgimento de novas áreas [Williams92]. Algumas delas são:

- **automação de escritório**, principalmente para produção/manipulação de documentos;
- **indústria de serviços**, em diversos setores como educação (enriquecendo e ampliando os métodos de ensino), finanças (a natureza do mercado financeiro requer rapidez para filtrar e tornar disponíveis informações sobre o mercado para tomada de decisões sobre investimentos) e de saúde (provendo infra-estrutura na construção de sistemas que suportem variadas informações, especialmente imagens, e permitam acesso rápido e fácil em qualquer localização);

- **aplicações domésticas**, como *home shopping*, *home banking*, vídeo sob demanda (usuários podem exibir em suas casas vídeos armazenados remotamente), manutenção e segurança;
- **cultura e lazer**, com sistemas que permitam aos artistas construir, armazenar e exibir suas criações, assim como às galerias e museus enriquecerem suas exposições com sistemas de informação multimídia.

[Gibbs94] identifica diversas categorias de aplicações multimídia, as quais são empregadas em inúmeras áreas:

- **aplicações videodisco interativas** - aparelhos de videodisco controlados por computador permitem acesso a quadros de vídeo específicos e execução de seqüências selecionadas em várias velocidades. Aplicações desenvolvidas utilizando-se videodisco (ex.: educacionais) são uma das primeiras aplicações multimídia interativas.
- **browsers hipermídia** - hipermídia é uma forma de organizar informações multimídia ligando os elementos de mídia. Os elementos ligados são chamados nós e o conjunto inteiro é uma *web* (teia) hipermídia. As ligações representam relacionamentos semânticos entre os elementos, por exemplo, uma imagem digital pode ser ligada a uma descrição textual que por sua vez é ligada ao áudio ou vídeo relacionado. Um *browser* hipermídia permite aos usuários verem os nós e cruzarem as ligações de um nó para outro.
- **sistemas de apresentação multimídia** - um sistema de apresentação multimídia é uma “máquina” que exibe, sincroniza, provê interação e, geralmente manipula material multimídia.
- **sistemas de autoria multimídia** - preparação de material multimídia, em particular quando usado para apresentação, treinamento ou entretenimento, é chamado autoria. Um sistema de autoria é uma coleção de ferramentas de *software* que auxiliam muitos passos da produção multimídia. Estes passos incluem captura, conversão de formato, edição, composição, interação e distribuição.
- **sistema de mail multimídia** - trata mensagens eletrônicas contendo áudio, gráficos, entre outras mídias e necessita de três componentes: um para criar a mensagem, um serviço de transferência e um componente para apresentação. Padrões são decisivos para que uma mensagem criada em um sistema possa ser exibida em outro.
- **sistemas de vídeo desktop** - sistemas de vídeo *desktop* usam estações de trabalho ou PCs para captura, edição e “pós-produção” (ex.: colocação de título e efeitos) de vídeo.

- **sistemas de conferência *desktop*** - um computador *desktop* equipado com microfone, alto-falantes, câmera de vídeo e ligado a uma rede multimídia pode ser usado para estabelecer conexões de áudio e vídeo entre outras máquinas equipadas similarmente. O poder de uma conferência *desktop* é aumentado pela adição de ferramentas multi-usuários, tais como editores de grupo, o que permite aos participantes compartilharem documentos (podem editar simultaneamente e observar as alterações dos outros) e outras formas de dados.

2.3. Tipos de Mídia

De maneira geral, pode-se dividir os tipos de mídia em dois grupos [Gibbs94]: temporal e não temporal. Sinônimos desta classificação são dinâmico/estático, baseado em tempo/não baseado em tempo e contínuo/discreto.

O termo estático é usado para referir-se à mídia que não possui dimensão temporal, como por exemplo texto, imagens e gráficos. Em contraste, mídias contínuas possuem uma dimensão temporal implícita, isto é, são apresentadas a uma taxa particular e, portanto, requerem um compromisso de continuidade dos serviços fundamentais do sistema. Nesta categoria são incluídos áudio, vídeo, música e animação [Williams91].

2.4. Sistemas Multimídia Distribuídos

Um ponto importante a ser destacado é a distinção entre sistemas multimídia que operam em uma única estação de trabalho e aqueles que podem estender-se através de um ambiente interligado por rede.

O termo **sistema multimídia distribuído** é introduzido para descrever o caso mais geral, de várias estações multimídia interligadas por uma ou mais redes de computadores. Além disto, uma **aplicação multimídia distribuída** designa uma aplicação que executa sobre um sistema multimídia distribuído [Williams91].

Entretanto, as aplicações multimídia distribuídas impõem diversos requisitos sobre a tecnologia de suporte, os quais serão discutidos nas próximas subseções.

2.4.1. Plataformas Para Sistemas Distribuídos

O papel de uma plataforma para sistemas distribuídos é de oferecer aos programadores um conjunto de abstrações que facilitem o desenvolvimento de aplicações distribuídas. Em geral, tais plataformas provêm um nível de transparência de distribuição, isto é, o programador não precisa estar ciente da natureza distribuída do sistema.

Há um consenso geral que as plataformas para sistemas distribuídos devem oferecer um ambiente orientado a objeto, onde as aplicações consistam de um

conjunto de objetos interagindo [Blair93]. A orientação a objetos provê a semântica mais lógica para formalizar e descrever sistemas distribuídos [Ben-Natan95].

Ambientes de objetos distribuídos são extensões à tecnologia de orientação a objetos. Estes ambientes coordenam a comunicação entre objetos locais e remotos de forma transparente à aplicação, e provêm facilidades para nomear e encontrar objetos. Outros serviços importantes incluem controle de autorização e ocultamento da heterogeneidade das máquinas [Gibbs94].

Neste contexto, plataformas distribuídas têm sido desenvolvidas para operar em ambiente heterogêneo, mascarando detalhes da arquitetura de *hardware*, infraestrutura de comunicação e implementações de sistemas. Exemplos de tais plataformas incluem CORBA (a ser detalhada na seção 2.6), e ODP [ODP95a], [ODP95b], [ODP95c] e [ODP95d].

2.4.2. Requisitos das Aplicações Multimídia Distribuídas

Os requisitos das aplicações sobre o sistema de suporte são considerados por [Blair93] sob os seguintes pontos:

a) Qualidade de Serviço

Mídias contínuas possuem uma dimensão temporal e devem ser exibidas em uma taxa particular. Além disto, utilizam recursos intensivamente e requerem um desempenho previsível do suporte para preservar suas características temporais.

É comum caracterizar as diversas necessidades das aplicações em termos de diferentes níveis de Qualidade de Serviço (*Quality of Service - QoS*). Em vista deste nível de diversidade, é exigido que o sistema suporte uma variedade configurável de níveis de QoS e que permita aos programadores de aplicações requisitarem o nível de QoS apropriado.

É importante notar também, que dentro de um sistema multimídia, níveis de QoS devem ser mantidos em termos fim-a-fim, ou seja, desde a fonte até o destino da informação. As aplicações devem se abstrair dos níveis de QoS suportados por componentes individuais do sistema.

b) Sincronização em tempo-real

Aplicações multimídia requerem mecanismo de sincronização em tempo-real para controlar a ordenação de eventos e precisão das interações na apresentação multimídia.

c) Interoperabilidade

Muitas classes de aplicações multimídia envolvem usuários remotos que provavelmente não utilizam uma infra-estrutura comum. Por conseguinte, é importante prover suporte para usuários trocarem informação através de diferentes tipos de redes, sistemas operacionais, estações de trabalho e formatos de mídia.

A tecnologia existente hoje não atende a tais requisitos. Pesquisas em várias áreas buscam adequar o sistema de suporte para atender os requisitos multimídia. Neste sentido, [Gibbs94] destaca:

a) Sistemas Operacionais

Pesquisas em sistemas operacionais multimídia estão em desenvolvimento, e incluem propostas a respeito de:

- i) Escalonamento em tempo-real - escalonadores em tempo-real permitem aos processos estabelecerem seus requisitos temporais; devem garantir a eles o acesso necessário à CPU e que não fiquem aguardando a execução de processos com prioridades menores.
- ii) Abstrações de mídia contínua - sistemas de arquivos tradicionais não foram projetados para acesso contínuo, como requerem as denominadas mídias contínuas; não conseguem prover dados (na gravação ou na recuperação) suficientemente veloz, o que leva a uma qualidade inaceitável. Sistemas operacionais multimídia devem oferecer serviços especiais que permitam a criação e o controle de *streams* para fluxos de dados contínuos.

b) Banco de Dados Multimídia

Sistemas de gerência de bancos de dados tradicionais tratam basicamente dados numéricos e caracteres, enquanto sistemas de banco de dados multimídia devem gerenciar dados de texto, imagem, gráfico, áudio e vídeo. Características e funcionalidades adicionais que provavelmente serão encontradas nos bancos de dados multimídia incluem:

- i) grandes volumes de dados - técnicas especiais de armazenamento são necessárias para otimizar o acesso a grandes volumes de dados, característicos em imagens e vídeo digitais;
- ii) tipo de atributos adicionais - além de armazenarem grandes volumes, bancos de dados multimídia precisam ter conhecimento a respeito dos tipos de dados sendo armazenados e precisam de tipos de atributos adicionais para valores de áudio, vídeo e outras formas de dados de mídia;
- iii) linguagens de consulta estendidas - as atuais linguagens de consulta e manipulação de banco de dados não são apropriadas ao acesso de dados multimídia. As linguagens de consulta para multimídia necessitam de operações que criem conexões entre o banco de dados e a aplicação permitindo intercâmbio de *streams* de dados de mídia.
- iv) operações longas em tempo-real - controle de concorrência em sistemas de banco de dados convencionais envolve o escalonamento de operações de leitura e escrita de curta duração. Elas oferecem ao gerente de concorrência flexibilidade no escalonamento, uma vez que

alteração na ordem de execução das operações provavelmente não atrasa seriamente a aplicação. Por isso, as operações são escalonadas de maneira a maximizar a performance do banco e manter a integridade dos dados. Entretanto, esta abordagem não é adequada para banco de dados multimídia, onde operações de gravação e recuperação/apresentação (*playback*) podem levar minutos ou horas e, além disso, são operações em tempo-real, que não podem ser arbitrariamente interrompidas por outras requisições. Portanto, bancos de dados multimídia devem suportar longas operações com restrições temporais e garantir que recursos suficientes serão alocados para a realização destas operações.

c) *Redes Multimídia*

As aplicações multimídia distribuídas requerem redes digitais para transferir dados de áudio e vídeo. Muitas das redes atuais não são apropriadas para estas aplicações pois os protocolos que as governam e limitação na largura de banda que elas oferecem são insatisfatórios para tráfego de áudio/vídeo digital. As redes multimídia diferem das redes convencionais pelas seguintes características:

- i) largura de banda - um fluxo de vídeo digital, mesmo comprimido, pode resultar em vários mega bits por segundo. O tráfego de poucos vídeos seria capaz de saturar redes como *Ethernet*. Redes multimídia devem oferecer larguras de banda da ordem de gigabits ou mais. Redes digitais de serviços integrados e ATM são exemplos de tecnologias relevantes;
- ii) *multicasting* - aplicações multimídia distribuídas frequentemente requerem transmissão de dados de uma fonte para vários destinos. Os protocolos para redes multimídia devem oferecer suporte a comunicação de grupo e permitir que o grupo se altere ao longo do tempo;
- iii) restrições de tempo-real - a transmissão de dados multimídia está sujeita a restrições temporais, como limites de atraso e de *jitter* (variação do atraso). Isto é essencial para aplicações que possuam fontes “ao vivo” ou que devem apresentar fluxos sincronizados;
- iv) confiabilidade - em redes multimídia, confiabilidade é uma questão de qualidade. Embora áudio e vídeo sejam menos sensíveis a erros, comparativamente a dados numéricos ou texto, não se pode ignorar totalmente a confiabilidade; a quantidade de erros introduzidos pode crescer demais e tornar a qualidade da apresentação inaceitável;
- v) qualidade de serviço (QoS) - aplicações possuem diferentes requisitos de comunicação. Uma aplicação de conferência, onde dados são apresentados uma vez e descartados, pode tolerar uma taxa de erro maior do que uma aplicação que armazena os dados para exibição

futura. Embora seja menos sensível a erros, a aplicação de conferência requer entrega rápida, enquanto que para a aplicação de armazenamento, atrasos não são relevantes. Uma forma das redes suportarem variações de requisitos é permitindo que cada aplicação especifique seus parâmetros de QoS.

2.5. Software Multimídia

A crescente variedade e sofisticação das aplicações multimídia, especialmente as distribuídas, evidencia a necessidade de suporte ao desenvolvimento destas aplicações, levando em consideração as características particulares da multimídia, que segundo [Mühlhäuser96] são: mídia (formatos, operações, tipo de dados), tempo, sincronização, qualidade de serviço (representação, mapeamento, negociação) *streams* e semânticas (como expressar significado, estrutura de relações e configuração de mídia, e aplicações).

[Gibbs94] destaca os objetos, ambientes e *frameworks* como tecnologias promissoras para desenvolvimento de software multimídia. Os objetos provêm blocos para construção das aplicações multimídia. Eles permitem encapsular a representação de várias mídias e especificar as operações disponíveis. Estes objetos podem ser combinados e coordenados usando serviços providos por ambientes multimídia (conjunto de serviços, abstrações, interfaces e bibliotecas destinadas à programação multimídia). Ambientes auxiliam na captura, no processamento e na apresentação de mídia, provendo interfaces de alto nível para tarefas como controle de dispositivos, *buffering*, escalonamento de processos, sincronização e conversão de dados. Eles podem prover ainda interfaces para banco de dados e redes multimídia, e suporte ao desenvolvimento de aplicações distribuídas. Os ambientes multimídia são frequentemente dependentes de *hardware* e de sistema operacional e devem, portanto, se adaptar a mudanças de plataformas. Surge então a necessidade dos *frameworks*, software genérico e interfaces genéricas que permitam novos dispositivos, novas mídias e novos métodos de comunicação serem suportados, com pouco impacto sobre as interfaces visíveis por programadores de aplicações.

Nas próximas subseções serão detalhadas as questões a respeito de objetos e *frameworks* para multimídia, devido à importância destes conceitos para todo o restante do trabalho.

2.5.1. Multimídia Orientada a Objetos

O paradigma de orientação a objetos tem-se mostrado adequado na produção de suporte a desenvolvimento das aplicações multimídia por diversas razões. [Gibbs94] destaca:

- **Encapsulamento** - um dos pontos fortes da orientação a objetos é a habilidade de encapsular informação, o que ajuda a proteger

programadores das particularidades de cada mídia e do *hardware* especializado.

- **Modularidade** - o desenvolvimento orientado a objetos é apropriado na captura das complexas interfaces dos serviços de processamento de mídia de forma modular, tornando-se de fácil uso por desenvolvedores.
- **Extensibilidade** - a programação orientada a objeto oferece mecanismos para estender código existente, permitindo acompanhar a evolução tecnológica da multimídia.
- **Portabilidade e desenvolvimento em plataforma cruzada** - idealmente, aplicações multimídia deveriam executar em diferentes plataformas e tolerar variações de *hardware* dentro delas. Interfaces orientadas a objeto podem fazer dependências de plataformas mais evidentes e com isso simplificar tanto o desenvolvimento em plataforma cruzada quanto desenvolvimento destinado a plataformas heterogêneas.
- **Software legacy** - ou necessidade de manter a compatibilidade com aplicações anteriores. Como o desenvolvimento de aplicações multimídia está menos restrito a *software* já existente, desenvolvedores são relativamente livres para tirar vantagem de novas técnicas como programação orientada a objetos.

2.5.2. Frameworks para Multimídia

Segundo [Gibbs94], uma forma compulsória de se aplicar as idéias de orientação a objeto na programação multimídia é construir um *framework* englobando os objetos e operações essenciais que aparecem em aplicações multimídia. Pode-se imaginar o *framework* como um conjunto de classes abstratas inter-relacionadas que são ajustadas e especializadas para diferentes plataformas multimídia. As aplicações que usam as classes abstratas adaptam-se às variações de desempenho e funcionalidade da plataforma e são, portanto, portáteis (importante objetivo uma vez que plataformas estão em contínua evolução).

Portanto, um *framework* de classes especifica interfaces, deixando em aberto suas implementações. Através de especialização, adicionam-se novas classes e as interfaces podem ser estendidas. Esta habilidade torna o *framework* apropriado para construção de ambientes de software aberto [Gibbs94]. Por exemplo, um *framework* para multimídia pode conter duas classes abstratas, *Vídeo* e *Componente_de_Processamento*, com o propósito de representar valores de vídeo e os componentes de *software* ou *hardware* que processam os valores de vídeo. As interfaces para estas classes podem ser especificadas sem levar em consideração os detalhes de uma representação de vídeo ou componentes de processamento particulares. Suporte às representações e componentes específicos é fornecido na implementação de subclasses concretas de *Vídeo* e *Componente_de_Processamento*.

A Figura 2-1 mostra uma visão esquemática e idealizada do processo de especialização de *framework*. As classes abstratas exibem uma interface de programação de aplicação (*Application Programming Interface* - *API*) visível em múltiplas plataformas. Cada plataforma (combinação de *hardware* e *software* de sistema operacional) oferece uma variedade de serviços de baixo nível (mecanismos de escalonamento e sincronização, armazenamento, protocolos de rede, etc.) através de uma interface de programação dependente. Quando o *framework* é especializado, classes concretas são introduzidas para implementar a API na plataforma em questão.

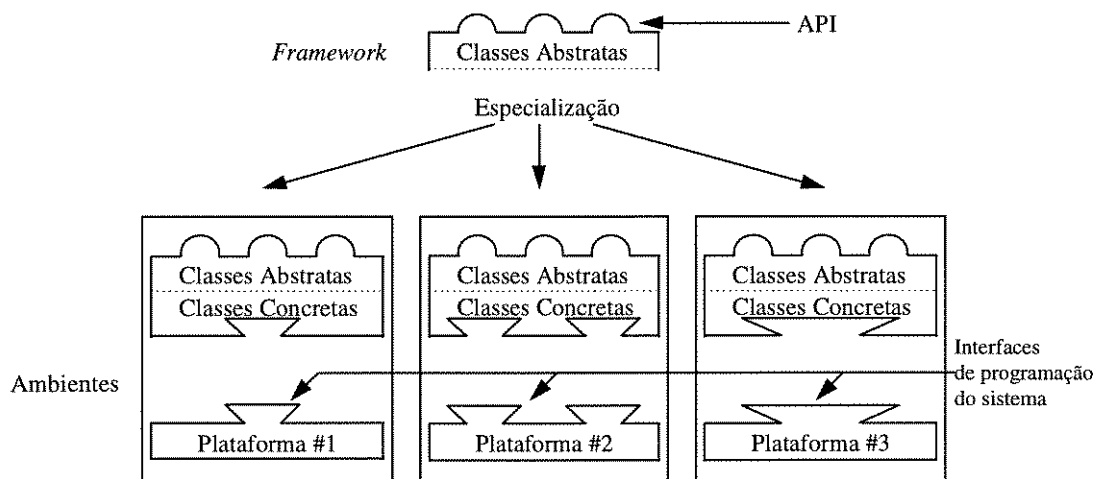


Figura 2-1: Especialização de um *framework*.

Juntas, a API, as classes (concretas e abstratas) e a plataforma formam um ambiente de programação multimídia específico. Aplicações usando a API podem ser portadas de um ambiente para outro (Figura 2-2) independentemente da plataforma. Entretanto, [Gibbs94] salienta que, na prática, a portabilidade é limitada por vários fatores:

- plataformas não oferecem as mesmas capacidades;
- diferenças de desempenho podem inviabilizar certas operações em algumas plataformas;
- aplicações podem desviar da API e acessar interfaces específicas da plataforma;
- não existe consenso sobre suporte, no nível de sistema, para programação multimídia; projeto de serviços de rede e de sistema operacional é uma área de pesquisa em desenvolvimento.

Hoje existem várias propostas de *frameworks*. Entre elas DAVE [Friesen95], o *framework* de Simon Gibbs [Gibbs94] e o *MSS* da IMA [IMA94].

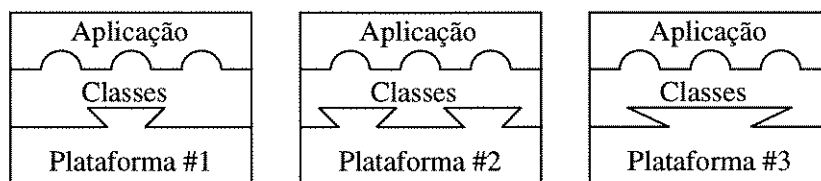


Figura 2-2: Independência da plataforma.

DAVE ou *Distributed Audio Video Environment* constitui-se de uma API, um gerente de conexão, um gerente de objeto e objetos dispositivos, que são organizados segundo uma hierarquia de classes. Um subconjunto desta hierarquia é mostrado na Figura 2-3. Usando a API os programadores podem criar objetos dispositivos distribuídos, que são abstrações de dispositivos multimídia reais. Através de herança e independência de dados, podem definir dispositivos e tipos de mídias adicionais, integrando tudo em um ambiente. Esta abordagem *plug-and-play* não apenas simplifica a programação como facilita o reuso de *software*.

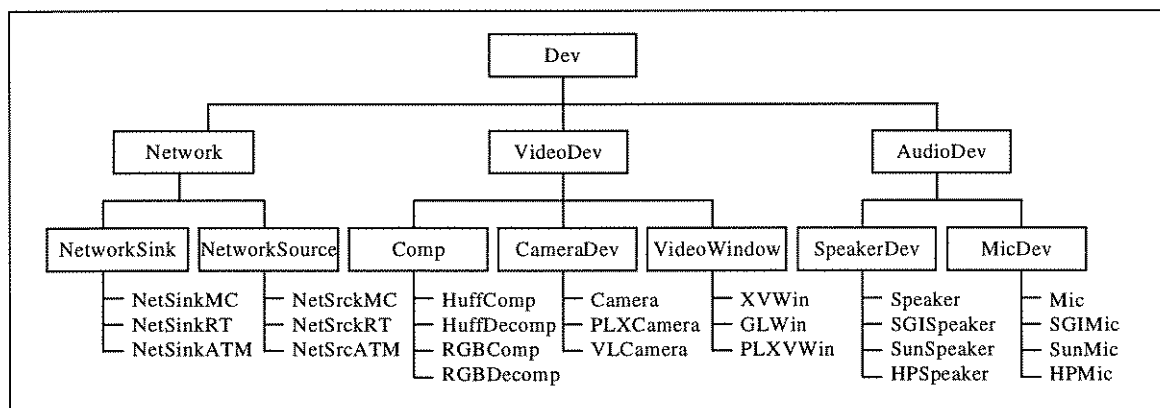


Figura 2-3: Subconjunto da hierarquia de classes de dispositivos do framework DAVE.

O *framework* de Simon Gibbs é composto por quatro grupos distintos de classes: classes de mídia, classes de transformação, classes de formato e classes componentes. De maneira geral, as classes de mídia correspondem a áudio, vídeo e outros tipos de mídia; classes de transformação representam operações de uma maneira flexível e extensível; classes de formato tratam da representação externa do dado de mídia e componentes representam os dispositivos e processos que modificam dados de mídia. Cada um destes grupos possui uma hierarquia de classes, cujos componentes são combinados na programação de uma aplicação. As hierarquias não serão tratadas aqui, mas os leitores interessados encontrarão maiores detalhes em [Gibbs94].

O leitor em encontrará [Blum96] uma comparação entre cinco plataformas para construção de aplicações multimídia distribuídas, algumas baseadas em *frameworks* de objetos, outras não, dentre elas o MSS, o qual será detalhado no Capítulo 4.

2.6. CORBA

2.6.1. Introdução

CORBA, ou *Common Object Request Broker Architecture* é um padrão da OMG (*Object Management Group*), uma organização internacional sem fins lucrativos, dedicada à construção de um *framework* para sistemas distribuídos orientados a objeto, o Modelo de Referência OMA (*Object Management Architecture*). As especificações dos componentes deste *framework* definem funcionalidade em termos de interfaces de objetos, sem estabelecer implementação, para possibilitar reusabilidade, portabilidade e interoperabilidade [Ben-Natan95]. Como mostra a Figura 2-4, o modelo compreende quatro categorias de componentes: *Object Request Broker* (ORB), Serviços de Objetos (*Object Services*), Facilidades Comuns (*Common Facilities*) e Objetos de Aplicação.

Os Objetos de Aplicação são objetos específicos de aplicações do usuário final e fazem uso das outras três categorias. Os Serviços de Objeto provêm serviços básicos necessários à criação de aplicações, tais como ciclo de vida, propriedades, persistência de objeto, segurança, gerência de transações, etc. As Facilidades Comuns provêm serviços de alto nível orientados ao usuário final, tais como correio eletrônico, impressão, etc. A divisão entre Facilidades Comuns e Serviços de Objetos é imprecisa, embora os Serviços de Objetos devam ser implementados em todo ambiente baseado em ORB enquanto as Facilidades Comuns são opcionais.

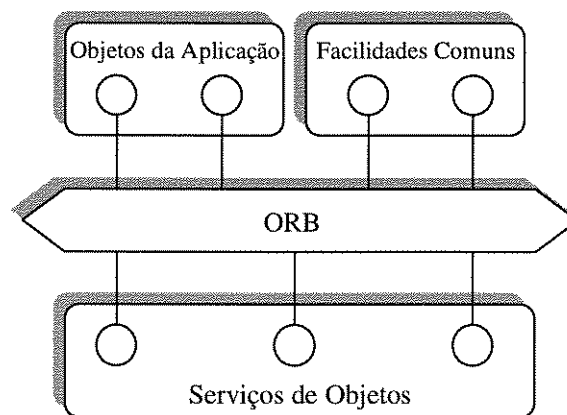


Figura 2-4: Modelo de Referência OMA.

O ORB (*Object Request Broker*) é, segundo [Ben-Natan95], o principal componente da arquitetura OMA. É ele quem envia e recebe mensagens entre diferentes objetos e componentes. Usando um ORB, um objeto cliente pode transparentemente invocar um método em um objeto servidor, o qual pode estar na mesma máquina ou na rede. O ORB intercepta a chamada, localiza o objeto que implementa o serviço, invoca o método e retorna o resultado, como ilustra a Figura 2-5.

A especificação CORBA [CORBA93] define a arquitetura dos ambientes baseados no ORB, sendo responsável por garantir portabilidade e interoperabilidade em um ambiente distribuído heterogêneo.

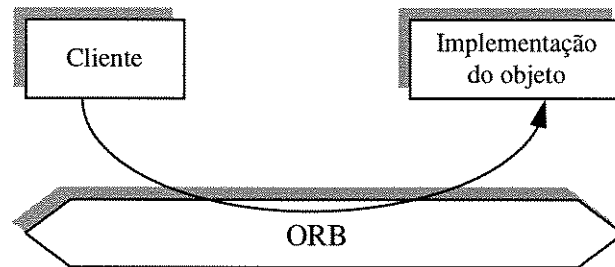


Figura 2-5: Requisição de um cliente usando o ORB.

2.6.2. Estrutura de um ORB na Arquitetura CORBA

A Figura 2-6 mostra os componentes de um ORB na arquitetura CORBA versão 1.2. As porções sombreadas indicam suas interfaces. Suas funcionalidades permitem agrupá-los em componentes do lado cliente e componentes da implementação (ou lado servidor). No lado do cliente, os componentes são:

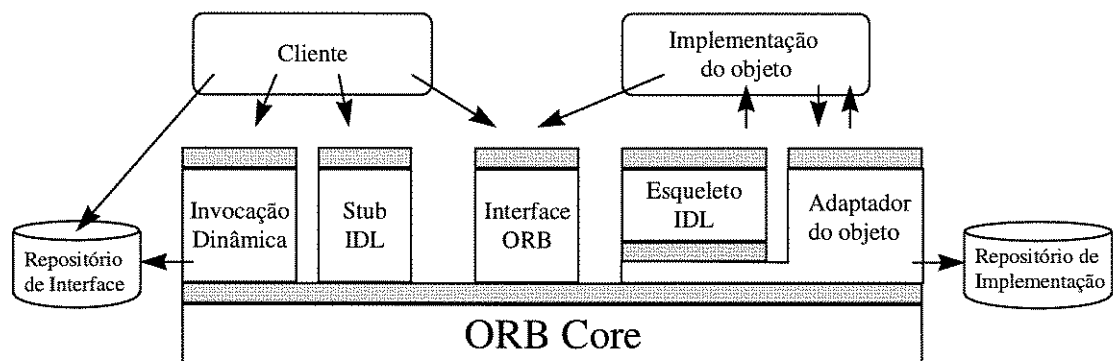


Figura 2-6: Componentes de um ORB-CORBA.

- **Stubs IDL** - provêm interfaces estáticas para os serviços dos objetos. Os *stubs* pré-compilados definem como os clientes invocam serviços nos servidores. Os serviços são definidos através da Linguagem de Definição de Interface (*Interface Defination Language - IDL*) e *stubs*, tanto do cliente como de servidores, são gerados pelo compilador IDL.
- **Interface de Invocação Dinâmica (*Dynamic Invocation Interface - DII*)** - permite que requisições sejam construídas e invocadas dinamicamente. O CORBA define APIs para obtenção de definições de serviços, geração de parâmetros, submissão da chamada e obtenção dos resultados. Não são usados os *stubs* na chamada através da DII.
- **Repositório de Interface** - permite que o cliente obtenha descrições de todas as classes registradas, os métodos que elas suportam e os

parâmetros que elas requerem. CORBA chama esta descrição de assinatura do método. O Repositório de Interface mantém representações das definições IDL na forma de objetos persistentes, disponíveis a tempo de execução.

- **Interface do ORB** - consiste de APIs para serviços locais que podem ser de interesse da aplicação e não fazem parte do mecanismo de invocação.

No lado do servidor, os componentes do ORB são:

- **Esqueletos IDL (*Skeletons*)** - ou *stubs* do servidor, provêm interfaces estáticas para cada serviço exportado pelo servidor. São criados através do compilador IDL e são chamados esqueletos pois não incluem nenhum detalhe de implementação. O programador deve preencher os esqueletos com o código real que será invocado. São os esqueletos que na realidade invocam os métodos que fazem parte da implementação do objeto.
- **Adaptador de Objeto** - fornece uma interface para os serviços do ORB que as implementações de objetos podem usar. Fornece também serviços necessários ao repasse das requisições que chegam do núcleo do ORB (*ORB Core*) e devem ser entregues a esqueletos para invocação de métodos. O adaptador de objeto é o responsável pela atribuição da identificação do objeto, ou seja, sua **referência**. Ele ainda registra as classes que suporta e suas instâncias (objetos) no repositório de implementação. CORBA especifica que todo ORB deve suportar um adaptador padrão denominado Adaptador de Objeto Básico (*Basic Object Adapter* - *BOA*). Outros adaptadores especializados para determinados tipos de serviços também podem ser definidos.
- **Repositório de Implementação** - mantém informação necessária ao ORB para localização e iniciação de implementações de objetos (classes que uma implementação de objeto suporta, os objetos que estão instanciados e suas identificações). É usado também para armazenar informações adicionais tais como segurança e depuração.
- **Interface do ORB** - consiste de APIs para serviços que podem ser de interesse das implementações de objetos e não fazem parte do mecanismo de invocação.

Componentes do cliente e da implementação comunicam-se através do **núcleo do ORB (*ORB core*)**. Ele provê a representação básica para os objetos e comunicação de requisições. Ele pode ser visto como mecanismo fundamental usado como camada de transporte. Sua implementação é específica e não definida pelo CORBA. A idéia é que o CORBA suporte múltiplos sistemas e por isso foi subdividido em componentes (interfaces) acima do núcleo de maneira a mascarar as diferenças entre diferentes implementações. Entretanto, como os fabricantes podem usar qualquer tipo de infra-estrutura para transporte do tráfico do ORB e para representação dos objetos em seus sistemas, o resultado é que os *ORBs* de diferentes

fabricantes podem não interoperar. Esta questão é solucionada na versão 2.0 do CORBA.

2.6.3. Requisições em CORBA

A Figura 2-7 mostra os dois tipos de invocações suportadas por um ORB CORBA: estática e dinâmica. Em geral, o cliente submete uma requisição usando uma **referência de objeto** (veja detalhes na seção 2.6.3.1), o nome de um método e um conjunto de parâmetros. As interfaces estática e dinâmica satisfazem a mesma semântica de requisição; o receptor de uma mensagem não reconhece como a requisição foi invocada. Em ambos os casos, o ORB localiza o adaptador de objeto, transmite os parâmetros e transfere o controle para a implementação do objeto através do esqueleto.

Os clientes vêem as interfaces dos objetos pela perspectiva de um mapeamento de linguagem (ou *binding*), que eleva o ORB ao nível do programador. Os programas clientes devem estar aptos a executar sem alterações de código em qualquer ORB que suporte o mapeamento de linguagem e com qualquer instância de objeto que implemente a interface. A implementação do objeto, seu adaptador e o ORB usado para acessá-lo são totalmente transparentes aos clientes, tanto em invocações estáticas quanto dinâmicas.

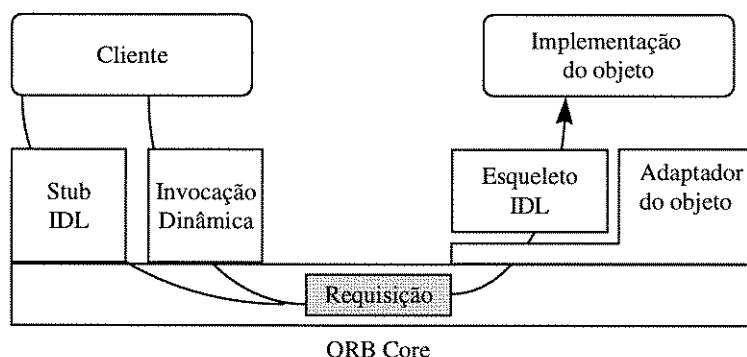


Figura 2-7: Invocação de serviços através do ORB.

2.6.3.1. Referências de Objetos

Uma referência de objeto é uma representação opaca que é usada para denotar um objeto em um ambiente baseado em ORB. Uma referência de objeto provê as informações necessárias para identificar um objeto e esta identificação é única. A especificação CORBA não define como implementar a referência de objeto, o que significa que diferentes implementações de ORBs podem prover diferentes representações. Um ORB provê um mapeamento das referências de objetos para a linguagem de programação usada, isolando os programas da representação real da referência no ORB. Todos os ORBs devem oferecer o mesmo mapeamento de referência de objeto para uma determinada linguagem de programação.

Referências de objetos servem tanto de “ponto de acesso” através do qual as operações são invocadas, como de valores de parâmetros. Além disso, referências podem ser convertidas para *strings* e então armazenadas (ex.: em disco). Posteriormente, estas *strings* podem ser retornadas para referências pelo mesmo ORB que fez a conversão inicial.

2.6.4. Adaptador de Objeto

O Adaptador de Objeto é o provedor de serviços primário do ORB para as implementações de objetos. Ele provê um ambiente completo para execução do servidor da aplicação. É ele quem define como um objeto é ativado. Isto pode ser feito através de um novo processo, através da criação de uma nova *thread* dentro de um processo existente, ou através do reuso de uma *thread* ou processo existente.

Abaixo seguem alguns dos serviços providos pelo Adaptador de Objetos. A Figura 2-8 ilustra melhor a estrutura e os serviços de um adaptador.

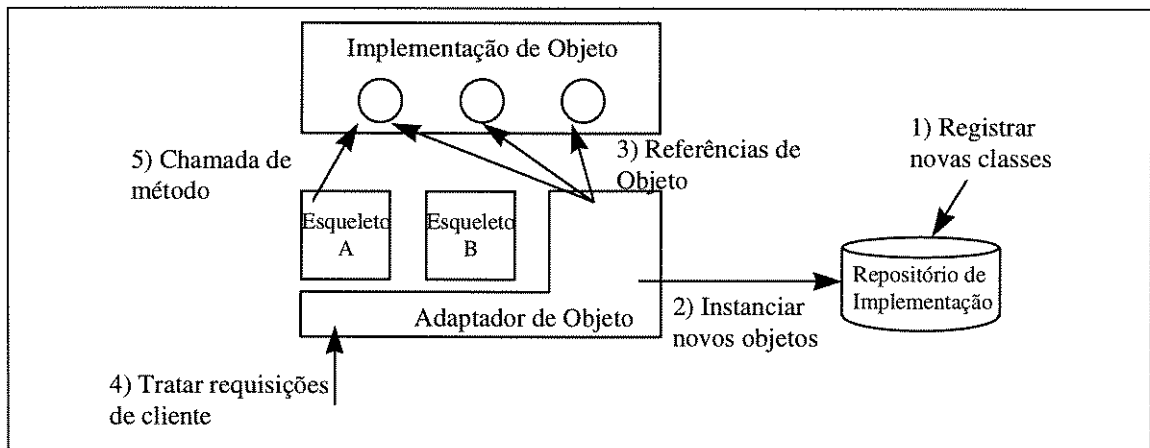


Figura 2-8: Serviços de um Adaptador de Objeto.

- 1) registra as classes implementação de objetos junto ao Repositório de Implementação;
- 2) instancia novos objetos em tempo de execução. O Adaptador de Objeto é o responsável por criar instâncias de objetos a partir das classes de implementação;
- 3) cria e gerencia referências de objetos. O Adaptador de Objeto atribui referências aos novos objetos que ele cria. É responsável pelo mapeamento das referências de objetos entre a representação específica do ORB e a representação específica da implementação;
- 4) trata requisições de clientes. O Adaptador de Objeto interage com a camada superior da pilha de comunicação do ORB *core*, extrai a requisição e entrega a mesma ao esqueleto. O esqueleto é o responsável por interpretar os parâmetros e apresentá-los na forma

adequada para a invocação do método no objeto (isto é, *unmarshaling* dos parâmetros)

- 5) direciona a chamada para o método apropriado. O Adaptador está implicitamente envolvido na invocação dos métodos descritos nos esqueletos. Por exemplo, o Adaptador pode estar envolvido na ativação da implementação ou autenticação da requisição.

2.6.4.1. Adaptador de Objeto Básico

A especificação CORBA requer que todo ORB tenha disponível um Adaptador de Objeto Básico (*Basic Object Adapter - BOA*). Qualquer implementação que faça uso do BOA deve estar apta a executar em qualquer ORB que suporte o mapeamento de linguagem adequado. Especifica ainda, que as seguintes funções sejam providas pelo BOA:

- instalação e registro de implementação junto ao Repositório de Implementação;
- criação e interpretação de referências de objetos (inclusive tradução de referências para *string* e vice-versa);
- ativação e desativação de implementação de objeto;
- ativação e desativação de outros objetos;
- invocação de métodos da implementação de objeto através do esqueleto IDL;
- autenticação do cliente da requisição.

CORBA define quatro políticas de ativação que devem ser suportadas por qualquer BOA. Estas políticas especificam as regras que uma determinada implementação segue para ativação de objetos. As quatro políticas são as seguintes:

- i) servidor compartilhado - múltiplos objetos ativos consistem em um programa servidor (processo). O servidor é ativado pelo BOA mediante a chegada da primeira requisição para algum dos objetos nele contidos. A implementação realiza seu processo de iniciação e notifica o BOA que está preparada para receber requisições (*impl_is_ready*). Todas as requisições subsequentes são então direcionadas para este processo servidor, que trata uma requisição por vez. O servidor deve notificar o BOA quando termina o processamento de uma requisição (*deactivate_obj*) e quando está pronto para ser desativado (*deactivate_impl*).
- ii) servidor não compartilhado - cada objeto consiste em um processo servidor diferente. Um servidor novo é ativado pela chegada da primeira requisição para o objeto. Quando o objeto termina sua iniciação, ele notifica o BOA que está pronto para tratar requisições

(*obj_is_ready*). Um novo processo é disparado toda vez que uma requisição é feita para um objeto que ainda não está ativo. O servidor notifica o BOA para desativação usando *deactivate_obj*.

- iii) servidor por método: um novo servidor é disparado toda vez que uma requisição é feita. O servidor executa somente durante a duração do método particular. A implementação não precisa fazer nenhuma notificação ao BOA.
- iv) servidor persistente: os servidores são disparados por elementos externos, não pelo BOA, mas se registram junto a ele (*impl_is_ready*) e recebem invocações. O BOA trata este servidor como um servidor compartilhado.

2.6.5. IDL: Interface Defination Language

IDL é a linguagem utilizada para especificar as interfaces que são usadas por clientes e que as implementações provêm. Ela é um subconjunto da linguagem ANSI-C++ com construções adicionais para suporte à distribuição.

IDL não é usada para escrever código, mas somente para especificação. Portanto, o cliente não será escrito em IDL; ele usa as especificações de interfaces, mas as chamadas são feitas na linguagem de programação nativa. Estas chamadas serão o resultado de um mapeamento das interfaces IDL para a linguagem de programação. As implementações de objetos provêm métodos que suportam os serviços definidos pelas interfaces e também são escritas em uma linguagem de programação (possivelmente diferente da usada no cliente).

As especificações em IDL incluem definições de módulos, constantes, tipos, interfaces ou exceções. Interfaces são compostas de um cabeçalho e um corpo. O cabeçalho identifica a interface (nome) e especifica herança (opcional). O corpo da interface pode conter declarações de constantes, tipos, exceções, atributos e operações. A interface é o principal componente das especificações IDL, por isso o nome Linguagem de Definição de Interface.

2.6.5.1. Mapeamento de Linguagens de Programação

CORBA foi projetado para ser um ambiente neutro com relação a linguagens e facilitar a interoperabilidade de clientes e implementações em diferentes plataformas, possivelmente implementadas em diferentes linguagens de programação. Mas IDL é apenas uma linguagem de especificação e para usar o ORB, é necessário saber como acessar suas funcionalidades a partir de uma linguagem de programação particular. Um mapeamento completo oferece ao programador acesso a todas as funcionalidades do ORB de maneira conveniente.

Para suportar a portabilidade, todas as implementações de ORB devem suportar o mesmo mapeamento para uma linguagem particular. Todos os

mapeamentos possuem aproximadamente a mesma estrutura. Eles devem definir meios de expressar na linguagem de programação:

- todos os tipos básicos IDL;
- todos os tipos construídos IDL;
- constantes definidas em IDL;
- referências de objeto;
- operações;
- atributos;
- exceções;
- interfaces definidas para o ORB e outros componentes tais como adaptadores de objeto, interfaces de invocação dinâmica e etc.

2.6.6. O Processo de Desenvolvimento em um Ambiente CORBA

A Figura 2-9 ilustra o processo de definição e a implementação de uma aplicação em um ambiente CORBA, cujos passos são descritos a seguir:

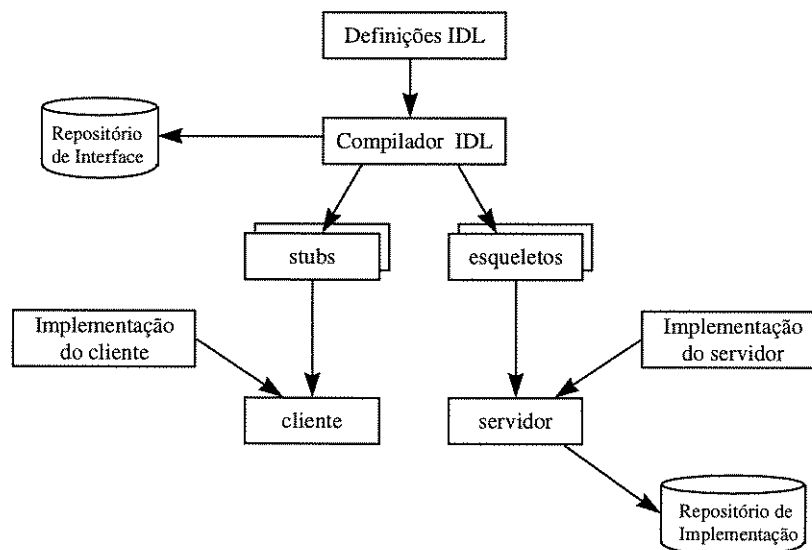


Figura 2-9: Desenvolvimento em ambiente CORBA..

- definir as interfaces dos objetos usando IDL - é através das definições IDL que os objetos declaram quais operações estão disponíveis e como devem ser invocadas.
- compilar as definições com um compilador IDL - através deste processo são automaticamente gerados os *stubs* do cliente e os

esqueletos do servidor. Além disso, as definições das interfaces são armazenadas no Repositório de Interface.

- iii) implementar os serviços e escrever o programa cliente - a implementação dos serviços e do cliente é feita utilizando uma linguagem de programação (ex.: C++). Os *stubs* devem ser ligados ao código do cliente e os esqueletos ao código dos respectivos servidores;
- iv) registrar os servidores no Repositório de Implementação.

Este processo de desenvolvimento é utilizado na criação dos componentes do protótipo descrito no Capítulo 5. Nele encontram-se descritos também os servidores e a aplicação cliente. As definições que servem de base para confecção das interfaces IDL dos serviços são apresentadas no modelo do Capítulo 4. Por outro lado, a aplicação cliente inspira-se no cenário da seção 3.5.

O próximo capítulo apresenta o *framework* *MSS*. Serão expostos seus componentes e suas funcionalidades, o que permitirá a compreensão do trabalho de modelagem e implementação descritos nos capítulos subsequentes.

3. Serviços de Sistema Multimídia

3.1. Introdução

O *framework* denominado Serviços de Sistema Multimídia (*Multimedia System Services - MSS*) foi desenvolvido pela *IMA (Interactive Multimedia Association)* na forma de uma Prática Recomendada (*Recommended Practice - RP*). Ele é baseado em um trabalho conjunto submetido pela *Hewlett-Packard (HP)*, *International Business Machines (IBM)* e *SunSoft*. Nesta proposta inicial, o *framework* é definido com base na arquitetura CORBA e declarado como um conjunto de interfaces IDL.

Posteriormente, o MSS foi incorporado pela proposta de padronização PREMO (*Presentation Environment for Multimedia Objects*) da ISO/IEC, o que elevou o nível de generalidade do mesmo e promoveu total desvinculação de implementação. A utilização do CORBA, ou qualquer outro ambiente de programação distribuído como suporte aos objetos, tornou-se uma opção de implementação. A representação das interfaces mudou para linguagem formal e algumas sofreram modificações. Foram introduzidos novos elementos à arquitetura (objetos de configuração) e a estrutura hierárquica das interfaces foi alterada para adequá-la aos outros componentes do PREMO [PREMO96a] e [PREMO96b]. Atualmente o PREMO constitui-se das quatro partes descritas abaixo [PREMO96a].

- **Parte 1: Fundamentos** - Contém a visão geral das motivações do PREMO, expondo seu escopo, justificativa e uma explicação sobre os principais conceitos; descreve a arquitetura global do PREMO e estabelece a semântica comum para especificar as características externamente visíveis dos objetos PREMO de uma maneira independente de implementação.
- **Parte 2: Componente Fundamental** - lista um conjunto inicial de tipos de objetos e de tipos não objetos, úteis para ao processamento de informação multimídia. Qualquer implementação em conformidade com o PREMO suportará estes tipos de objetos.

- **Parte 3: Componente Serviços de Sistema Multimídia** - descreve objetos que provêm infra-estrutura para construção de plataformas computacionais de suporte a aplicações multimídia interativas que lidam com mídias sincronizadas, temporais, em um ambiente distribuído heterogêneo.
- **Parte 4: Componente de Modelagem, Apresentação e Interação** - descreve objetos que são requeridos por sistemas computacionais avançados usando gráficos, vídeo, áudio ou outros tipos de mídia realçadas por aspectos temporais.

O PREMO pretende prover uma especificação formal de seu modelo de objetos e de alguns componentes [Herman96]. Entretanto, não será abolida a especificação em linguagem natural. O formalismo adotado pelo PREMO baseia-se nas linguagens *Z* e *Object-Z* [PREMO96a].

O componente MSS provê um conjunto padrão de serviços que pode ser utilizado por desenvolvedores de aplicações multimídia em uma variedade de ambientes computacionais. O MSS constitui um *framework* de *middleware* (camada de componentes de *software* entre o sistema operacional e as aplicações) e não uma plataforma, uma vez que não é direcionado à segurança, *toolkits*, *scripting*, interface com usuário e compartilhamento de aplicações [PREMO96c], [Blum96]. Como *middleware*, o MSS comanda os recursos de sistema de baixo nível na tarefa de suporte ao processamento multimídia.

O MSS do PREMO [PREMO96c] apoia-se no modelo de objetos definido na Parte 1 (Fundamentos do PREMO) [PREMO96a] e nos tipos de objetos e tipos não objetos definidos na Parte 2 (Componente Fundamental do PREMO) [PREMO96b].

3.2. Objetivos e Motivações do MSS

Em [PREMO96c] encontram-se os objetivos e as motivações na definição do MSS. O objetivo primário é prover infra-estrutura para construção de plataformas computacionais que suportem aplicações multimídia interativas, envolvendo mídias temporais, sincronizadas em um ambiente distribuído heterogêneo. Os principais motivos para definição do MSS são:

- prover abstrações e mecanismos que possibilitem as aplicações tratarem os problemas de computação multimídia distribuída;
- facilitar a implementação de aplicações complexas, como vídeo conferência;
- prover abstrações que possibilitem as aplicações lidarem com dispositivos de mídia sem considerar características específicas da plataforma, dos dispositivos conectados, ou de rede(s) interligando as plataformas e dispositivos;

- prover uma metodologia padrão;
- garantir aplicabilidade em larga escala;
- garantir performance adequada em condições adversas;
- facilitar compromisso com a qualidade de serviço;
- considerar a natureza temporal dos dados;

Na próxima seção é apresentada uma visão geral da arquitetura de serviços do MSS, assim como uma explicação genérica sobre as funcionalidades de seus componentes, como definido em [PREMO96c],

3.3. A Arquitetura MSS

Esta seção apresenta o MSS de três pontos de vista diferentes, que juntos representam um sumário da arquitetura. Estes pontos de vista incluem:

- visão do cliente;
- diagrama de subtipos, que descreve a hierarquia entre os objetos do MSS;
- breve descrição do ciclo de vida dos objetos do MSS.

Cada um destes tópicos é detalhado nas próximas subseções.

3.3.1. Visão do Cliente

A especificação [PREMO96c] preocupa-se com as interfaces visíveis pelo cliente. Implementações podem estender estas interfaces por razões de especificidade, para obedecer o comportamento dos objetos definidos na padronização.

A Figura 3-1 é um sumário das interações entre o cliente e os objetos do *framework* MSS. Ela é apenas ilustrativa, uma vez que são mostradas instâncias de classes abstratas ao invés de classes concretas. Nela não aparece também o processo de criação e destruição dos objetos. O cliente comunica-se com um pequeno grafo de fluxo de dados, composto por dois dispositivos virtuais e uma conexão virtual. Um objeto grupo que assiste ao cliente também é mostrado. Como se vê na Figura 3-1, o cliente interage com os objetos indicados por setas. Embora não esteja explícito na figura, o cliente pode interagir diretamente com os objetos *stream* e *formato*.

Cada dispositivo virtual é um nó do grafo. A natureza do processamento (captura, codificação, filtragem, etc.) varia de acordo com o objeto específico (e é implementado por subtipagem). Associado a ele, há um objeto *stream* e um ou mais objetos *formato*. Conexões virtuais e grupos também possuem um objeto *stream* associado. Estas associações são referidas como inclusão.

Um objeto *stream* oferece ao cliente uma interface para observar a posição da mídia. Alguns objetos *stream* oferecem operações para controlar o fluxo de mídia e alguns oferecem operações de sincronização. Além do *stream*, um dispositivo virtual contém uma ou mais portas, as quais descrevem um mecanismo de entrada e saída para o dispositivo. O objeto porta não tem interface visível ao cliente.

O objeto formato oferece uma abstração dos detalhes da formatação da mídia, separadamente do processamento e do controle de fluxo.

A conexão virtual provê operações para criar uma conexão entre uma porta de saída de um dispositivo virtual e uma porta de entrada de outro, encapsulando completamente questões de baixo nível a respeito do transporte. Conexões virtuais também oferecem suporte para conexão multiponto (*multicast*). Um objeto *stream* incluso oferece operações para controlar o fluxo de dados na conexão virtual.

O objeto grupo provê assistência ao cliente na gerência do grafo de fluxo de dados dos dois dispositivos virtuais e da conexão virtual. Ele oferece um mecanismo conveniente para alocação atômica de recursos e especificação de qualidade de serviço (*QoS*) para todo o grafo. O grupo oferece também acesso a um objeto *stream*, através do qual o cliente pode controlar o fluxo de dados do grafo.

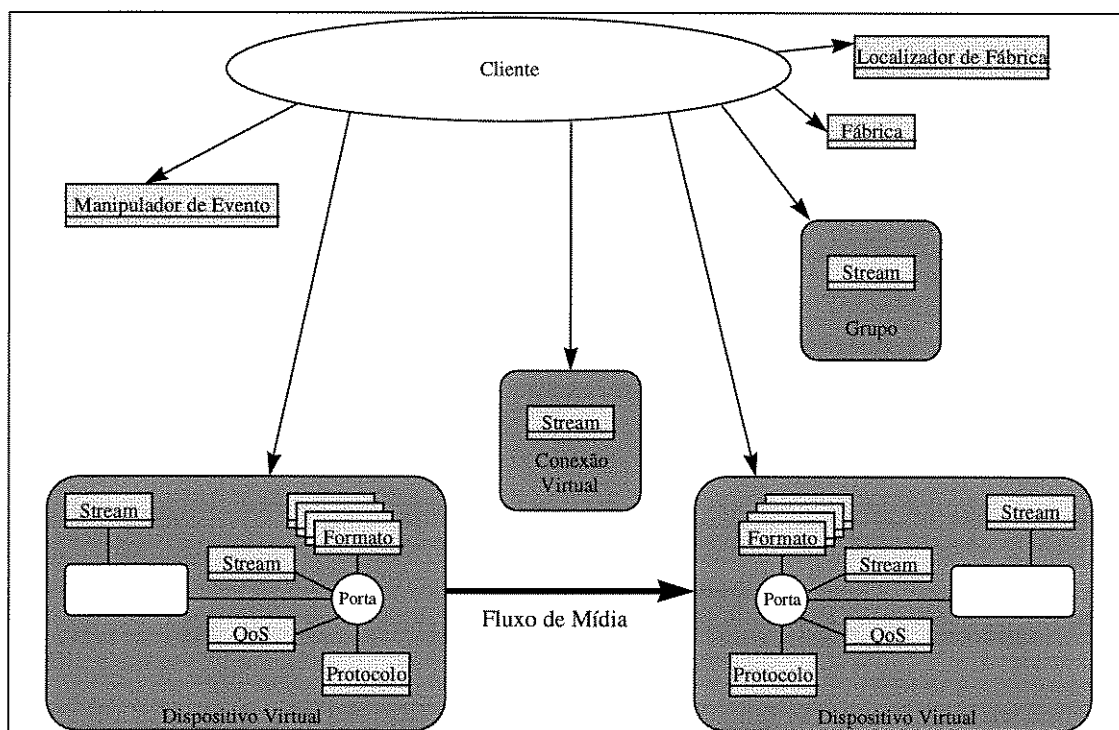


Figura 3-1: Interação do cliente com o MSS.

Os objetos do MSS são instanciados por fábricas (*Factory*). A fábrica oferece ao cliente facilidades para seleção entre os diversos objetos que ela é capaz de criar. Os clientes também podem utilizar o serviço de localizador de fábrica

(*FactoryFinder*), que permite encontrar a referência de uma fábrica capaz de instanciar um objeto cujas propriedades satisfazem uma lista de restrições.

O cliente pode registrar interesse em receber eventos específicos produzidos por vários objetos. Este mecanismo de evento é baseado no modelo descrito na Parte 2 [PREMO96b].

A Figura 3-2 mostra as conexões internas entre os objetos MSS. Na maior parte das vezes, o cliente desconhece que elas existem. [PREMO96c] não focaliza detalhes de baixo nível destas conexões, nem suas interfaces. O propósito da maioria delas é reduzir a carga de trabalho do cliente. Note, por exemplo, que a conexão virtual interage com os formatos de ambos os dispositivos virtuais, para conciliar os mesmos sem a intervenção do cliente. O grupo e seu *stream* associado também provêm assistência ao cliente, o grupo na alocação de recursos e o *stream* no controle do fluxo. As setas tracejadas mostram que os objetos enviam eventos ao cliente através do manipulador de eventos.

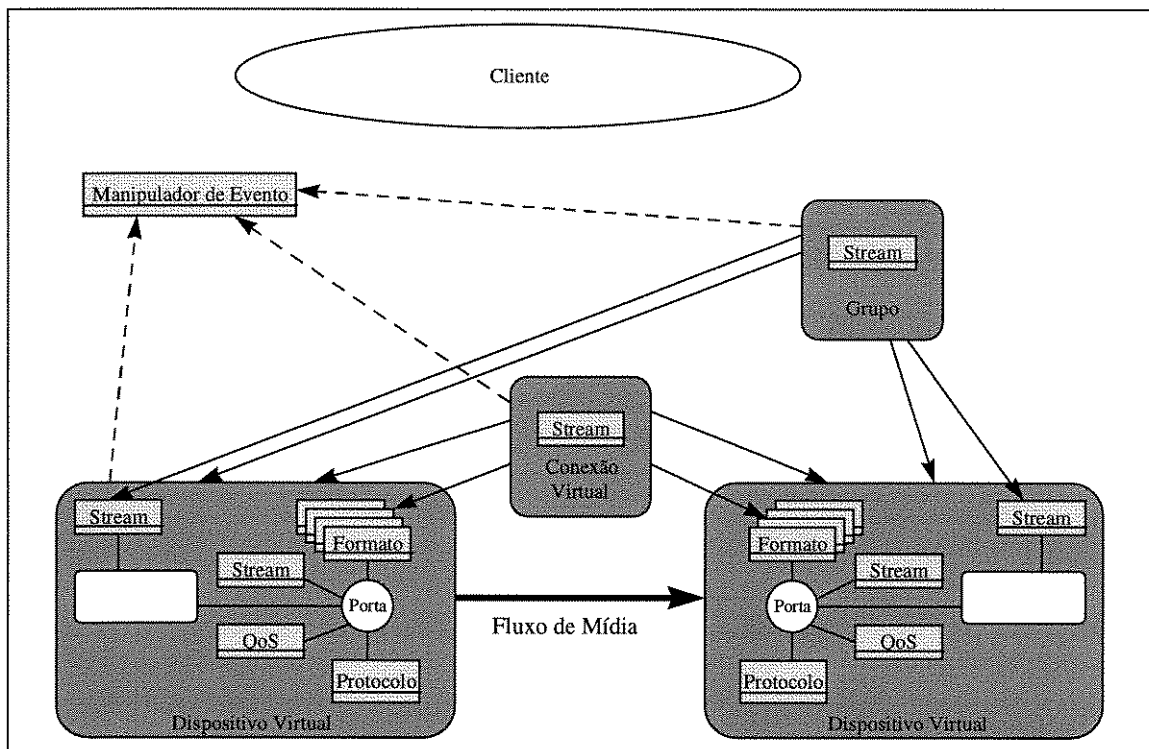


Figura 3-2: Interfaces internas do MSS.

3.3.2. Diagrama de subtipos

Um outra visão da arquitetura do MSS é dada pela Figura 3-3. Ela mostra um diagrama de subtipos simplificado, que não contém todos os tipos de objetos definidos no PREMO. Nela pode-se verificar as seguintes características:

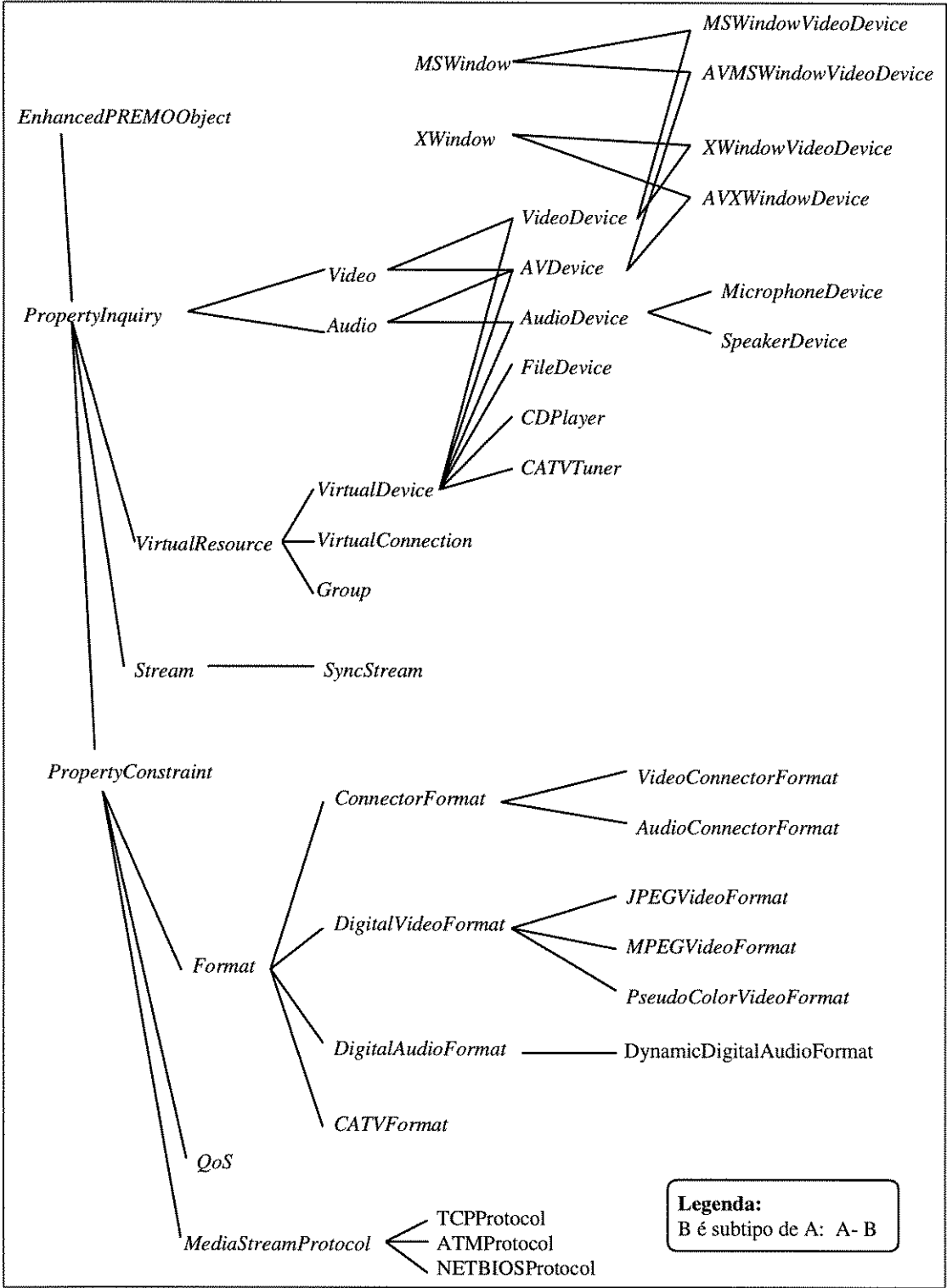


Figura 3-3: Diagrama de subtipos (Hierarquia de Interfaces)

- todos os tipos definidos no MSS são subtipos de *EnhancedPREMOObject* (veja [PREMO96b]). Portanto, herdam todas as características descritas no Componente Fundamental do PREMO;
- os descendentes de *VirtualDevice* são frequentemente subtipos múltiplos de outros tipos (*Audio*, *Video*, etc.) que especificam operações requeridas por dispositivos reais;
- a especificidade das mídias e dos tipos de dispositivos cresce da esquerda para direita. Estes tipos têm intenção de ilustrar tipos simples que podem ser suportados pelo MSS e não de ser o final da abstração das interfaces específicas de mídia;
- Alguns dos objetos no diagrama referem-se a objetos fora do escopo e não especificados no PREMO (*MSWindow*, *Xwindowm* etc.). Eles são apresentados no diagrama apenas por propósitos ilustrativos, para mostrar como os objetos MSS podem ser combinados com ambientes externos.

A especificação do MSS salienta que os vários formatos, protocolos, dispositivos virtuais, entre outros subtipos de objetos definidos por ela, não pretendem esgotar todas as possibilidades. Os tipos de objetos definidos em [PREMO96c] são utilizáveis, sem dúvida. Mas devem ser considerados como modelos gerais. Especificações mais especializadas podem ser definidas.

3.3.3. Ciclo de Vida dos Objetos

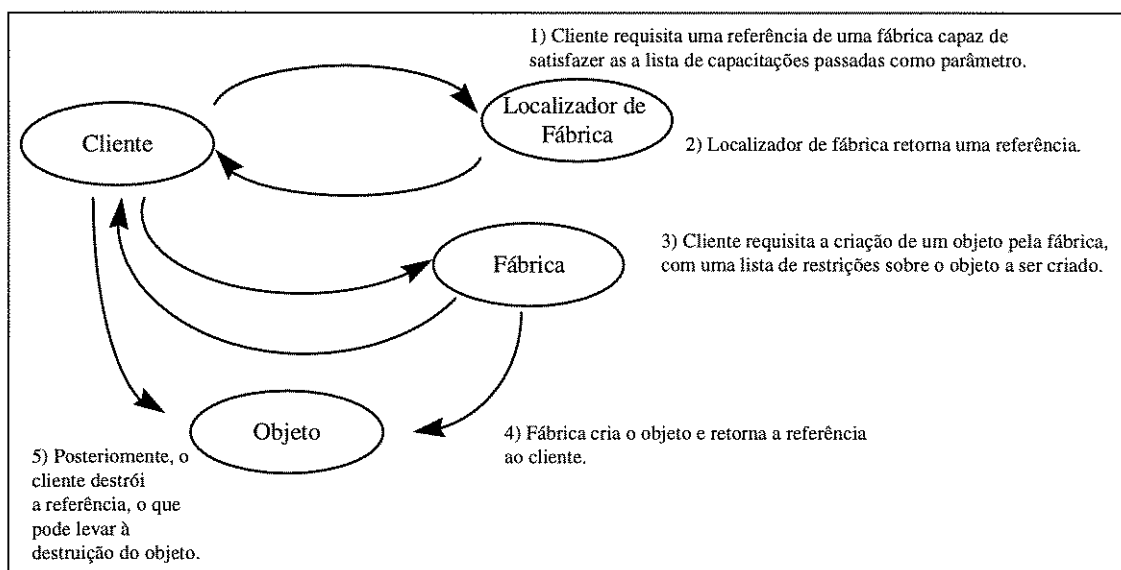


Figura 3-4: Ciclo de Vida dos objetos MSS.

A Figura 3-4 oferece uma visão esquemática do ciclo de vida de um objeto MSS e de sua referência. Além do ciclo de vida básico descrito em [PREMO96a], os

objetos apoiam-se nos conceitos de fábricas e localizadores de fábrica, os quais encapsulam a criação do objeto e da referência. O cliente tem a habilidade de criar objetos através de fábricas, especificando restrições sobre capacidades (*capabilities*) do objeto a ser criado (ex.: formatos de mídia que ele pode processar, qualidade de serviço que ele pode prover, etc.) O cliente recebe da fábrica uma referência para um objeto que obedece tais requisitos.

3.4. Descrição dos componentes

3.4.1. Objetos MSS

Todos objetos MSS são subtipos de *EnhancedPREMOObject*. Como tal, eles herdam as operações para gerência de propriedades definidas em [PREMO96b]. Propriedades são pares de chaves e seqüência de valores que estão conceitualmente armazenadas dentro de um objeto PREMO. As propriedades refinam a definição dos objetos e o comportamento deles, além daquele que é definido em seu tipo (isto é, as operações em sua interface). Algumas propriedades são comuns a vários objetos MSS, e outras são particulares do tipo ao qual o objeto pertence.

Uma capacitação (*capability*) é um tipo especial de propriedade (somente de leitura) que descreve os valores que outra propriedade pode assumir. As capacitações são definidas como parte da especificação do objeto, isto é, a informação que elas fornecem é a mesma para todas as instâncias daquele tipo. As propriedades de um objeto são identificadas por chaves (tipo não objeto, por exemplo, uma *string*). Na notação do PREMO, os nomes das propriedades utilizadas na especificação funcional dos objetos terminam com a letra “K” e as capacitações terminam por “CK”.

A Figura 3-5 fornece uma visão esquemática dos conceitos envolvidos na representação da faixa de valores pertencentes a uma propriedade. **Capacitação** (*capability*) descreve a faixa possível de valores pertencente a uma chave. É uma informação somente de leitura que pertence a um **tipo** específico. Uma instância deste tipo pode possuir **valores nativos** para esta chave, que descrevem a faixa de valores possíveis que esta **instância** pode associar a ela (subconjunto da capacitação).

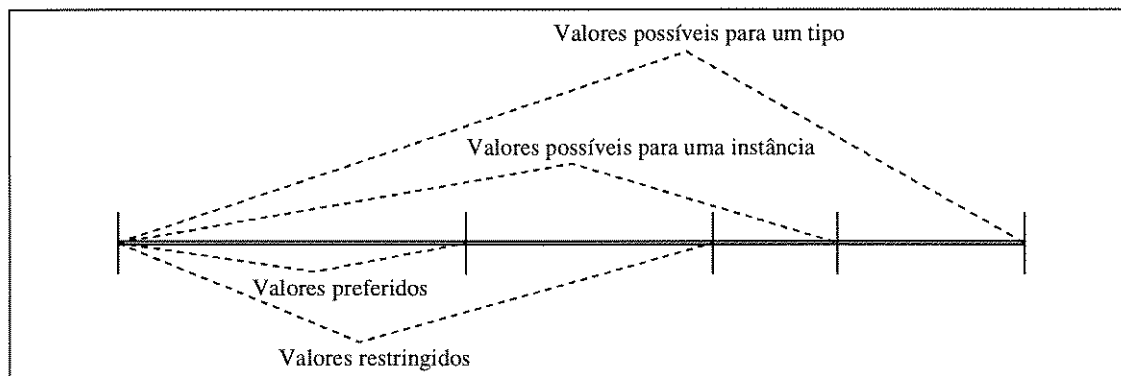


Figura 3-5: Faixa de valores pertencentes a uma chave.

O tipo *PropertyConstraint* oferece facilidades adicionais para restringir os valores associados a uma chave (dentro da faixa dos valores nativos dos objetos) e para selecionar os melhores valores para uma dada chave (dentro da faixa dos valores correntes).

Para elucidar estes conceitos e sua utilização dentro do MSS, é citado o seguinte exemplo: um tipo de objeto de áudio pode ser definido em termos da taxa de amostragem que ele pode processar. Os valores possíveis de taxa de amostragem são caracterizados por sua *capability*, ex.: <8KHz; 11,3KHz; 22,05KHz; 44,1KHz>. No processo de instanciação, o objeto pode observar que o *hardware* do ambiente não suporta taxas de 44,1KHz. Conseqüentemente, o valor correspondente da propriedade nativa para esta instância e esta chave será <8KHz; 11,3KHz; 22,05KHz>. Em um próximo passo, um cliente pode restringir uma faixa aceitável da taxa de amostragem para esta instância, requisitando que ela seja 11,3KHz ou 22,05KHz; isto significa que o valor corrente da propriedade para esta chave torna-se <11,3KHz; 22,05KHz>. Finalmente, como resultado de uma subsequente invocação da operação *select*, o objeto de áudio pode decidir pelo valor de 22,05KHz, para aumentar a qualidade da apresentação. O conjunto <8KHz; 11,3KHz; 22,05KHz> (isto é, o valor nativo da propriedade) pode sempre ser solicitado pelo cliente.

3.4.2. Objetos de Configuração

Os tipos de objetos MSS categorizados como **objetos de configuração**, são usados como depósitos de informação para outros objetos. Eles não provêm comportamento complicado (ou seja, não realizam operações complexas) e o papel deles é fornecer os parâmetros necessários ao funcionamento de outros objetos.

Todos os objetos de configuração são subtipos de *PropertyConstraint*. Os parâmetros que eles fornecem aos outros objetos são armazenados e manipulados através do mecanismo de restrição de propriedade, o qual caracteriza os objetos *PropertyConstraint*. Os objetos configuráveis (ex.: objetos recursos) contêm várias instâncias de objetos de configuração. O MSS tem três categorias de objetos de configuração:

a) Objeto Formato (*Format*)

Seu papel é representar os detalhes do formato de mídia (organização do *bitstream*) em uma porta particular de dispositivo. O formato permite ao cliente detalhar o quanto quiser sobre uma representação particular. Quando uma conexão é feita para uma porta, a conexão virtual interage com o objeto formato para negociar os detalhes do *bitstream* a ser passado entre as portas dos dispositivos virtuais.

Uma hierarquia de tipos é usada para descrever uma variedade de formatos de mídia. O objeto *Format* propriamente dito age como um supertipo comum para todos os objetos formato; ele define apenas duas propriedades gerais (nome do formato e ordem do *bitstream*). Todas as características específicas são deixadas para os vários subtipos.

b) *Transport and Multimedia Stream Protocol (MSP)*

O propósito dos objetos *Transport and Multimedia Stream Protocol* é prover informação sobre como enviar dados de mídia, juntamente com informação necessária para ajuste, sincronização e entrega dos dados em tempo-real. Para isto, o *MSP* define um pacote de mídia, que consiste de dados de mídia e um cabeçalho. Do ponto de vista dos objetos *MSP*, o dado de mídia é uma entidade opaca, cujo único atributo visível é o tamanho. O *MSP* pode transmitir, junto com o pacote de mídia, informações como estampa de tempo (*time stamp*), prioridade, número de seqüência, etc.

As capacitações (*capabilities*) de transporte variam de um tipo para outro. Entretanto, se um campo estiver presente em um dado transporte, este campo deve estar presente em todas as implementações daquele tipo. Para permitir futuras revisões, o *MSP* deve incluir o número de versão.

c) *Qualidade de Serviço (QoS)*

O cliente deve especificar a qualidade de serviço desejada quando requisita um recurso. Isto é feito estabelecendo-se as propriedades do objeto *QoS*. O *MSS* define um conjunto núcleo de propriedades que podem ser usadas pelo cliente para especificar a qualidade de serviço de interesse, o qual inclui:

- ✓ nível de garantia (garantido, melhor esforço, sem garantia);
- ✓ confiabilidade (entrega de dados confiável ou não);
- ✓ limites de atraso;
- ✓ limites de *jitter*;
- ✓ limites de largura de banda;

3.4.3. Streams

O tipo *Stream* e seus subtipos provêm um ponto singular para consultar e controlar o fluxo da mídia, independentemente de qual tipo ela seja. Os objetos *Stream* nunca são criados isoladamente; eles são objetos inclusos dos recursos virtuais, com o papel de monitorar e controlar o progresso do fluxo por todo recurso. Diversos objetos recursos virtuais necessitam de subtipos dos objetos *stream* definidos no *MSS* para fazer frente às várias características dependentes das mídias (ex.: detalhes da apresentação de mídia).

O *Stream* é declarado como um subtipo de *TimeSynchronizable*, que é definido em [PREMO96b]. O *MSS* fornece a especificação formal de somente dois tipos de objetos: *Stream* e *SyncStream*. A Figura 3-6 traz a hierarquia do tipo *Stream* que está parcialmente definida em [PREMO96a] e completada em [PREMO96b].

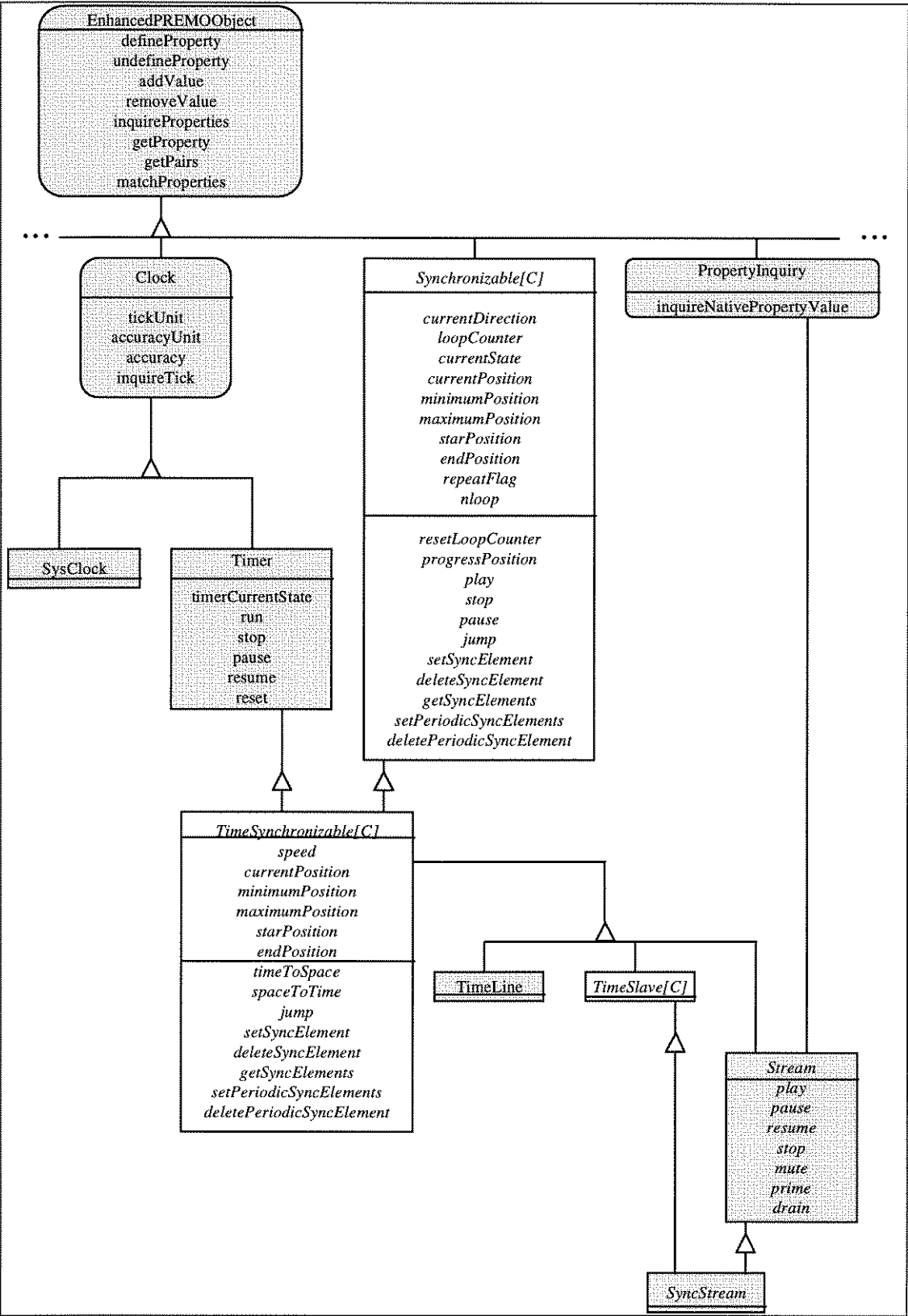


Figura 3-6: Hierarquia do tipo Stream.

O tipo *Stream* provê métodos para observar a posição da mídia, sendo que diferentes objetos abstraem diferentes conceitos de posição:

- *transport aware object*: entende transporte de pacotes;
- *stream aware object*: entende amostras de mídia;
- *time aware object*: entende como extrair tempo do fluxo de mídia;

Os objetos *Stream* oferecem controle de mídia e seu comportamento é governado por uma máquina de estados finita, que define os seguintes estados: STOOPE, PLAY, PAUSED, WAITING, MUTED, PRIMING e DRAINING.

Os objetos *SyncStream* são projetados para permitir sincronização de múltiplos *streams* de mídia, deixando que cliente especifique um segundo objeto *Stream* como mestre. Esta funcionalidade é suprida através de herança do comportamento dos objetos *TimeSlave*, também definidos em [PREMO96b].

O tipo *SyncStream* oferece ao cliente uma variedade de opções com respeito à sincronização. Por exemplo, um dispositivo de apresentação de vídeo pode ser sincronizado com um dispositivo de apresentação de áudio estabelecendo-se a relação apropriada entre seus *Streams* associados. Assumindo que o dispositivo de vídeo suporta um tipo *SyncStream*, o objeto *Stream* do dispositivo de áudio pode ser estabelecido como mestre do objeto *SyncStream* do dispositivo de áudio. Em outra situação, o cliente pode querer sincronizar dois dispositivos de apresentação a uma referência de tempo comum. Neste caso, ambos os dispositivos devem suportar o tipo *SyncStream*. O cliente faria do objeto *Stream* que detém a referência de tempo, o mestre dos objetos *SyncStream* associados aos dispositivos de apresentação.

3.4.4. Recursos Virtuais

Recurso virtual é uma abstração de um recurso físico que oferece ao desenvolvedor um modelo de programação consistente, independente de detalhes de implementações específicas. Isto torna as aplicações mais portáteis através de uma variedade de sistemas, ao mesmo tempo que permite o compartilhamento transparente de recursos físicos.

O MSS define um tipo *VirtualResource* que serve como um supertipo abstrato comum para as seguintes três classes básicas de recursos virtuais:

- i) dispositivo virtual - abstrai os processadores de mídia;
- ii) conexão virtual - abstrai conexões entre dispositivos virtuais;
- iii) grupo - oferece uma forma conveniente de interagir com uma coleção de dispositivos e conexões virtuais.

Os aspectos dos recursos virtuais, visíveis externamente, são descritos em termos de um conjunto de objetos de configuração. Tais objetos são criados e contidos por uma instância de *VirtualResource*.

- **elemento de processamento:** é a parte do dispositivo virtual que realiza as operações abstraídas por um tipo particular de *VirtualDevice*;
- **portas:** são os elementos de entrada e saída de dados para o elemento de processamento. Para o cliente, portas são designadas por um tipo de dado abstrato, não objeto; no caso do dispositivo virtual, alguns dos objetos de configuração estão agrupados nas portas, que contêm os seguintes objetos de configuração associados
 - ✓ **seqüência de formatos ordenados no tempo** - a informação de mídia que flui através da porta pode mudar o formato associado quando um determinado tempo for alcançado;
 - ✓ **objeto *Protocol*** - descreve os aspectos de comunicação através da porta;
 - ✓ **objeto *QoS*** - estabelece requisitos de qualidade de serviço na porta;
 - ✓ **um *EventHandler*** - age como manipulador de eventos específicos da porta
- ***Stream*:** o cliente direciona toda consulta e controle a respeito da posição do fluxo sobre a interface *Stream*;

3.4.4.2. Grupo

Freqüentemente é desejável manipular múltiplos recursos como um grupo, por exemplo, para exprimir requisitos de qualidade de serviço e para controlar o movimento de dados ao longo de um conjunto de dispositivos conectados.

O tipo *Group* permite o agrupamento de recursos virtuais: *VirtualDevice*, *VirtualConnection* e *Groups* (ou seja, hierarquia de grupos é permitida). As operações adicionais que ele define são para adicionar e remover objetos *VirtualResource* em um grupo. É possível também adicionar e remover um grafo inteiro de recursos.

O papel do tipo *Group* é realizar as seguintes operações:

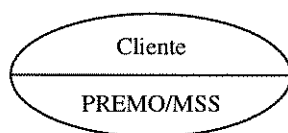
- **aquisição de recursos e qualidade de serviço fim-a-fim** - o cliente especifica a qualidade de serviço fim-a-fim e o grupo realiza o trabalho de atribuir *QoS* para cada objeto individual; além disso, chamando a operação *acquireResource* do grupo, ele promove a alocação atômica de recursos, evitando *deadlock* ou sub-utilização dos recursos;
- **controle de fluxo** - o grupo possui um *Stream* associado e os objetos *Stream* dos recursos constituintes podem ser entendidos como filhos do *Stream* do grupo. O *Stream* do grupo fornece ao cliente informação sobre o progresso do fluxo de dados dentro do grafo e permite que as operações invocadas sobre si sejam remetidas a seus filhos. O cliente

chama um único método e tem seu comando propagado para cada *stream* dos membros do grupo.

3.5. Cenário Típico de Uso do MSS

Nesta seção serão traçadas, de maneira simplificada, as ações de um cliente usando o MSS na realização de uma aplicação multimídia distribuída simples: captura de áudio com um microfone em um sistema e apresentação através de um alto-falante em outro sistema.

- a) Primeiro, o cliente declara e inicia o ambiente cliente dos Serviços de Sistema Multitimídia PREMO (PREMO/MSS).



- b) O cliente cria uma instância de *Microfone* no sistema correto, usando um localizador de fábrica e uma instância de fábrica. A localização do microfone é expressa em uma restrição passada à fábrica.



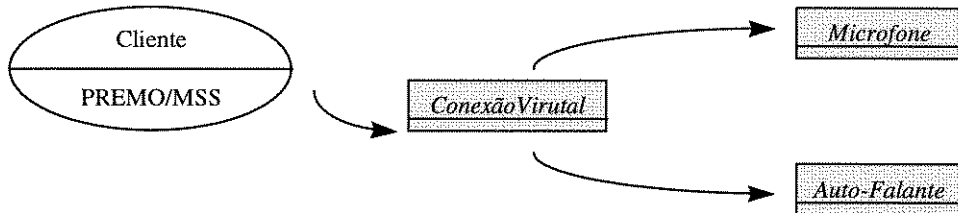
- c) O cliente segue um processo similar para criar o *Auto-Falante* em um sistema diferente.



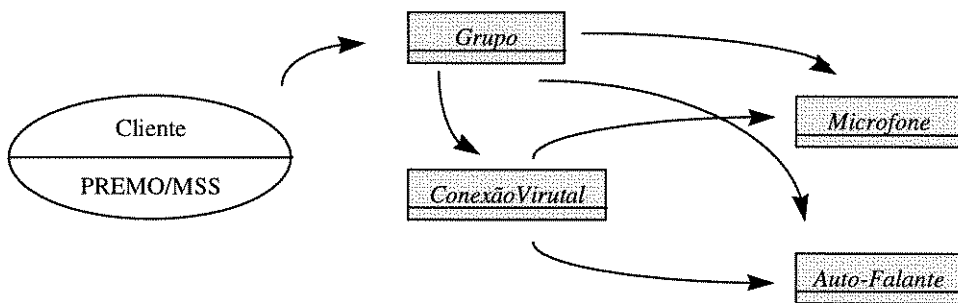
- d) O cliente cria uma *ConexãoVirtual* capaz de conectar os dois dispositivos virtuais em duas máquinas separadas.



- e) O cliente conecta os dois dispositivos enviando uma requisição *connect* à conexão virtual. A simplicidade desta conexão pode ser vista no código exemplo da seção 5.4.

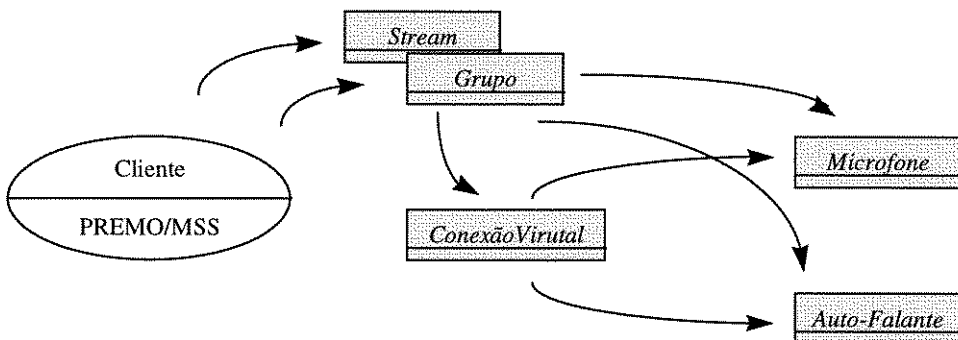


- f) O cliente cria o grupo e adiciona todos os recursos virtuais do grafo.



- g) O cliente pede ao grupo para adquirir recursos.

- h) O cliente obtém o objeto *Stream* do grupo e então dispara o fluxo.



Até este ponto foram apresentados conceitos e descrições de ambientes distribuídos, *frameworks* de objetos, suas características e a que se destinam. Embora o capítulo subsequente ainda apresente a descrição de um componente do MSS, ele já introduz uma proposta de concretização de tais conceitos, o que reflete o trabalho de interpretação realizado sobre eles.

No próximo capítulo será apresentada a conexão virtual, o componente do MSS responsável pelo estabelecimento da conexão entre dois dispositivos virtuais. Será apresentada também uma proposta de implementação para este componente, englobando tanto suas funcionalidades, como seus relacionamentos com outros elementos do *framework*.

Em seguida, o Capítulo 5 descreve a implementação de um protótipo e para fins de teste, utiliza um cenário com a mesma estrutura daquele ilustrado acima. O exemplo da Seção 5.4 ilustra de maneira bastante simplificada como um desenvolvedor de *software* multimídia deve programar suas aplicações se dispuser de uma implementação do *framework* MSS.

4. A Conexão Virtual

4.1. Introdução

A conexão virtual é um objeto muito importante dentro do *framework* MSS devido à abrangência das questões que ela aborda, como é destacado abaixo:

- **topologia** - a conexão virtual deve prover suporte para conexões um-para-um e um-para-vários;
- **modificação do stream** - na topologia um-para-vários a conexão (*multicast*) permite a reconfiguração da conexão para adicionar ou remover portas dinamicamente;
- **compatibilidade** - a conexão promove a verificação de compatibilidade entre os objetos de configuração (formato, protocolo e QoS) dos dispositivos virtuais a serem conectados;
- **múltiplos protocolos** - através dos diversos adaptadores de conexão que a conexão virtual é capaz de instanciar, ela permite o uso de variados protocolos de transporte;
- **QoS** - a conexão permite expressar requisitos de qualidade de serviço através de um objeto de configuração QoS;
- **estabelecimento e término da conexão** - sendo que o processo de estabelecimento envolve a determinação da qualidade de serviço ótima e negociação junto ao gerente de recursos;
- **controle e sincronização** - através do objeto *Stream* incluso, a conexão virtual permite a execução de comandos de controle e sincronização do fluxo de mídia, assim como determinação de informações sobre ele;
- **terminação do fluxo** - a conexão virtual permite os fluxos serem terminados em dispositivos virtuais tanto de *hardware* com de *software*.

Na próxima seção será apresentada uma descrição da conexão virtual, tal qual àquela oferecida pela especificação [PREMO96c]. Em seguida, será apresentada

uma proposta de implementação para este objeto, levando em consideração também os outros objetos do *framework* com os quais ele interage.

4.2. Descrição

A conexão virtual é o objeto que oferece à aplicação uma visão abstrata do fluxo de dados, encapsulando questões de fluxo de baixo nível.

As atribuições da conexão virtual consistem em negociar o acordo de conexão entre os dispositivos virtuais e prover um ponto focal para comando e obtenção de informações da conexão real. O transporte propriamente dito é realizado por um dispositivo virtual ou por um adaptador de conexão virtual.

4.2.1. Objetos inclusos

A conexão virtual inclui outros objetos do MSS, sendo eles: um objeto de configuração do tipo *QoS* (*quality of service*) e um objeto do tipo *Stream*. As propriedades do *QoS* devem refletir as expectativas da aplicação cliente. Através do objeto *stream* associado, o cliente pode obter informações e controlar a transmissão dos dados entre os dispositivos virtuais para fins de controle e sincronização.

4.2.2. O acordo de conexão

A conexão virtual constitui um acordo sobre os seguintes pontos.

a) Tipo de mídia a ser transportado entre as duas portas dos dispositivos virtuais

Cada porta de dispositivo virtual terá um objeto do tipo *Format* (formato) associado, que define quais os tipos de mídia que podem ser suportados. É possível para um objeto *Format* suportar múltiplos tipos de mídia (depende do valor de suas características). Um acordo sobre o tipo de mídia é alcançado quando o objeto *Format* do dispositivo fonte e o objeto *Format* do dispositivo destino tiverem valores de características compatíveis.

A atribuição dos valores para as características nos objetos *Format* pode ser realizada das seguintes maneiras:

- i) *Client media master*: o cliente atribui os valores das características para ambos os formatos. A conexão virtual é responsável apenas por garantir que os dois objetos são compatíveis.
- ii) *Device media master*: o cliente faz as atribuições para um dos formatos (mestre) e a conexão virtual irá negociar a atribuição do outro objeto.
- iii) *Connection media master*: o cliente não atribui nenhum dos dois e a conexão virtual irá negociar as atribuições de ambos os objetos formato, pesquisando primeiro as capacidades deles, e usando as

operações apropriadas sobre ambos objetos para encontrar uma combinação.

b) Tipo da conexão

As conexões podem ser dos seguintes tipos:

- i) *hardware*;
- ii) *direct*;
- iii) *local*; ou
- iv) *network*.

A conexão virtual irá determinar o tipo apropriado de conexão. Cada porta identificará suas características, algumas das quais podem ser obtidas através da estrutura de configuração da porta e algumas devem ser obtidas usando um acesso privado (ou seja, por meios não padronizados pelo PREMO). Estas características podem incluir identificações como *master/slave*, *PIO/DMA/shared memory/LAN/WAN* etc. A conexão virtual irá determinar também se os dispositivos virtuais estão no mesmo espaço de endereçamento, em espaços de endereçamento separados mas na mesma máquina ou em sistemas separados. A conexão virtual irá usar estas informações para determinar o tipo de conexão que pode ser feito.

c) Qualidade de serviço

Os parâmetros de qualidade de serviço (*quality of service - QoS*) são parte integrante da conexão virtual e refletem as expectativas da aplicação. Esta qualidade é parcialmente definida pelo objeto *QoS* associado à conexão virtual, e parcialmente pelos objetos *QoS* associados às portas dos dispositivos virtuais. Baseando-se nos diversos valores das propriedades de todos estes objetos de *QoS*, a conexão virtual pode estabelecer a qualidade de serviço que ela possa honrar.

d) Capacidades de sincronização e Stream

Os parâmetros nesta área do acordo de conexão descrevem:

- i) mecanismo de troca de dados;
- ii) tempo; e
- iii) políticas e mecanismos de sincronização

A conexão virtual irá determinar se os dispositivos virtuais concordam sobre um mecanismo de troca de dados comum. Irá determinar também o tipo dos objetos *Stream* associados a cada dispositivo virtual, assim como às portas. Usando esta informação, a conexão virtual irá, se necessário, instanciar o adaptador de conexão virtual apropriado.

4.3. Interface

O tipo *VirtualConnection* é derivado de uma hierarquia de tipos (Figura 3-3) e herda deles as propriedades, os atributos e as operações. O esquema em *Object-Z* [PREMO96a] que define o tipo *VirtualConnection* pode ser encontrado no Anexo A.

O MSS provê dois tipos de conexões virtuais: *unicast* e *multicast*. No entanto, especifica que ambos são abstratos, o que significa que nenhuma classe de algum destes tipos pode ter instâncias diretas. Isto implica que outros tipos ainda mais especializados devem ser definidos. Esta questão será tratada com mais detalhes na seção 4.4.3.

4.3.1. Unicast

VirtualConnection é o tipo base para conexões virtuais e provê suporte para conexões um-para-um. Sua interface oferece as operações básicas para criar e para destruir uma conexão entre dois dispositivos virtuais. Estas operações são chamadas *connect* e *disconnect* respectivamente.

Oferece também, a operação *getEndpointInfoList* para obter os elementos que a conexão interliga. Os métodos para adquirir e liberar recursos, *acquireResource* e *releaseResource*, respectivamente, são herdados de *VirtualResource*.

4.3.2. Multicast

O tipo *VirtualConnectionMulticast* provê suporte para conexões um-para-vários. Neste tipo de conexão, uma porta de saída fornece uma única instância dos dados para todas as conexões ou uma única porta de entrada pode receber dados de diversas conexões.

VirtualConnectionMulticast oferece operações adicionais para acrescentar e remover portas à conexão, chamadas *attach* e *dettach*, respectivamente.

4.4. Proposta de Implementação

A proposta deste trabalho consiste na implementação da interface da conexão virtual (*VirtualConnection*). Adicionalmente, serão propostas implementações para algumas outras interfaces, padronizadas ou não pela especificação, que são parte integrante do modelo e necessárias à execução dos serviços oferecidos através da interface da conexão virtual.

4.4.1. Escopo

Como mencionado na seção 4.1, a conexão virtual trata de várias questões em diversas áreas. Com o intuito de prover um protótipo, é necessário delimitar o escopo da implementação. Nesta proposta são consideradas as seguintes questões:

- **topologia** - a implementação provê suporte apenas à conexão um-para-um;
- **compatibilidade** - verifica-se a compatibilidade dos formatos das mídias através do objeto de configuração *Format*;
- **multi-protocolos** - a estratégia de verificação do adaptador adequado é implementada, embora não seja implementado um adaptador para um protocolo de transporte específico;
- **controle e sincronização** - são implementadas apenas algumas funções de controle daquelas definidas na interface do *Stream*;
- **terminação do fluxo** - esta proposta trata apenas dispositivos de *software*;

Não são tratadas questões relativas à:

- conexão multiponto;
- qualidade de serviço;
- sincronização;
- conexão terminada em dispositivos virtuais de *hardware*;

4.4.2. Metodologia de Desenvolvimento

A realização da proposta de implementação compreendeu duas fases: análise e projeto/implementação. Na fase de análise realiza-se:

- interpretação da especificação;
- levantamento das interações entre os elementos do MSS para realização dos serviços da *VirtualConnection*;
- estudo de operações ou atributos adicionais necessários ao cumprimento das atividades dos objetos e interações entre eles;
- modelagem estática e funcional pelo método OMT [Rumbaugh91], com mapeamento dos esquemas em *Object-Z* [PREMO96a] para classes.

A modelagem pelo método OMT [Rumbaugh91] foi escolhida devido à clareza com que este método emprega os conceitos de orientação a objetos e pela disponibilidade da ferramenta CASE LOV/OMT [LOV/OMT94a] e [LOV/OMT94b] como suporte.

No decorrer deste capítulo serão descritos os resultados obtidos na fase de análise. No Capítulo 5 serão detalhados o projeto e a implementação de um protótipo.

4.4.3. A Hierarquia de Tipos da Conexão Virtual

Os tipos *VirtualConnection* e *VirtualConnectionMulticast* são definidos na Especificação Funcional (cláusula 11.8.5) de [PREMO96c] como sendo tipos abstratos. Isto significa que estes tipos não podem ter instâncias diretas. Entretanto, em nenhuma outra parte da especificação é definida a hierarquia descendente. A Figura 4-1 mostra uma proposta para a extensão à hierarquia de tipos.

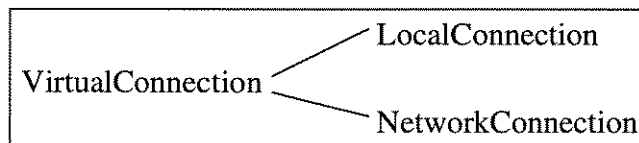


Figura 4-1: Extensão proposta à hierarquia de *VirtualConnection*

Esta opção foi escolhida levando-se em consideração os seguintes critérios:

- divisão de carga dos objetos conexão virtual;
- separação das questões de comunicação em rede e comunicação local.

Além disso, com base na versão anterior da especificação, [PREMO94], foram mantidas as propriedades *SourceInternetLocation* e *TargetInternetLocation*, as quais identificam as localizações dos dispositivos virtuais fonte e destino respectivamente. O valor destas propriedades deve ser fornecido pelo cliente na ocasião da criação da conexão virtual, permitindo à fábrica:

- criar o tipo adequado de conexão virtual (*Local* ou *Network*);
- identificar as capacitações de transporte que a instância da conexão virtual terá, ou seja, uma interseção entre os tipos de transporte suportados pelo sistema e os tipos de adaptadores que a conexão tem capacidade de instanciar.

Esta extensão produz um impacto direto sobre o item “tipo da conexão”, do acordo de conexão. Com a hierarquia, a instanciação de um tipo *NetworkConnection* implica em ter tipo de conexão em rede, enquanto que a instanciação de um tipo *LocalConnection*, deixa em aberto para a instância decidir se o tipo da conexão é *local*, *direct* ou *hardware*.

4.4.4. Interação com outros objetos

Para a realização de seus serviços, a conexão interage com outros objetos definidos no MSS, como os dispositivos virtuais, objetos de configuração, portas e adaptador. Entretanto, nem a porta nem o adaptador de conexão tiveram suas interfaces definidas na especificação PREMO [PREMO96c].

Especificamente para a porta, está definida uma estrutura de dados que permite armazenar e recuperar informações pertinentes à configuração da mesma. Todavia, estas informações não são suficientes para a conexão virtual determinar o

tipo da conexão entre as portas. A própria especificação atenta para este fato e menciona algumas outras características que devem ser levadas em consideração (seção 4.2.2). No caso do adaptador de conexão virtual, a especificação é ainda mais aberta e nada define.

Com o intuito de oferecer à conexão virtual a infra-estrutura mínima para a realização de suas tarefas, foram elaboradas interfaces para a porta e para o adaptador de conexão. Estas interfaces foram baseadas nos *templates* propostos em [PREMO96b] e serão descritas nas próximas subseções. Questões relativas a outros objetos do MSS, como os dispositivos virtuais (*VirtualDevice*), também serão tratadas.

4.4.4.1. Dispositivos Virtuais

Um ponto em que a especificação não é clara diz respeito à definição das responsabilidades sobre o objeto de configuração do tipo protocolo. A especificação estabelece que os dispositivos virtuais devem criar seus objetos de configuração, inclusive os da porta. Entretanto, não faz sentido que o dispositivo virtual decida sobre qual objeto protocolo usar na configuração da mesma. É de se esperar que isto seja decisão da conexão virtual (responsável pelas questões de transporte) ou do cliente (através de alguma restrição).

Para permitir que a conexão escolha o objeto protocolo para configurar a porta, é necessário que o dispositivo virtual crie as instâncias de protocolo e publique as informações necessárias à identificação dos mesmos. De posse destas informações, a conexão virtual poderá encontrar o tipo correto a ser utilizado, atribuir valores adequados às propriedades dos objetos protocolo, configurar as portas e instanciar o adaptador apropriado. Com a finalidade de suprir tais informações, foi acrescentado à interface *VirtualDevice* um atributo denominado *ConfigProtocols*.

Uma operação denominada *process* foi também acrescentada à interface *VirtualDevice* e representa o elemento de processamento, ou seja, a porção que realiza o processamento específico de cada dispositivo e cuja interface não é definida pela especificação. A operação *process* foi definida para permitir que o *stream* do dispositivo possa comandar o processamento interno do mesmo e cada subtipo de dispositivo virtual terá sua própria implementação para a esta operação.

Os *streams* dos dispositivos serão implementados da mesma forma que o *stream* da conexão virtual, como descrito na seção 4.4.5.2

4.4.4.2. A Porta

A estrutura *PortConfig*, definida pela especificação, guarda informações que permitem:

- identificar o tipo da porta (entrada/saída);
- obter as referências dos objetos manipulador de eventos e *stream* associados à porta;

- obter informações que possam ser convertidas em referência para os objetos de configuração *QoS*, *Protocol* e *Format*, associados à porta.

Além destas informações, a conexão virtual necessita também:

- identificar as formas de comunicação que a porta suporta. Para isto foi criado o atributo *connectionList*. Este atributo conterá informações como LAN, indicando comunicação em rede; LocalIPC, indicando comunicação entre processos local; SharedMemory, para memória compartilhada, etc;
- definir o modo de transferência de dados, isto é, o papel da porta na transferência dos dados. Para isto foi criado o atributo *IOMode*, que pode ser do tipo *DataDriven* ou *ControlDriven*. Detalhes sobre o mecanismo de troca de dados são fornecidos no item que trata sobre capacidades de sincronização e *Stream* na seção 4.4.5.1.

Três operações também são propostas:

- *getData*: através da qual obtém-se os dados da porta;
- *putData*: para colocar os dados na porta;
- *setTarget*: para indicar o destinatário e qual operação deve ser evocada, quando a porta estiver no modo *ControlDriven*.

A Figura 4-2 mostra a representação da porta em notação OMT [Rumbaugh91].

PortClass
config:PortConfig connectionList: SequenceString IOmode: TransferMode
setTarget(target:string, function:string) putData(data:StmBuffer) getData(data:StmBuffer)

Figura 4-2: Interface da porta em notação OMT.

4.4.4.3. O Adaptador de Conexão Virtual

O adaptador de conexão virtual, ou simplesmente adaptador, é o objeto instanciado pela conexão virtual para a realização do transporte de dados entre dois dispositivos quando esse transporte não puder ser realizado diretamente. Isto ocorre nos seguintes casos:

- quando os dispositivos virtuais estão em máquinas separadas;
- quando os dispositivos virtuais estão na mesma máquina mas não concordam sobre quem dirige o movimento de dados;

- quando os dispositivos virtuais estão na mesma máquina mas em processos diferentes com gerentes de *buffer* incompatíveis.

No caso em que os dispositivos virtuais estão em máquinas separadas, o adaptador será a interface com o mecanismo de transporte em rede, atuando como filtro para as particularidades de cada tipo de transporte. Desta forma, uma implementação do MSS deverá oferecer várias classes de adaptadores para suportar uma variedade de protocolos de rede, como por exemplo TCP(UDP)/IP, ATM, etc. O adaptador (de conexão em rede) deve consistir de duas entidades separadas, que se comunicam através da rede para transferência de dados de mídia e troca de informações de controle. Trabalhos que enfatizam questões de comunicação, como [Araujo96], [Schmidt95], [Pyarali96] servem de base para o estabelecimento de um modelo de implementação para o adaptador. Todavia, neste trabalho, foi utilizado um modelo simplificado composto por uma única entidade, um servidor. Detalhes desta implementação são encontrados na seção 5.3.5.

O adaptador de conexão é uma construção privada da conexão virtual, não visível ao cliente e não padronizada pelo PREMO. A interface concebida para ele (Figura 4-3) permite que o adaptador seja configurado para atuar em diferentes modos de transferência de dados, de acordo com os valores estabelecidos para os atributos *InputMode* e *OutputMode*.

ConnectionAdapter
InputMode: TransferMode OutputMode: TransferMode ByteCounter: real
send(buf_in: StmBuffer) recv(buf_out: StmBuffer) setSource(target:string, function:string) setTarget(target:string, function:string) transfer()

Figura 4-3: A interface do adaptador de conexão virtual em notação OMT.

As operações *setSource* e *setTarget* permitem indicar os objetos fonte e destino dos dados, respectivamente, assim como a operação que deve ser evocada sobre eles para obter/entregar os dados.

As operações *send* e *recv* permitem que as portas dos dispositivos comandem respectivamente o envio e a recepção dos dados, diretamente com o adaptador. A operação *transfer* é a operação que executa a transferência de dados. Para isso, algum dos atributos que indicam o modo de transferência deve ter valor *ControlDriven*. Neste caso, a cadência da transferência é determinada pelo *Stream* da conexão virtual, ou seja, pela frequência com que ele evocada a função *transfer*.

O atributo *ByteCounter* armazena a posição do fluxo. Quando um cliente solicita a posição do fluxo para o *stream* da conexão, ele se reporta a este atributo do adaptador.

4.4.5. Os Serviços da Conexão Virtual

Conforme descrito anteriormente, a conexão virtual é responsável por estabelecer um acordo de conexão com respeito a tipo de mídia, tipo de conexão, qualidade de serviço e capacidades de sincronização e *Stream*. Deve também, oferecer um ponto para informação e comando da conexão real. Cada um destes tópicos é abordado a seguir, visando apresentar a interpretação e a proposta de implementação dos mesmos.

4.4.5.1. Estabelecimento do Acordo de Conexão

O acordo de conexão é constituído em duas fases: a primeira no momento do estabelecimento da conexão (operação *connect*); a segunda no momento da aquisição de recursos. Na primeira fase são estabelecidos o tipo de mídia, o tipo da conexão e o mecanismo de troca de dados. Na segunda fase é estabelecida a qualidade de serviço.

Os processos envolvidos na primeira fase do estabelecimento do acordo são descritos a seguir e inicia-se com chamada da operação *connect* pelo cliente. Como ilustra a Figura 4-4, na forma de diagramas de fluxo de dados (DFD) [Rumbaugh91],

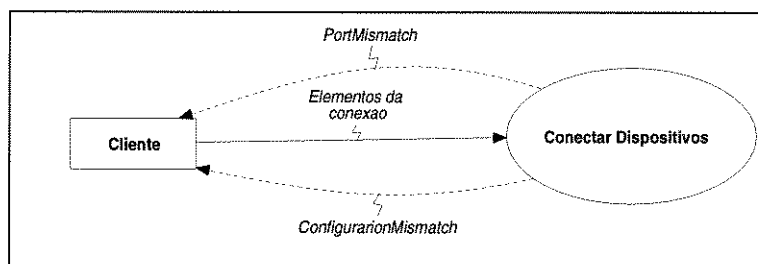


Figura 4-4: Estabelecimento da conexão. DFD nível zero.

a) Tipo de mídia

Para promover este item do acordo, deve ser executada a comparação dos objetos de configuração *Format* associados às portas dos dispositivos virtuais a serem interconectados. O primeiro nível de avaliação considera os tipos dos objetos *Format*: o formato designado como mestre (aquele que está associado à primeira porta passada como parâmetro em *connect*) deve ser do mesmo tipo ou supertipo do formato designado como escravo (aquele que está associado à segunda porta passada como parâmetro para *connect*).

O segundo nível de avaliação envolve as propriedades de cada objeto formato. A comparação será realizada seguindo a política *connection media master* (seção 4.2.2), ou seja, deixando que a conexão virtual estabeleça os valores correntes para as propriedades, e retirando do cliente todo encargo da configuração. Dentro do processo “Comparar Formatos” (Figura 4-5), a conexão virtual realiza diversas atividades, cuja seqüência é ilustrada na Figura 4-7 e descrita abaixo. Para estabelecer a configuração das propriedades dos formatos a conexão virtual:

- inquire o formato mestre sobre as propriedades que devem ser comparadas (operação *getProperty* para chave *MatchPropertyListK*);
- cria a lista de propriedades e restrições para o casamento dos formatos (os valores de uma propriedade no formato escravo deve incluir pelo menos um valor igual aos valores da mesma propriedade no formato mestre) e solicita ao formato escravo que faça a verificação (operação *matchProperty*);
- promove a seleção do melhor valor para cada propriedade (função *select* no formato mestre seguida da função *constraint* no formato escravo, utilizando como parâmetro os valores atribuídos pela função *select*);

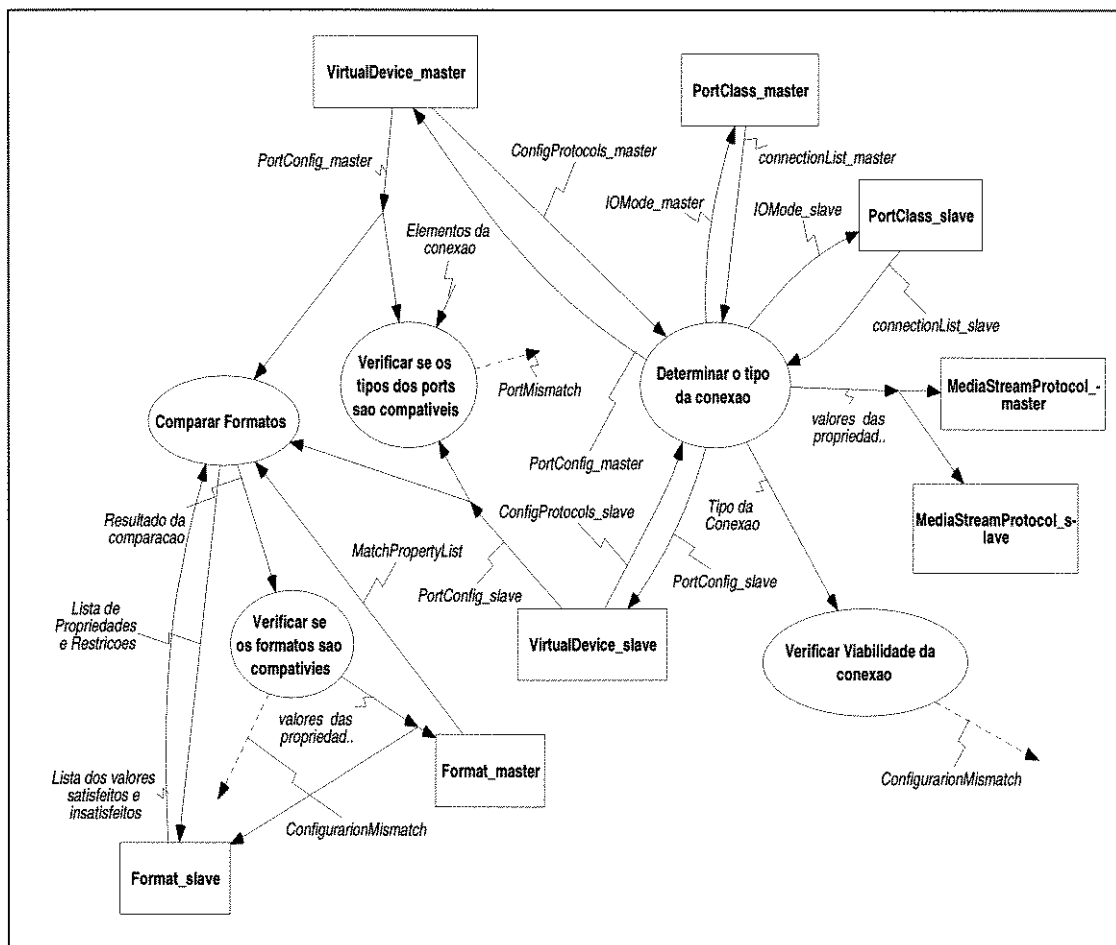


Figura 4-5: Estabelecimento da conexão. DFD nível um.

A propriedade *MatchPropertyListK* não faz parte da especificação. Ela foi adicionada ao tipo *PropertyConstraint* (supertipo de todos os objetos de configuração). Sua finalidade é armazenar o nome das propriedades que devem ser verificadas em testes de compatibilidade, já que nem todas são relevantes. Por exemplo, a propriedade *InternetLocation* herdada do tipo básico

EnhancedPREMOObject não diz respeito à tipo de mídia e, portanto, não deve ser utilizada na comparação de formatos.

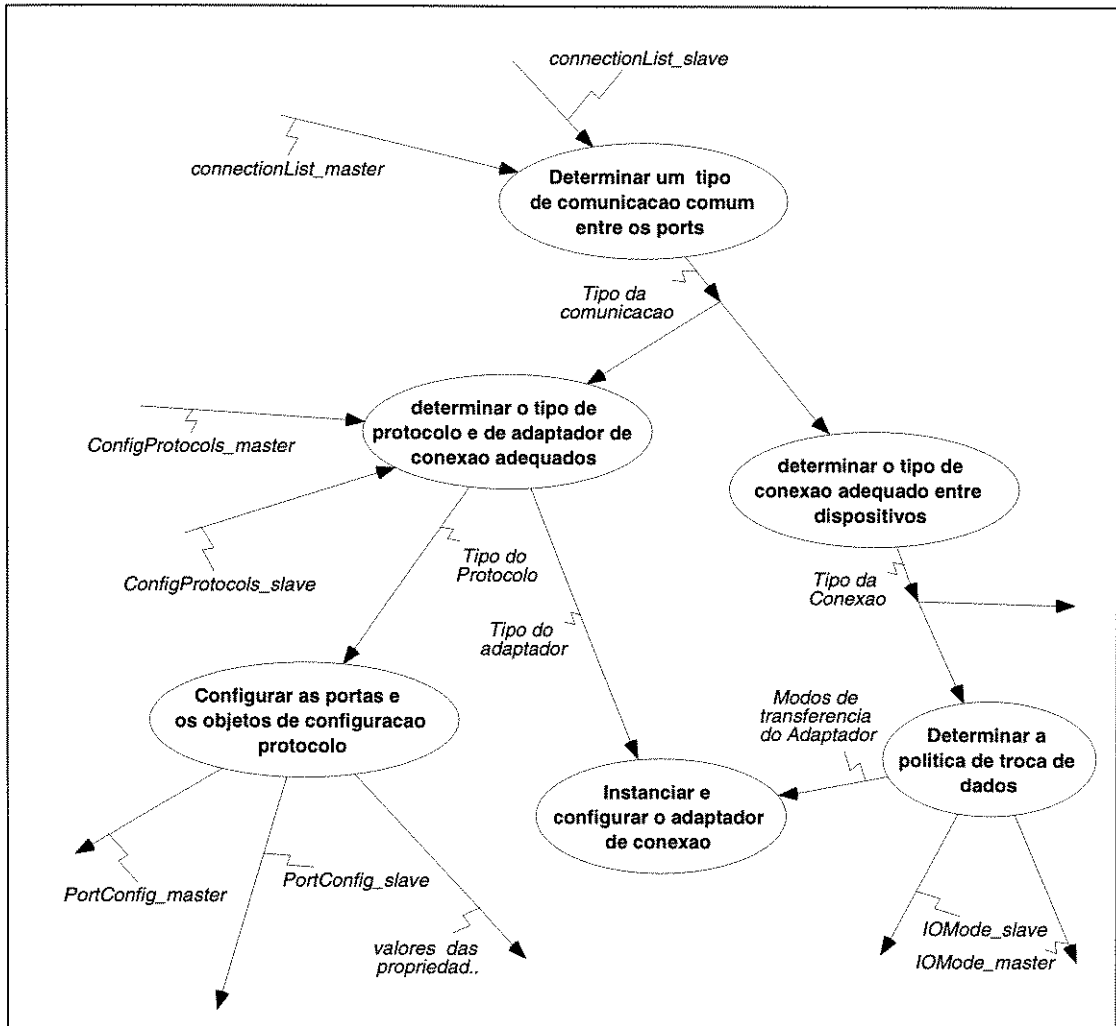


Figura 4-6: Estabelecimento da conexão. DFD nível dois

b) Tipo da conexão

A determinação do tipo da conexão envolve a avaliação de diversas características em diferentes objetos, como pode ser visto em Figura 4-5 e Figura 4-6. Inicialmente, a conexão virtual consulta as portas para identificar as formas de comunicação suportadas por elas (atributo *connectionList* descrito na seção 4.4.4.2) e então busca um tipo comum e que seja adequado à localização dos dispositivos (máquina/processo). A partir do tipo de comunicação é deduzido o tipo da conexão.

Em seguida, a conexão consulta os dispositivos virtuais para conhecer os objetos de configuração protocolo (atributo *configurationProtocols*) e procura uma interseção entre os tipos do protocolo e os tipos de transporte por ela suportados. Encontrando um tipo comum, a conexão realiza a configuração das características dos objetos, equivalentemente ao processo realizado com os formatos, configura a porta

com o respectivo objeto protocolo e restringe sua propriedade *TransportType* para o tipo escolhido. Se a conexão for em rede, um adaptador que suporta o tipo de transporte determinado é instanciado.

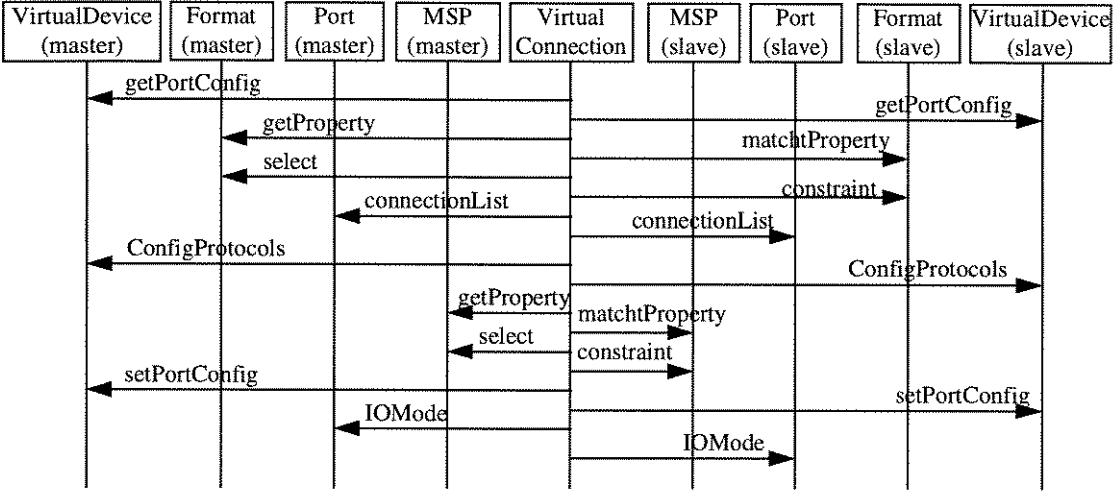


Figura 4-7: Interações para o estabelecimento da conexão.

c) *Qualidade de Serviço*

Para estabelecer a qualidade de serviço da conexão real, a conexão virtual deve procurar uma combinação entre as características dos objetos de configuração *QoS* e interagir com o Gerente de Recursos do sistema para averiguar a disponibilidade de recursos e então reservá-los. Por este motivo acredita-se que o estabelecimento desta parte do acordo deva ser realizado na fase de aquisição de recursos (operação *acquireResource*).

Entretanto, o gerenciamento de recursos está fora do escopo deste trabalho. Ele deve ser realizado por outro membro participante deste grupo. Como ainda não se conhece a interface para o objeto gerente de recursos, decidiu-se que esta parte do acordo não seria implementada.

d) *Capacidades de Sincronização e Stream*

Dos parâmetros relacionados neste item do acordo (seção 4.2.2), só será verificado aquele que envolve o estabelecimento do mecanismo de troca de dados, uma vez que questões de sincronização não são tratadas neste trabalho. Para isto foi criado um modelo que permite definir o comportamento dos elementos envolvidos no fluxo, ou seja, as portas dos dispositivos e o adaptador de conexão virtual.

Um objeto porta pode ter diversos comportamentos, dependendo de seu tipo, *input* (entrada) ou *output* (saída) e do valor de seu atributo *IOMode*, como descrito a seguir:

- *tipo = input; IOMode = DataDriven* - porta recebe os dados do adaptador de conexão que utiliza-se a função *putData*;

- ***tipo = input; IOMode = ControlDriven*** - porta obterá os dados em um objeto destino (outra porta ou adaptador), cuja identificação é passada pela conexão virtual através da função *addTarget*.
- ***tipo = output; IOMode = DataDriven*** - porta apenas disponibiliza os dados que serão retirados via uma chamada de função *getData*;
- ***tipo = output; IOMode = ControlDriven*** - porta irá enviar seus dados para um destino, cujas identificações serão passadas pela conexão virtual através da função *addTarget*.

Por outro lado, o adaptador pode ter as seguintes configurações:

- ***InputMode = ControlDriven; OutputMode = ControlDriven*** - o adaptador é o responsável por retirar os dados da porta de saída e entregá-los na porta de entrada. As identificações das portas e das funções a serem chamadas são passadas pela conexão virtual através das operações *setSource* e *setTarget*;
- ***InputMode = ControlDriven; OutputMode = DataDriven*** - o adaptador retira os dados da porta de saída (identificação passada através da operação *setSource*) mas não entrega na porta de entrada, a qual deve retirá-los chamando a operação *recv* da interface do adaptador;
- ***InputMode = DataDriven; OutputMode = ControlDriven*** - o adaptador recebe os dados que a porta de saída transmite (através da operação *send*) e entrega-os na porta de entrada;
- ***InputMode = DataDriven; OutputMode = DataDriven*** - o adaptador apenas recebe e transmite os dados na rede. Os dados são passados a ele pela porta de saída e retirados pela porta de entrada.

Este modelo permite estabelecer diversos modos de transferência de dados, e conseqüentemente, escolher o mais adequado para diversas situações, mesmo que não seja necessária a presença do adaptador de conexão (a operação *addTarget* do dispositivo virtual poderia ser utilizada para identificar uma outra porta de dispositivo e permitir que as duas portas troquem dados diretamente).

Estabeleceu-se que no caso de conexão em rede, o transporte deve ser realizado pelo adaptador, ou seja, ele opera com os atributos *InputMode* e *OutputMode* com valores *ControlDriven*. Desta forma, o dispositivo virtual de captura apenas disponibiliza seus dados na porta e o adaptador transporta-os até a porta do dispositivo de apresentação, como mostra a Figura 4-8. A vantagem desta configuração está no fato de que dispositivos virtuais ficam totalmente desacoplados do transporte de dados. Com isto permite-se:

- redução da complexidade do processamento do dispositivo virtual;
- escolha da melhor configuração, dependendo do tipo da conexão;

- desenvolvimento de dispositivos virtuais adaptáveis a qualquer ambiente de transporte, uma vez que troca de dados é feita através de uma interface bem definida (*PortClass*).

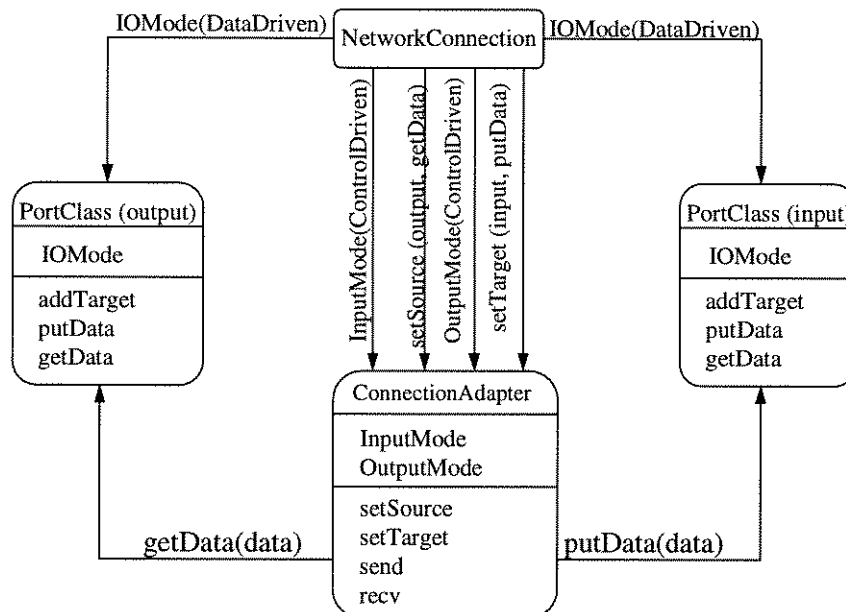


Figura 4-8: Mecanismo de troca de dados para conexão em rede.

4.4.5.2. O Stream

O *Stream* será o responsável por comandar dispositivos (*stream* de dispositivo) ou adaptador (*stream* da conexão), atuando como um “relógio”. As informações sobre qual será o elemento a ser comandado (elemento de sincronização) e o tempo com que os comandos devem ser executados são passadas ao objeto *stream* através de uma função denominada *setParameters*. Esta função não faz parte da especificação, mas faz as vezes da função *setSyncElement*, definida no tipo *TimeSynchronizable*, que é um supertipo de todo objeto *Stream* e está definido em [PREMO96b]. Através do *stream* é possível também obter informações sobre a posição do fluxo, utilizando-se para isto o método *getPosition* (equivalente ao atributo *currentPosition* definido em *TimeSynchronizable*).

Todavia, não será implementada a hierarquia de tipos da qual o *Stream* descende (Figura 3-6), pois esta trata questões de relógio e sincronização, as quais fogem ao escopo definido inicialmente. Além disto, os objetos mais altos da hierarquia são definidos como objetos fundamentais do PREMO e este trabalho está direcionado aos objetos do MSS definidos nas Parte 3 do PREMO. Implementar as abstrações de sincronização oferecidas pelos objetos básicos do PREMO é uma das opções para continuidade deste trabalho.

A interface do Stream que se propõe implementar é ilustrada na Figura 4-9. Pode-se verificar que estão definidas três operações para controle do fluxo: *play*, *stop*

e *pause*. Um tipo especializado de *Stream* foi criado para a conexão em rede e denomina-se *NetworkStream*. Outros tipos também foram criados para os dispositivos virtuais que são implementados como exemplo. O *NetworkStream* é o responsável por transmitir comandos recebidos do grupo ou do cliente ao adaptador que por sua vez realiza a transmissão dos dados. Os *streams* de dispositivos comandam o processamento interno dos respectivos dispositivos virtuais que os incluem.

Stream
play() stop() pause() getPosition() : long setParameters(str_ref : string, period : long)

Figura 4-9: Interface do tipo *Stream* em notação OMT.

A Figura 4-10 mostra os estados e as transições que definem o comportamento dos *streams* desta proposta.

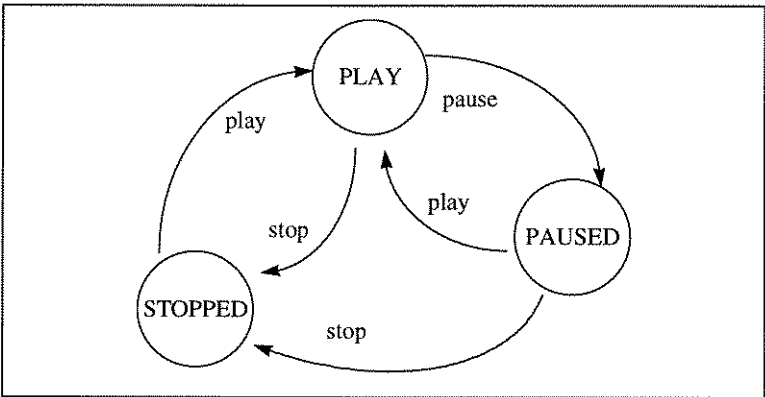


Figura 4-10: Diagrama de transição de estados para o *Stream*.

A Figura 4-11 traz um cenário de conexão em rede para ilustrar a interação entre os diversos objetos na realização do fluxo de dados. Neste caso o adaptador é o responsável por fazer a transferência dos dados que o dispositivo virtual produtor encaminha à porta. Considerando a existência do grupo, o trabalho do cliente fica bastante simplificado: ao evocar o método *play* no *stream* do grupo, a chamada é propagada aos *streams* dos dispositivos e da conexão virtual. O *streams* do dispositivo produtor evoca a operação *process*, desencadeando a execução de processamento interno e o encaminhamento dos dados gerados para a porta (*putData*); O *stream* da conexão virtual, por sua vez, evoca o método *transfer* no adaptador, que retira os dados da porta de saída (*getData*) e entrega-os na porta de entrada (*putData*). Então, o dispositivo consumidor, que também teve seu método *process* disparado pelo respectivo *stream*, retira os dados da porta (*getData*) para alimentar seu processamento interno (exibição).

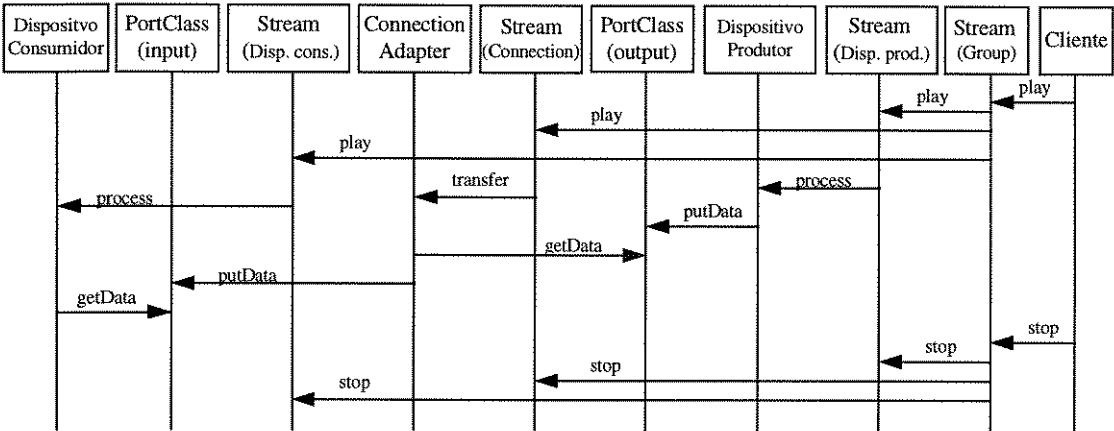


Figura 4-11: Diagrama de interações entre objetos durante o fluxo de dados.

4.4.6. O Modelo de Classes

A Figura 4-12 representa o modelo estático do sistema que se propõe implementar neste trabalho, segundo a metodologia OMT [Rumbaugh91]. São apresentadas as classes com o nome de suas operações e atributos, e o relacionamento estático entre elas, neste caso, somente herança e agregação. A inclusão de um objeto por outro, como por exemplo um dispositivo virtual inclui um *Stream*, é modelada com um relacionamento de agregação.

A classe *EnhancedPREMOObject* não faz parte da proposta de implementação pois consiste do trabalho de outro membro participante do grupo. Contudo, ela é exibida neste modelo para que a herança das interfaces seja preservada e permita o uso dos serviços básicos de propriedades. A integração da implementação da interface *EnhancedPREMOObject* ao protótipo proposto neste trabalho é descrita no Capítulo 5. Convém ressaltar que as propriedades de cada objeto não foram retratadas no modelo e serão descritas na fase de projeto e implementação.

A classe denominada *MSS* é uma conveniência para auxiliar o processo de criação dos recursos virtuais pelos clientes. Ela é prevista em uma especificação anterior [IMA94] e oculta o processo de localização de uma fábrica que seja capaz de instanciar o objeto, segundo as restrições passadas pelo cliente. A Figura 4-13 mostra o modelo de classes especializado, sendo que as classes folhas derivadas de *VirtualResource* (exceto a classe *LocalConnection*), *Stream*, *Format* e *MultimediaStreamProtocol* e *ConnectionAdapter* representam as classes utilizadas na criação da aplicação exemplo, que é descrita no capítulo seguinte.

O próximo capítulo descreve como o modelo aqui exposto foi realizado em um ambiente que suporta uma implementação da especificação CORBA. Serão apresentadas as características deste ambiente e as decisões de implementação influenciadas por elas.

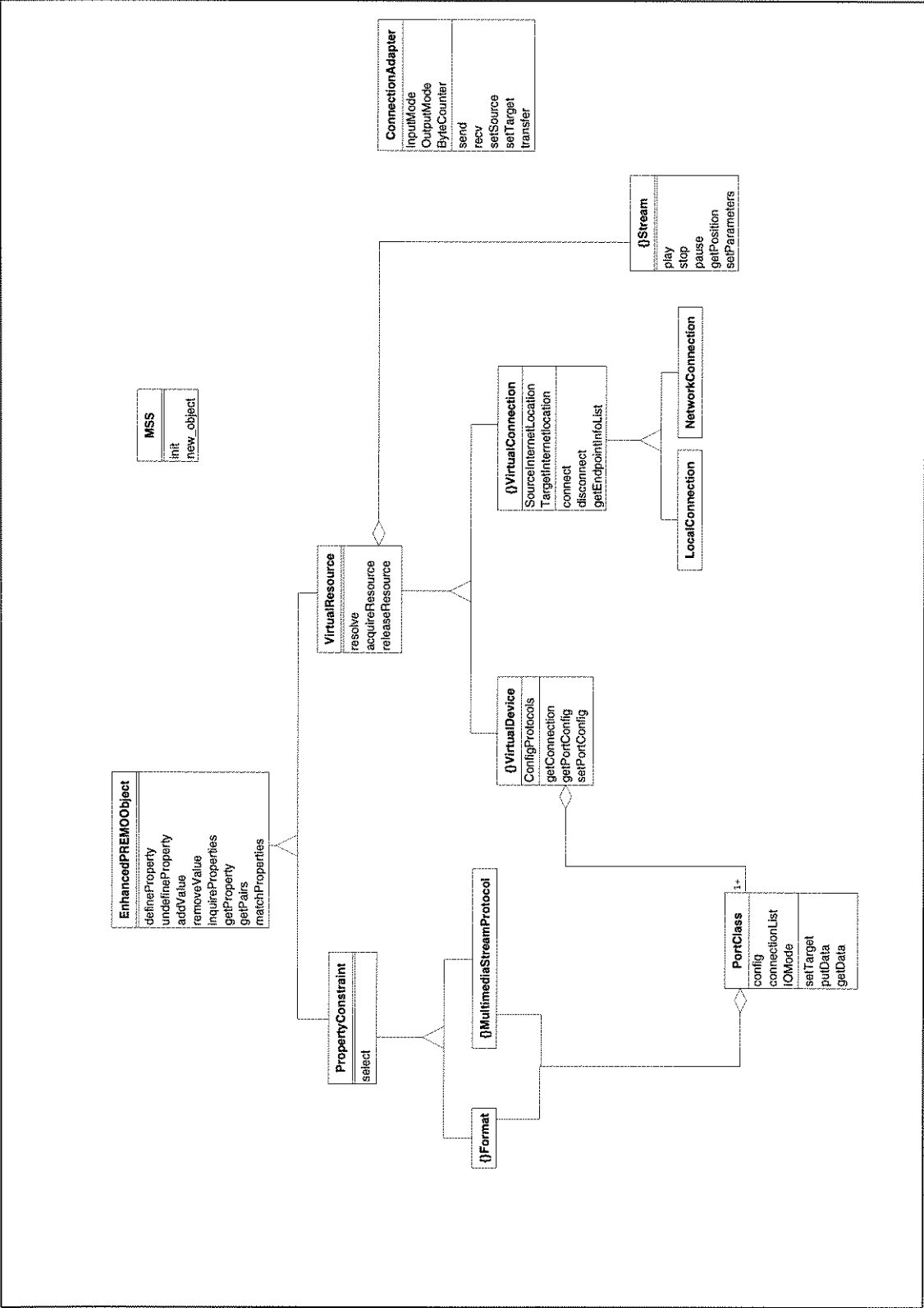


Figura 4-12: Modelo de Classes segundo a notação OMT.

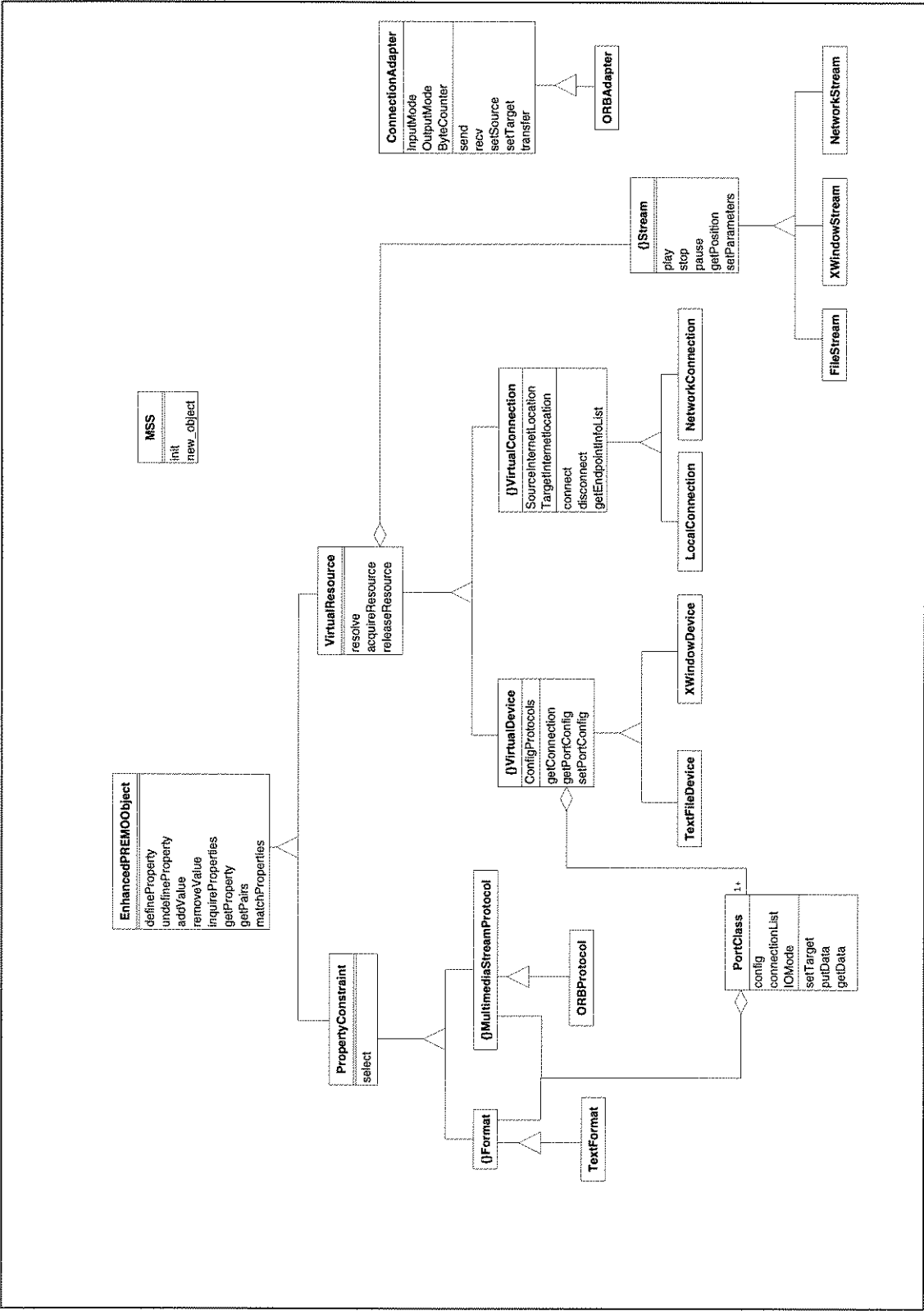


Figura 4-13: Modelo de Classes especializado.

5. Implementação

5.1. Introdução

Neste capítulo é apresentada a fase de projeto e implementação de um protótipo do *MSS*, focalizando a conexão virtual, objeto central das discussões deste trabalho.

O ambiente de implementação constitui-se de:

- estações de trabalho IBM RISC/6000;
- sistema operacional AIX versão 3.25;
- software para suporte aos objetos distribuídos Orbix. versão 1.3;
- linguagem de programação C++, com compilador C Set ++ da IBM.

A utilização do Orbix foi motivada pelo fato de ser uma implementação comercial da arquitetura CORBA e a escolha da linguagem C++ deve-se a dois fatores: ela é orientada a objetos, permitindo aplicação dos conceitos da análise com maior facilidade; por restrição desta versão do Orbix, que possui apenas mapeamento IDL-C++.

As versões citadas de sistema operacional e do Orbix possuem algumas limitações as quais impactaram em restrições para implementação. A maior delas é a ausência de *multi-thread*, um ponto fundamental devido à necessidade de execução de atividades concorrentes pelos objetos. Os impactos causados por estas limitações serão discutidos ao longo da descrição da implementação e estão resumidas na seção 6.3.

5.2. Escopo do Protótipo

O protótipo tem a função de validar a arquitetura da conexão virtual proposta no Capítulo 4, concentrando-se esforços nas questões de multimídia distribuída. Sendo assim, decidiu-se implementar apenas os tipos *VirtualConnection* e *NetworkConnection* da hierarquia da conexão virtual (Figura 4-1).

Para testar as interações da conexão virtual com outros objetos, foram criados dois dispositivos virtuais simples, sem grandes aspirações multimídia, primeiro porque programação de recursos multimídia excede o escopo deste trabalho; segundo porque não havia *hardware* nem *software* especializados que permitissem a elaboração de dispositivos virtuais sofisticados para processamento de, por exemplo, áudio ou vídeo. Os dispositivos virtuais do exemplo foram concebidos com seus respectivos *stream* e objetos de configuração tipos formato e protocolo, para fins de teste.

O adaptador de conexão virtual foi elaborado como um subtipo de *ConnectionAdapter*. Ele foi implementado utilizando o próprio ORB como veículo de transporte e não leva em consideração questões de qualidade de serviço.

As propriedades dos objetos de configuração, assim como da conexão virtual, serão iniciadas estaticamente através de uma função evocada no construtor de cada classe (na realidade pelo construtor da classe base *EnhancedPREMOObject*). Isto tornou-se necessário pois não foi implementado um mecanismo de registro (em arquivo ou banco de dados) que permitisse armazenar as propriedades de cada classe. Não foram implementadas também fábricas que criassem os objetos e atribuíssem a eles os valores correntes de suas propriedades, levando em conta o registro realizado previamente e as limitações do sistema. Convém lembrar que esta implementação fazia parte do trabalho de outro membro do grupo, o mesmo que implementou a classe *EnhancedPREMOObject*.

5.3. Projeto e Implementação

O primeiro passo nesta fase consiste em gerar as interfaces IDL, tomando como base as classes do modelo OMT e os esquemas em *Object-Z*. O mapeamento é bastante simples, quase direto. Além de atributos e operações, a sintaxe da linguagem IDL permite definir restrição de acesso a atributos, assim como exceções e política de sincronização na chamada das operações da interface (isto é, se o cliente de uma chamada fica bloqueado ou não até que ela se complete). Detalhes sobre a linguagem IDL (sintaxe e mapeamento para C++) são encontrados em [Orbix95c].

Uma vez definidas as interfaces IDL, segue-se o processo de desenvolvimento descrito na seção 2.6.6. O compilador IDL gera uma classe IDL-C++ contendo informações necessárias à invocação dos métodos de cada interface. O código que implementa os serviços da interface, assim como operações internas não acessíveis ao cliente, é escrito em uma classe C++ denominada classe de implementação. A partir dela constrói-se o servidor e instanciam-se objetos Orbix. Portanto, cada classe IDL-C++ será mapeada em alguma classe de implementação. Convencionou-se para este trabalho que a classe de implementação teria o mesmo nome da classe IDL-C++ seguido de “_i”.

Nas próximas subseções serão discutidas as interfaces IDL e as respectivas classes de implementação dos elementos discutidos no Capítulo 4, além das classes

geradas para fins de teste. Antes, porém, serão declarados os tipos e estruturas de dados utilizados para esta implementação.

5.3.1. Tipos e Estruturas de Dados

Da mesma maneira que se definem interfaces em IDL, pode-se também definir tipos e estruturas de dados, com a facilidade de que *stubs* e *skeletons* se encarregam de fazer *marshalling* e *unmarshalling* dos parâmetros de operações. Os seguintes tipos foram definidos para esta implementação:

- ***StmBuffer***: sequência de caracteres. Definido a partir do tipo *sequence* de IDL. É o tipo usado nos parâmetros das operações *getData/putData* da porta e *send/recv* do adaptador. Os dados são passados dentro de uma sequência de caracteres;
- ***TransferMode***: tipo enumerado que indica o modo de transferência de dados. Pode assumir os valores *DataDriven* e *ControlDriven*;
- ***Port***: tipo *short* usado como índice na identificação das portas de um dispositivo virtual;
- ***PortType***: tipo enumerado que indica o tipo da porta, *Input* ou *Output*;
- ***ConnectionType***: tipo enumerado que indica o tipo da conexão retornado pela função *type_of_connection*. Os valores possíveis são *hardware*, *direct*, *local*, *network* e *none*;
- ***EndpointInfo***: estrutura composta por três campos. Um apontador para dispositivo virtual, índice identificador da porta do dispositivo e indicação (booleana) se a porta é mestre ou não;
- ***EndpointInfoList***: sequência do tipo *EndpointInfo*;
- ***ConfInfo***: representa informações sobre objetos de configuração. É uma estrutura composta por duas *strings*: *SemName* constitui um nome significativo para identificar o objeto; *ObjectType* identifica o tipo do objeto. Estes campos são obtidos de uma *string* que representa uma referência para objeto. A Figura 5-1 mostra o formato da *string* referência de objeto (gerada através de uma função válida para todo objeto CORBA) e como são compostos os campos da estrutura *ConfInfo*. Conhecendo-se a *ConfInfo* de um objeto de configuração pode-se obter um apontador para ele através da operação *resolve* (disponível em todo recurso virtual);
- ***ConfInfoList***: sequência do tipo *ConfInfo*;
- ***PortConfig***: como restrição para este protótipo, a estrutura *PortConfig* conterá apenas a identificação do tipo da porta, apontador para o *stream* e *ConfInfo* para o objeto de configuração protocolo e para um formato

somente (não será permitida a alteração de formato durante a execução da aplicação);

5.3.2. Raiz da Hierarquia de Classes

Todo dispositivo e toda conexão virtual são recursos virtuais, que por sua vez são *EnhancedPREMOObject*. As operações de *EnhancedPREMOObject* declaradas no modelo de classes (Figura 4-12) foram implementadas em outro trabalho. As primeiras classes derivadas de *EnhancedPREMOObject* foram *VirtualResource* e *PropertyConstraint*.

Para o tipo *VirtualResource* foi declarada a seguinte interface IDL:

```
interface VirtualResource : EnhancedPREMOObject {
    exception ResourceNotAvailable{};
    readonly attribute Stream resource_stream;
    PropertyConstraint resolve(in string semanticName,
                              in string type);
    void acquireResource raises (ResourceNotAvailable);
    void acquireResource raises (ResourceNotAvailable);
    void releaseResource();
};
```

As operações *acquireResource* e *releaseResource* foram declaradas na classe de implementação como virtuais puras, necessitando portanto, de redefinição e implementação nas folhas da hierarquia de classes.

A operação *resolve* concatena as duas *strings* que recebe como parâmetro, montando assim uma *string* referência para objeto; realiza a ligação (*bind*) com o objeto destino e retorna um apontador para um objeto do supertipo *PropertyConstraint*, que deve ser convertido para o tipo específico do objeto de configuração esperado.

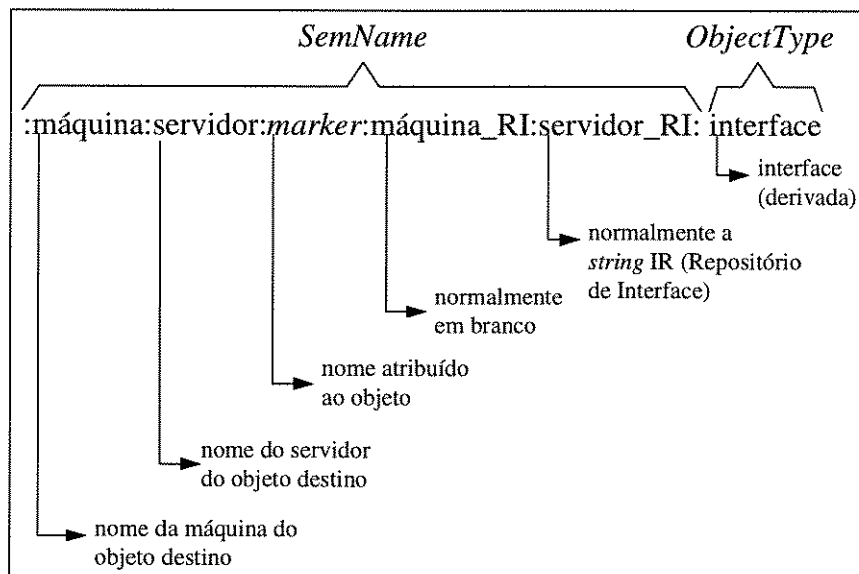


Figura 5-1: Formato de uma string referência de objeto.

PropertyConstraint é o supertipo de todo objeto de configuração. Para fins de teste, uma versão simplificada de sua interface foi declarada, como pode-se ver abaixo. Nela é definida a operação *select*, a qual deveria fazer a seleção dos valores de propriedades mais adequados a partir da interseção entre conjunto de valores indicados no argumento *constraint* e os valores correntes das propriedades do objeto. Para isto deveria ser implementado um algoritmo capaz de realizar tal escolha. Entretanto, isto ultrapassa o escopo do trabalho e este método apenas retornará a mesma sequência de propriedades passada como parâmetro.

```
interface PropertyConstraint : EnhancedPREMOObject {
    void select(in KeyValues constraints,
               out KeyValues currentValues);
};
```

5.3.3. Conexão Virtual

A interface da conexão virtual descrita na Figura 4-12 e no esquema do Anexo A é transcrita para IDL da seguinte maneira:

```
interface VirtualConnection : VirtualResource {
    exception ConfigurationMismatch { ConfInfo master;
                                     ConfInfo slave; };
    exception PortMismatch { string reason; };
    void connect(in VirtualDevice deviceMaster, in Port portMaster,
                in VirtualDevice deviceSlave, in Port portSlave);
        raises (ConfigurationMismatch, PortMismatch);
    void disconnect();
    EndpointInfoList getEndpointInfoList();
};
```

A classe C++ que implementa esta interface é:

```
class VirtualConnection_i: public virtual VirtualResource_i {
protected:
    EndpointInfoList *connection_points;
    VirtualDevice *devMaster, *devSlave;
    unsigned short port_master, port_slave; //identif. das portas
    PortConfig config_master, config_slave; //config. das portas
    unsigned char compare(ConfInfo& ci_master, ConfInfo& ci_slave);
    virtual ConnectionType type_of_connection() = 0;
    SequenceString* match_lists (SequenceString listMaster,
                                SequenceString listSlave);

public:
    // construtor
    VirtualConnection_i();
    //destrutor
    ~VirtualConnection_i();
    virtual void connect(VirtualDevice* deviceMaster,
                        Port portMaster, VirtualDevice*
deviceSlave,
                        Port portSlave);
    virtual void disconnect ();
    virtual EndpointInfoList getEndpointInfoList ();
};
```

Note que ela possui três operações adicionais. Entretanto, elas não são acessíveis por clientes da interface:

- **compare** - função designada à verificação da compatibilidade dos objetos de configuração identificados por `ci_master` e `ci_slave`
- **match_lists** - função designada à determinação de elementos comuns entre duas seqüências de strings.
- **Type_of_connection** - todo código a respeito de decisões sobre o tipo da conexão e a política de troca de dados fica concentrado neste ponto; é uma função virtual pura, ou seja, não possui código nesta classe e deve ser redefinida e implementada em classes derivadas da *VirtualConnection_i*. Com isto obtém-se:
 - ✓ que *VirtualConnection* torna-se um tipo abstrato, como estabelece a especificação;
 - ✓ encapsulamento e reuso de código. Quando um subtipo de *VirtualConnection_i* for criado (por exemplo *NetworkConnection_i*), nele será implementada esta operação de forma a adequá-la às necessidades deste tipo de conexão, sem que seja necessário reescrever todo o código da operação *connect*.

As variáveis internas contêm:

- **connection_points**: identificação dos elementos que a instância da conexão conecta. Estes elementos são acessíveis ao cliente através da operação *getEndpointInfoList*;
- **devMaster e devSlave**: apontador para os dispositivos virtuais passados como parâmetro na operação *connect*;
- **port_master e port_slave**: identificação (índice) das portas nos dispositivos virtuais;
- **config_master e config_slave**: configuração das portas dos dispositivos virtuais.

5.3.3.1. NetworkConnection

Na interface IDL de *NetworkConnection* nada foi acrescentado. Apenas a herança de *VirtualConnection* é declarada, como pode ser ver abaixo

```
interface NetworkConnection : VirtualConnection {};
```

Entretanto, na classe de implementação, alguns métodos herdados foram sobre-escritos e novos métodos internos foram definidos:

```
class NetworkConnection_i: public virtual VirtualConnection_i {
private:
    NetworkStream *my_net_stream;
    ORBAdapter *orb_adapter;
    unsigned short stream_interval;
    virtual ConnectionType type_of_connection();
```

```
Properties *initialize_transport_property();
public:
    // construtor
    NetworkConnection_i();
    // destrutor
    ~NetworkConnection_i();
    virtual void acquireResource ();
    virtual void releaseResource ();
};
```

type_of_connection: sobre-escreve a operação declarada na classe *VirtualConnection_i*, para tratar apenas questões de conexão em rede;

initialize_transport_property: inicia a propriedade *TransportTypes* da conexão virtual. A única propriedade atribuída ao tipo *NetworkConnection* foi a *TransportTypes*, tendo como valor possível, “ORBProtocol” (transporte através do ORB).

O valor de retorno da operação *initialize_transport_property* constitui um apontador para uma sequência de elementos do tipo *Property*, o qual é definido pela seguinte estrutura:

```
struct Property {
    char * key;
    unsigned char is_read_only;
    TypeCode type;
    any capability;
    any current;
}
```

A estrutura *Property* foi estabelecida pelo mesmo membro do grupo que realizou a implementação do tipo *EnhancedPREMOObject* e não será discutida em detalhes aqui. Todas as operações criadas para iniciar propriedades nos diversos objetos deste protótipo têm a mesma funcionalidade: preencher os campos da estrutura *Property* para cada propriedade; criar uma sequência com todas as propriedades do objeto; retornar um apontador para a sequência de propriedades.

Os métodos *acquireResource* e *releseResource* não desempenham seu papel verdadeiro. Neles deveriam ser realizadas respectivamente reserva e liberação de recursos de rede junto ao Gerente de Recursos, levando em consideração a configuração dos objetos *QoS* da própria conexão virtual e das portas dos dispositivos. Entretanto, tratamento de qualidade de serviço foge ao escopo deste trabalho e, além disto, a interface para o Gerente de Recursos ainda é desconhecida (deveria ser estabelecida por outro membro do grupo). Em sua implementação atual, *acquireResource* executa apenas algumas funções de configuração, como atribuir os elementos e o período de sincronização aos *streams* dos dispositivos e da conexão virtual.

5.3.4. Dispositivos Virtuais

O tipo *VirtualDevice* é supertipo de todos os dispositivos virtuais. Para este protótipo, os seguintes atributos e operações foram declarados para sua interface:

```
interface VirtualDevice : VirtualResource {
    readonly attribute ConfInfoSequence ConfigProtocols;
    PortConfig getPortConfig (in Port portId);
    void setPortConfig (in Port port, in PortConfig portConfig);
    PortClass getPortObject (in Port portNumber);
    oneway void process();
};
```

A operação *getPortObject* foi acrescentada à interface para permitir que a conexão virtual adquira a referência dos objetos *PortClass* com os quais deve interagir. A operação *process*, que implementa as funcionalidades específicas dos dispositivos, é declarada como *oneway* para que não bloqueie o objeto *Stream* enquanto o dispositivo executa a operação. O controle do dispositivo virtual pelo *stream*, através da operação *process*, é necessário uma vez que o ambiente não é *multi-thread* (se a operação *process* fosse disparada indefinidamente, o servidor ficaria bloqueado e nenhuma outra requisição subsequente seria tratada).

A classe de implementação *VirtualDevice_i* é mostrada abaixo. Os objetos *PortClass* de um dispositivo são criados e armazenados durante a execução do construtor da classe *VirtualDevice_i*. O construtor de cada classe derivada deve passar como parâmetro ao construtor da classe base o número de portas que o dispositivo possui. Cabe também às classes derivadas a tarefa de configurar as portas com as informações corretas para os atributos *config* e *connectionList* (veja a descrição da interface *PortClass* na seção 5.3.4.3).

```
class VirtualDevice_i : public virtual VirtualResource_i {
protected:
    PortTable *port_table;
    Stream *my_stream;
    ConfInfoSequence config_protocols;
public:
    // construtor
    VirtualDevice_i(unsigned short table_size);
    // destrutor
    ~VirtualDevice_i();
    virtual ConfInfoSequence ConfigProtocols ();
    virtual VirtualConnection* get_connection (Port port);
    virtual PortConfig getPortConfig (Port portId);
    virtual void setPortConfig (Port port,
                                const PortConfig& portConfig);
    virtual PortClass* getPortObj (Port portNumber);
    virtual void process () = 0;
};
```

Na gerência de suas portas, o dispositivo virtual utiliza-se de um objeto auxiliar instanciado a partir da classe *PortTable*. Este objeto cria um vetor de apontadores para objetos *PortClass* e oferece operações para atribuir (*setPort*) e recuperar (*getPort*) um apontador em uma posição do vetor. A posição do apontador no vetor subentende o índice da porta. A classe de implementação *PortTable_i* é mostrada abaixo.

```
typedef PortClass *PortPtr;
class PortTable_i {
private:
```

```
PortPtr *table;
unsigned short length;
public:
    // construtor
    PortTable_i (unsigned short size);
    // destrutor
    ~PortTable_i ();
    virtual void setPort (Port port_number, PortClass* port_obj);
    virtual PortClass* getPort (Port port_number);
};
```

Como exemplos de dispositivos virtuais foram criados os dispositivos *TextFileDevice* para captura de dados e *XWindowDevice* para apresentação, os quais serão detalhados na sequência.

5.3.4.1. *TextFileDevice*

A interface definida para este dispositivo é bastante simples. São declaradas duas constantes para facilitar a identificação das portas e um atributo que receberá o nome do arquivo a ser lido, como pode ser visto a seguir.

```
Interface TextFileDevice : VirtualDevice {
    const unsigned short InputPortC = 0;
    const unsigned short OutputPortC = 1;
    attribute string filename;
};
```

Por outro lado, sua classe de implementação traz várias redefinições de operações herdadas e novas operações internas.

```
class TextFileDevice_i: public virtual VirtualDevice_i {
private:
    FILE *pf;
    char *fname;
    TextFormat *text_format; // objeto de configuracao formato
    ORBProtocol *orb_protocol; // objeto de configuracao protocolo
    PortConfig port_config;
    FileStream *my_file_stream;
    unsigned short stream_interval;
    Properties *initialize_format_properties ();
    Properties *initialize_protocol_properties ();
public:
    // construtor
    TextFileDevice_i ();
    // destrutor
    ~TextFileDevice_i ();
    static const unsigned short InputPortC;
    static const unsigned short OutputPortC;
    virtual Stream* resource_stream ();
    virtual void filename (const char * filename);
    virtual char * filename ();
    virtual void acquireResource ();
    virtual void releaseResource ();
    virtual void process ();
};
```

Durante a execução do construtor desta classe, são criados e iniciados os objetos de configuração para formato e protocolo, assim como a porta de saída do dispositivo. É realizada também a associação com um objeto *FileStream* (tipo de *Stream* definido especificamente para este dispositivo).

Um dos métodos *filename* permite a atribuição e o outro permite a obtenção do valor do atributo *filename* definido na interface IDL (no mapeamento de IDL para C++, um atributo é convertido para uma ou duas operações, dependendo se ele é só para leitura ou para leitura e escrita).

Os métodos definidos como virtuais puros (ou abstratos) nas classes superiores *VirtualResource* e *VirtualDevice* são implementados com as seguintes funcionalidades:

- ***acquireResource***: abre o arquivo para leitura;
- ***releaseResource***: fecha o arquivo;
- ***process***: realiza a leitura do arquivo, linha a linha, e coloca os dados lidos na porta de saída do dispositivo.

O método privado *initialize_format_properties* cria a lista de propriedades do objeto de configuração formato. O valor de retorno desta operação é utilizado como parâmetro no construtor de *TextFormat* (classe de formato específica criada para este protótipo) que por sua vez transfere ao construtor de sua classe base *EnhancedPREMOObject*. De forma análoga, a operação *initialize_protocol_properties* realiza a atribuição das propriedades de um objeto de configuração protocolo, do subtipo *ORBProtocol*.

As propriedades criadas para as instâncias de *TextFormat* e *ORBProtocol* e seus respectivos valores são:

- *TextFormat*
 - ✓ NameK: TextFormat;
 - ✓ EncodingK: ASCII, ISO;
 - ✓ MatchPropertyListK: EncodingK.
- *ORBProtocol*
 - ✓ NameK: ORBProtocol;
 - ✓ VersionK: 1;
 - ✓ MatchPropertyListK: VersionK.

5.3.4.2. *XWindowDevice*

A interface IDL de *XWindowDevice* também é bastante simples e acrescenta apenas uma constante à interface herdada de *VirtualDevice*. Esta constante define apenas o índice de sua porta de entrada de dados.

A classe de implementação, *XWindowDevice_i* possui os mesmos métodos que a classe *TextFileDevice_i*, com exceção daqueles relacionados ao atributo *filename* e sobreescreve os mesmos três métodos, de tal forma que:

- *acquireResource* e *releaseResource* não realizam nenhum processamento;
- *process* retira os dados da porta de entrada e exibe-os no terminal de vídeo.

Além disso, realiza um processo de iniciação e configuração análogo ao de *TextFileDevice_i*.

5.3.4.3. *PortClass*

A interface proposta para a porta no capítulo anterior foi mapeada para IDL da seguinte maneira:

```
interface PortClass {  
    attribute PortConfig config;  
    attribute SequenceString connectionList;  
    attribute TransferMode IOMode;  
    void setTarget(in string target, in string function);  
    oneway void putData(in StmBuffer data);  
    void getData(out StmBuffer data);  
};
```

Detalhes de duas operações são destacados:

- *setTarget* recebe dois parâmetros - o primeiro é na verdade uma *string* referência de objeto (Figura 5-1) e o segundo o nome de uma operação que deve ser evocada em tal objeto.
- *putData* foi declarada como *oneway* para que as chamadas a este método sejam assíncronas. Desta forma, os clientes deste serviço (dispositivos virtuais ou adaptadores de conexão) não ficam bloqueados aguardando o término do processamento da operação, permitindo-lhes a execução de outras atividades .

5.3.5. Adaptador de Conexão Virtual

No mapeamento da interface proposta no Capítulo 4 para IDL, apenas detalhes de permissão dos atributos e sincronização na chamada de métodos foram acrescentados, como pode ser visto a seguir:

```

interface ConnectionAdapter {
    attribute TransferMode InputMode;
    attribute TransferMode OutputMode;
    readonly attribute unsigned long ByteCounter;
    void setSource(in string source, in string function);
    void setTarget(in string target, in string function);
    oneway void transfer();
    oneway void send(in StmBuffer stm_in);
    boolean recv (out StmBuffer stm_out);
};

```

A classe *ConnectionAdapter_i* contém a implementação das operações correspondentes aos atributos da interface, enquanto que os métodos propriamente ditos são implementados na classe derivada *ORBAdapter_i*. As operações *setSource* e *setTarget* são equivalentes à função *setTarget* da porta, descrita na seção 5.3.4.3.

A operação *send* é utilizada para enviar dados ao adaptador (evocada por uma porta de saída operando no modo *ControlDriven*) e não bloqueia a aplicação nem possui valor de retorno. Por outro lado, a operação *recv* (evocada por uma porta de entrada operando no modo *ControlDriven*), retorna (0) quando não há dados disponíveis no adaptador ou (1) caso contrário.

A operação *transfer* foi implementada especificamente para buscar os dados na porta de saída do dispositivo de captura e entregá-los na porta de entrada do dispositivo de apresentação, uma vez que para conexão em rede o adaptador opera sempre desta forma. A Figura 5-2 ilustra a execução da função *transfer* pelo adaptador (disparada através do *stream* da conexão) que utiliza o ORB para o transporte dos dados.

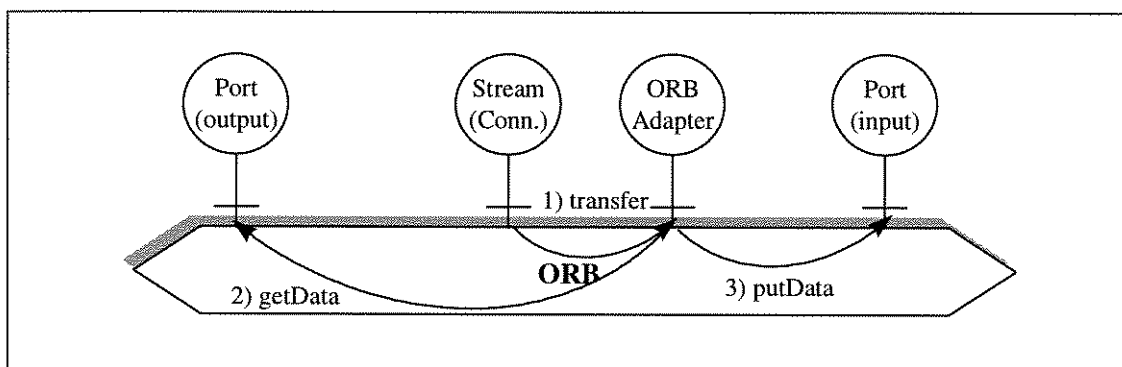


Figura 5-2: Adaptador realizando a transferência de dados

A abordagem utilizada para o controle das atividades do adaptador (na qual o *stream* da conexão virtual dispara as chamadas *transfer*) não é a mais adequada devido à sobrecarga causada no sistema de comunicação: estes objetos interagem com grande frequência mas estão em processos diferentes, e até mesmo em máquinas separadas. Outras abordagens foram estudadas para tornar o adaptador de conexão mais independente e são discutidas a seguir:

- inserir um objeto *stream* (objeto que abstrai as questões de tempo e sincronização) no próprio adaptador. Neste caso, a conexão virtual só

precisaria interagir com o *stream* do adaptador para repassar as chamadas de operações evocadas sobre seu próprio *stream*. Todavia, a restrição de *mono-thread* fazia recair sobre o mesmo problema: *stream* e adaptador deveriam ser instanciados como processos distintos, provocando a mesma sobrecarga de comunicação interprocessos.

- redefinir a interface do adaptador para que ela contenha operações de controle e implementá-lo como uma máquina de estados (equivalentemente ao *Stream*). Uma vez configurado, o adaptador ficaria indefinidamente transportando os dados de uma porta à outra, sendo interrompido apenas quando necessário.

5.3.6. Streams

A interface IDL definida para o tipo *Stream* será a mesma para todos os seus subtipos definidos nesta implementação, sendo ela a seguinte:

```
interface Stream {
    long getPosition();
    oneway void play();
    oneway void stop();
    oneway void pause();
    oneway void setParameters(in string str_ref,
                             in unsigned short period);
};
```

A implementação do *Stream* traz definições extra-classe, que serão compartilhadas pelo programa principal do servidor. As definições extras, assim como a classe de implementação *Stream_i* podem ser vistos abaixo.

```
enum Signal {SIG_PLAY, SIG_STOP, SIG_PAUSE, SIG_OTHER};
enum Status {PLAY, STOPPED, PAUSED};
Status var_status;
Signal var_signal;
long stream_position;
unsigned short transfer_period;
class Stream_i {
    protected:
        char * string_ref;
    public:
        // construtor
        Stream_i(){
            var_status = STOPPED;
            var_signal = SIG_OTHER;
        };
        // destrutor
        ~Stream_i() {};
        virtual long getPosition ();
        virtual void play ();
        virtual void stop ();
        virtual void pause ();
        virtual void setParameters (const char * str_ref,
                                    unsigned short period);
};
```

Os seguintes tipos são definidos:

- **Signal**: tipo enumerado utilizado na identificação de um sinal (ou evento). Seus valores possíveis são: SIG_PLAY, SIG_STOP, SIG_PAUSE e SIG_OTHER; os três primeiros valores indicam que algum cliente evocou o método correspondente; o último indica a ocorrência de qualquer outro evento do próprio ORB.
- **Status**: tipo enumerado utilizado na identificação do estado do objeto. Seus valores possíveis são: PLAY, STOPPED, PAUSED.

Das variáveis externas à classe são destacadas:

- **var_status**: identifica o estado corrente do objeto;
- **var_signal**: identifica o sinal recebido pelo objeto;
- **transfer_period**: período com que os comandos (chamadas de função) são disparados aos elementos de sincronização.

O processamento realizado pelos métodos *play*, *stop*, e *pause*, consiste apenas em alterar a variável *var_signal* (ex: uma chamada à *play* atribui SIG_PLAY à *var_signal*). A implementação concebida para a classe base *Stream_i* será herdada por todas as classes derivadas e o estado inicial dos objetos instanciados será STOPPED.

Cada uma das classes derivadas (*NetworkStream_i*, *FileStream_i* e *XWindowStream_i*) possui sua própria implementação para o método *setParameters*, pois nele é realizada a ligação (*bind*) para o elemento de sincronização específico, ou seja, dispositivo virtual ou adaptador de conexão.

O processamento de controle que cada *stream* deve executar, fica na realidade, no programa principal de cada servidor. Como a versão do Orbix empregada não permite programação *multi-thread*, tornou-se necessário implementar dois níveis de controle: um das atividades do elemento de sincronização e outro das requisições que o objeto recebe.

Ao receber uma requisição *play*, cada *stream* evoca o método correspondente em seu elemento de sincronização, e ao término, verifica se há eventos pendentes na fila de requisições do servidor. Em caso negativo, o *stream* continua com a execução das atividades do elemento de sincronização; caso contrário, processa a requisição pendente, identifica qual o tipo de sinal e o estado atual do dispositivo, para executar os procedimentos correspondentes.

5.3.7. Servidores

Como mencionado anteriormente, a plataforma de desenvolvimento possui limitações que impedem a programação paralela eficaz (com processos leves). Em função disto, até interfaces ligadas por relacionamentos de agregação (ou inclusão) tiveram suas implementações realizadas em servidores distintos quando as partes

necessitam realizar atividades de processamento concomitantes. Este é o caso do todos os recursos virtuais com seus respectivos *streams*. Para dispositivo virtual e seu respectivo *stream* são disparados dois processos pesados, bem como para conexão virtual e seu *stream*. A relação de agregação entre recurso virtual e *stream* é obtida através de uma ligação (*bind*) do objeto recurso com o objeto *stream* em outro servidor. A Figura 5-3 ilustra os objetos distribuídos sobre o ORB, para a realização da aplicação exemplo. Os objetos *streams* estão sombreados indicando que são processos servidores distintos na constituição dos serviços.

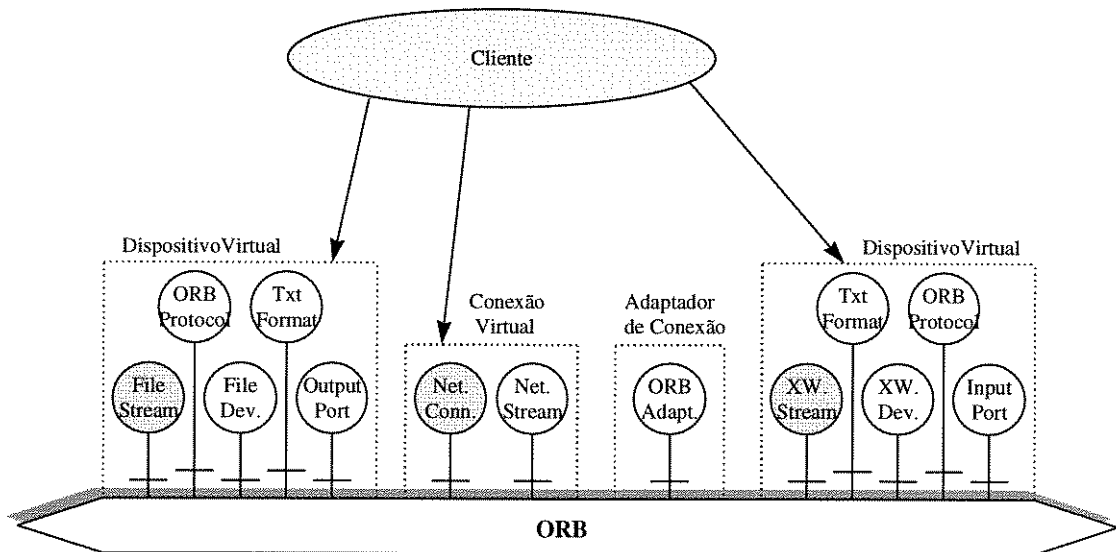


Figura 5-3: Distribuição da aplicação sobre o ORB.

Os servidores foram criados no modo compartilhado (*default* do Orbix), e são disparados automaticamente pelo ORB quando este recebe um pedido de ligação para o servidor. Além disto, os servidores foram disparados com *timeout* infinito (ele permanece indefinidamente aguardando requisições) para evitar que algum objeto seja desativado pelo Orbix e perca a configuração realizada anteriormente.

5.4. Exemplo de aplicação

Os tipo citados anteriormente são utilizados na construção de uma aplicação exemplo: exibição local de um arquivo texto remoto. A Figura 5-4 ilustra a visão do cliente sobre os objetos, os fluxos de controle e de dados para o exemplo proposto. Em seguida são descritos os passos empregados no desenvolvimento de uma aplicação, de maneira equivalente ao cenário da seção 3.5, sendo que abaixo acompanha o código gerado para ela. A Figura 5-3 ilustra a visão de implementação com relação ao ambiente distribuído Orbix/CORBA.

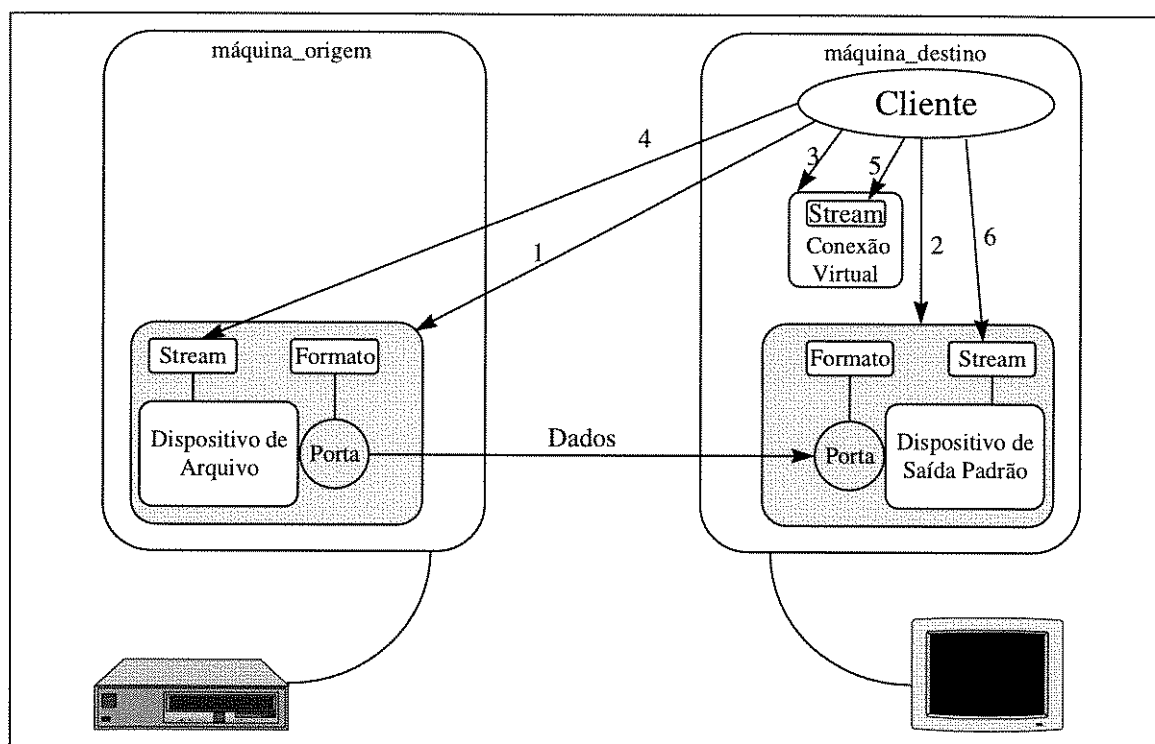


Figura 5-4: Utilizando o MSS na leitura e exibição de arquivo.

A aplicação cliente utiliza uma biblioteca de classes *MSS* (seus códigos são ligados). O cliente declara a biblioteca e a inicia. Através dela, a aplicação solicita a criação de objetos, estabelecendo restrições para os mesmos. Aqui faz-se necessária uma observação a respeito da função *new_object* da classe *MSS*. Ela é uma função de conveniência (especificada apenas em versão anterior do PREMO e do IMA) que simplifica a criação dos objetos, ocultando o processo de localização da fábrica.

As restrições são transmitidas através de uma sequência de *strings* pois ainda não existe um mecanismo adequado para construção de restrições. O significado das mesmas é interpretado de acordo com o tipo do objeto solicitado. O cliente solicita a criação do dispositivo de arquivo, indicando como restrições, a localização (na Figura 5-4, *máquina_origem* = "maresias.dca.fee.unicamp.br") e o nome do arquivo desejado. Ao receber a referência para o objeto fica estabelecido o fluxo de controle 1. Como a função *new_object* da biblioteca *MSS* retorna referências de objetos do tipo genérico *EnhancedPREMOObject*, o cliente deve convertê-la para o tipo específico desejado, utilizando para isto a função *_narrow* do próprio Orbix.

Na criação do dispositivo de saída, o cliente restringe apenas a localização (na Figura 5-4, *máquina_destino* = "trancoso.dca.fee.unicamp.br"). A referência recebida é convertida e o fluxo 2 fica estabelecido.

Ao criar a conexão virtual são passadas como restrições as localizações dos dispositivos fonte e destino e gera-se o fluxo de controle 3. Novamente, é necessário fazer uma conversão de tipo. Entretanto, o cliente não precisa conhecer o tipo da conexão (*NetworkConnection*). Basta conhecer o tipo genérico *VirtualConnection*.

Na ausência do objeto grupo, o cliente torna-se o responsável por adquirir os recursos e disparar o *stream* de cada membro. Ao obter as referências para os objetos *stream* de cada recurso, o cliente estabelece os fluxos de controle 4, 5 e 6. Através deles o cliente dispara a execução dos recursos, criando o fluxo de dados. Após um determinado período, a aplicação suprime a execução e libera os recursos, tanto físicos quanto lógicos.

Todas as funções que iniciam pelo caractere “_” (ex: *_bind*, *_narrow*, *_release*) são funções do Orbix.

```
void main(void) {
    MSS mss; // biblioteca do MSS
    EnhancedPREMOObjectRef obj;
    SequenceString capab(2);
    StreamRef file_stream, xwin_stream, vc_stream;
    VirtualConnectionRef vc;

    // INICIANDO APLICACAO MSS
    mss.init();

    // CRIANDO OBJETO TEXTFILEDEVICE
    //restricao InternetLocation

    capab[0] = new char[strlen("maresias.dca.fee.unicamp.br")+ 1];
    strcpy(capab[0], "maresias.dca.fee.unicamp.br");

    //restricao FileName
    capab[1]= new char[strlen("/proj/multimedia/ima/ORBTp.idl")+ 1];
    strcpy(capab[1], "/proj/multimedia/ima/ORBTp.idl");
    capab._length = 2;
    obj = mss.new_object( "TextFileDevice", capab);
    TextFileDeviceRef file = TextFileDevice::_narrow( obj );
    delete[] capab[0];
    delete[] capab[1];

    // CRIACAO DO OBJETO XWINDOWDEVICE
    //restricao InternetLocation

    capab[0] = new char[strlen("trancoso.dca.fee.unicamp.br") + 1];
    strcpy (capab[0], "trancoso.dca.fee.unicamp.br");
    capab._length = 1;
    obj = mss.new_object( "XWindowDevice" , capab);
    XWindowDeviceRef xwin = XWindowDevice::_narrow( obj );
    delete[] capab[0];

    // CRIACAO DO OBJETO VIRTUALCONNECTION
    //restricao SourceInternetLocation

    capab[0] = new char[strlen("maresias.dca.fee.unicamp.br") + 1];
    strcpy(capab[0], "maresias.dca.fee.unicamp.br");

    //restricao TargetInternetLocation
    capab[1] = new char[strlen("trancoso.dca.fee.unicamp.br") + 1];
    strcpy (capab[1], "trancoso.dca.fee.unicamp.br");
    capab._length = 2;
    obj = mss.new_object( "VirtualConnection" , capab);
    vc = VirtualConnection::_narrow( obj);
    delete[] capab[0];
}
```

```
delete[] capab[1];

// CONECTANTO OS DISPOSITIVOS
vc->connect(file, TextFileDevice::OutputPortC,
            xwin, XWindowDevice::InputPortC);

// FAZENDO AQUISICAO DE RECURSOS
file->acquireResource();
xwin->acquireResource();
vc->acquireResource();

// OBTENDO OS STREAMS PARA INICIAR A EXECUCAO
file_stream = file->resource_stream();
xwin_stream = xwin->resource_stream();
vc_stream = vc->resource_stream();

// DISPARANDO A EXECUCAO
file_stream->play();
vc_stream->play();
xwin_stream->play();
sleep(20);
file_stream->stop();
xwin_stream->stop();
vc_stream->stop();

// FIM DA APLICACAO. DESCONECTANDO SERVIDORES
file_stream->_release();
xwin_stream->_release();
vc_stream->_release();
file->releaseResource(); // liberando o recurso fisico
file->_release();       // liberando o recurso lógico(objeto Orbix)
xwin->releaseResource();
xwin->_release();
vc->releaseResource();
vc->_release();
}
```

A avaliação sobre a utilização do MSS e do CORBA será dada no próximo capítulo. Entretanto, uma característica que deve ser observada neste exemplo, é a linearidade no uso das interfaces e a facilidade de desenvolvimento oferecida pelo *framework*. Todos os objetos são criados e controlados de maneira equivalente, sem que o cliente se preocupe com detalhes do dispositivo de *hardware* e da mídia que ele processa.

6. Conclusão

6.1. Introdução

Neste capítulo é conduzida uma avaliação de diversos aspectos do trabalho realizado, tanto da proposta de implementação apresentada no Capítulo 4, quanto do *framework* MSS e da utilização de um ambiente CORBA. São expostas também as dificuldades encontradas nas diversas fases do desenvolvimento, desde a interpretação da especificação do MSS até a confecção do protótipo. São apontadas as contribuições geradas por este estudo e, finalizando, são sugeridas propostas para trabalhos futuros.

6.2. Avaliação

A utilização de uma arquitetura de serviços para suporte ao desenvolvimento de aplicações multimídia é fundamental, não apenas por prover abstrações e facilidades, mas também por oferecer um modelo de desenvolvimento. O *framework* MSS é um modelo poderoso, capaz de oferecer o suporte requerido, destacando-se que:

- os elementos de sua arquitetura provêm abstrações com respeito à plataforma, recursos, distribuição e características de mídia, permitindo aos programadores dedicarem maior atenção às aplicações multimídia do que à infra-estrutura do sistema;
- suas interfaces simplificam a programação (como pode ser visto no exemplo da seção 5.4) e oferecem linearidade no desenvolvimento;
- embora não tenham sido enfocados neste trabalho questões de tempo e sincronização, o MSS possui um modelo bastante elaborado para tratamento deste tema. O MSS abstrai tempo e sincronização através dos objetos *Stream* e de toda hierarquia definida pelos componentes fundamentais do PREMO;
- o tema qualidade de serviço não foi tratado neste trabalho, mas o MSS permite expressar requisitos de QoS. Todavia, o conjunto dos

parâmetros abordados na especificação é restrito. Eles levam em consideração somente requisitos de transporte (rede) e acredita-se que seja necessário a definição de parâmetros que permitam exprimir outros requisitos, como os de sistema, de dispositivos de mídia e, até mesmo, custo.

Contudo, o MSS apresenta alguma dificuldade por ser excessivamente genérico. Suas interfaces são definidas em um nível de abstração muito elevado, o que por vezes provoca confusão a respeito de suas funcionalidades. Além disso, muito das funcionalidades dos dispositivos virtuais são representadas através de objetos de configuração *Format*. Por exemplo, um determinado tipo de compressão é representado como um formato e não como um dispositivo específico. Como um dispositivo virtual pode ter mais de um formato, sua funcionalidade não fica completamente definida.

Talvez fosse mais apropriado criar tipos de dispositivos e de formatos mais especializados, ao exemplo do *framework* de Simon Gibbs [Gibbs94], oferecendo aos programadores de aplicações maior clareza sobre as funcionalidades do elemento a ser utilizado. Embora a especificação não limite a extensão das interfaces, permitindo especializações, elas não são padronizadas. Além disto, a especificação PREMO ainda possui alguns pontos em aberto, como é o caso dos objetos de configuração *MediaStreamProtocol*.

Considera-se que a escolha por utilizar o MSS como arquitetura de serviços para a plataforma Multiware tenha sido acertada, ainda que esta decisão tenha sido realizada há bastante tempo (por ocasião da definição do Projeto Temático), quando não havia outros *frameworks* disponíveis. O modelo tem evoluído e passa por processo de padronização pela ISO.

Ademais, seus conceitos estão sendo utilizados como fundamentos para outras definições, como é o caso do recente documento *Control and Management of A/V Streams* [Lucente97]. O referido documento é a submissão da Lucent Technologies Inc (Bell Labs) ao RFP (*Request for Proposal*) da OMG para criação de facilidades de controle e gerência de *streams* de mídia, no contexto da arquitetura CORBA/OMA. Nele são empregadas amplamente abstrações como dispositivo virtual, conexão virtual, porta e formato do MSS.

No que diz respeito ao CORBA, ele oferece grandes facilidades na interação entre os objetos distribuídos, provendo transparência de localização. Entretanto, CORBA não possui suporte para transmissão de *stream*, nem facilidades multi-protocolos. Por isto, o fluxo de dados deve ser transmitido fora do ORB para se ganhar eficiência e caso queira-se aproveitar benefícios de determinado recurso de transporte oferecido pelo sistema (ex.: uma interface para rede ATM);

Com relação ao ambiente de desenvolvimento empregado, pode-se dizer que ele é inadequado para a aplicação em questão. Aplicações multimídia exigem muito dos recursos do sistema, que a configuração utilizada não conseguiu suprir. Algumas das dificuldades encontradas são discutidas na próxima seção.

No que diz respeito ao modelo de implementação, reconhece-se a necessidade de redefinir algumas propostas feitas, especialmente para o adaptador de conexão virtual e para o *Stream*. No caso do adaptador de conexão, é necessário repensar sua estrutura de implementação de maneira a permitir o uso protocolos de transporte, ao invés do mecanismo de transmissão do ORB. Para o *Stream*, devem ser consideradas as interfaces bases que constituem sua hierarquia.

Do ponto de vista do protótipo, acredita-se que o fato de não terem sido implementados dispositivos virtuais com mídias contínuas, como áudio ou vídeo, não invalida a avaliação sobre a aplicabilidade do modelo MSS, uma vez que as interfaces são as mesmas independentemente da mídia que abstraem.

Numa avaliação global, acredita-se que o trabalho tenha cumprido seus objetivos. Ainda que os resultados não sejam vultuosos, os conhecimentos e a experiência adquiridos permitiram a formação de recursos para produção de versões futuras mais aperfeiçoadas.

6.3. Dificuldades Encontradas

Uma das principais dificuldades enfrentadas na realização do trabalho foi a de interpretação da especificação MSS. Isto é consequência de diversos fatores:

- alto grau de generalidade e de abstração das definições, uma vez que a especificação busca estabelecer um modelo geral e padrão;
- alterações de versões ocorridas na evolução do MSS, desde a proposta da IMA (*Interactive Multimedia Association*) até o atual estágio de padronização do PREMO pela ISO/IEC;
- indefinições, ambigüidades ou falta de clareza em alguns pontos, devido ao fato do PREMO não ser ainda um *International Standard*, e nem mesmo ter atingido o estágio final de padronização.

Do ponto de vista de implementação, vários obstáculos foram impostos pelo ambiente de desenvolvimento, dentre os quais podem ser destacados:

- ausência de *multi-thread*, o que comprometeu a eficácia e até mesmo o uso de estratégias mais evidentes;
- documentação pouco detalhada do Orbix, dificultando o processo de desenvolvimento das interfaces, especialmente no que diz respeito ao uso da interface dinâmica;
- *bug* do Orbix que não permitia o uso de chamadas do sistema operacional;
- falta de recursos nas estações IBM, como placas para captura e apresentação de dados multimídia assim como APIs para utilização de

tais recursos, tornando inviável a criação de dispositivos virtuais aprimorados;

Outro fator que gerou dificuldades foi a defasagem entre os membros do grupo responsável pela implementação do MSS como um todo. Interfaces já previstas na especificação não foram implementadas e interfaces internas que necessitavam de definição sequer foram determinadas. Isto comprometeu a criação de uma estrutura mais completa e elaborada de implementação.

6.4. Contribuições

Este trabalho contribui como passo inicial para concretização dos serviços de suporte ao desenvolvimento de aplicações multimídia, propostos pela plataforma Multiware, no âmbito do Projeto Temático em andamento na UNICAMP. O estudo realizado permitiu avaliar pontos fracos e vantagens da abordagem proposta, além de sugerir novas opções.

Finalmente, do ponto de vista pessoal, o trabalho contribuiu muito para a minha formação, gerando e consolidando conhecimentos, criando experiências e perspectivas para trabalhos futuros.

6.5. Trabalhos Futuros

Como consequência dos resultados obtidos neste estudo e da diversidade dos conhecimentos envolvidos, a perspectiva de trabalhos futuros é bastante ampla, a qual inclui:

- refazer algumas definições do modelo de implementação e acrescentar classes bases do MSS não consideradas no protótipo inicial;
- migrar o protótipo para um ambiente mais adequado. Atualmente, uma nova plataforma de desenvolvimento encontra-se disponível, a qual possui um ambiente distribuído *multi-thread* (também baseado em CORBA) e recursos para manipulação de multimídia, como dispositivos físicos para captura e exibição e APIs de programação que permitem a criação de dispositivos virtuais mais aprimorados;
- avaliar a implementação do modelo de serviços MSS em outra plataforma, por exemplo, em ambiente Java. Para isto, deve ser utilizado o JDK (*Java Development Kit*), uma vez que ele provê mecanismo de comunicação entre objetos, permite programação *multi-thread*, possui classes que facilitam a criação de interfaces gráficas [Java96] e oferece suporte para manipulação de áudio e vídeo através dos *Java Media Players* [Java97].

Bibliografia

- [Araujo96a] ARAÚJO, D. E. **Serviços de Gerenciamento ODP Utilizando a Arquitetura CORBA**. Campinas: FEEC, UNICAMP, 1996. Tese (Mestrado) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 1996. p. 21-28.
- [Araujo96b] ARAÚJO, D. E., PRADO, R. C. M. do, CARDOZO, E. **Implementação do Ambiente de Processamento Distribuído (DPE) da Arquitetura TINA-C Sobre CORBA**. Campinas: FEEC, UNICAMP, 1996. Relatório Interno. Faculdade de Engenharia Elétrica e de Computação, Universidade de Campinas, 1996. 14 p.
- [Ben-Natan95] BEN-NATAN, R. **CORBA: A Guide to the Common Object Request Broker Architecture**. McGraw-Hill, 1995. Cap. 1-3. p. 1-79.
- [Betz95] BETZ, M., **OMG's CORBA**. Dr. Dobb's Special Report, Winter 1994-95. p. 8-12.
- [Blair93] BLAIR, G. S., COULSON, G., DAVIES, N. **System Suport For Multimedia Applications: An Assessment Of The State Of The ART**. <ftp://ftp.comp.lancs.ac.uk/pub/mpg/MPG-93-29.ps.Z>
- [Blum96] BLUM, C., MOLVA, R. **A Software Plataform for Distribuited Multimedia Applications**. In: PROCEEDINGS OF INTERNATIONAL WORKSHOP ON MULTIMEDIA SOFTWARE DEVELOPMENT, March 1996, Berlin, p. 10-21.
- [CORBA93] **OMG The Commom Object Request Broker: Architecture and Specification**. Revision 1.2, December, 1993.

- [Gibbs94] GIBBS, S. J., TSICHRITZIS, D. C. **Multimedia Programing - Objects, Environments and Frameworks**. Addison-Wesley, 1994, 323 p.
- [Herman96] HERMAN, I., REYNOLDS, G. J., LOO, J.v. **Premo: An Emerging Standard for Multimedia Presentation - Part II: Specification and Applications** IEEE Multimedia, Winter 1996, v. 3, n. 4, p.72-75.
- [IMA94] INTERACTIVE MULTIMEDIA ASSOCIATION. **Multimedia Systems Services** version 1.2, 1994.
- [Java96] SUN MICROSYSTEMS INC. **JDK 1.1.1 New Featrure Summary**.<http://www.javasoft.com/products/jdk/1.1/docs/relnotes/features.html>.
- [Java97] SUN MICROSYSTEMS INC, SILICON GRAPHICS INC, INTEL CORPORATION. **Java Media Players**. <http://www.javasoft.com/products/javaedia/mediaplayer/playerguide/index.html>.
- [Lippman91] LIPPMAN, S. B. **C++ Primer**. 2nd ed, Addison Wesley, 1991. 614 p.
- [LOV/OMT94] VERILOG. **LOV/OMT Editor - Reference Manual**. version 1.1, 1994.
- [Lucent97] LUCENT TECHNOLOGIES INC. **Control and Management of A/V Streams**. Version 1.0, feb. 1997. <http://www.omg.org/docs/telecom/97-02-03.ps>
- [Madeira96] MADEIRA, E. R. M. **Software Architecture for Multimedia Communication and Management**. Campinas: IC, UNICAMP, 1996. Relatório Interno. Instituto de Computação, Universidade de Campinas, 1996. 41 p.
- [Mühlhäuser96] MÜHLHÄUSER, M. **Issues in Multimedia Software Development**. In: PROCEEDINGS OF INTERNATIONAL WORKSHOP ON MULTIMEDIA SOFTWARE DEVELOPMENT, March 1996, Berlin, p. 2-7.
- [ODP95a] ISO/IEC 10746; **ODP Reference Model Part 1, Overview**. June 1995.
- [ODP95b] ISO/IEC 10746; **ODP Reference Model Part 2, Foundations**. June 1995
- [ODP95c] ISO/IEC 10746; **ODP Reference Model Part 3, Architecture**. June 1995.

- [ODP95d] ISO/IEC 10746; **ODP Reference Model Part 4, Architectural Semantics**. June 1995.
- [Orbix95a] IONA Technologies Ltd. **Orbix distributed object technology, Programmer's Guide**. Release 1.3.1, February 1995.
- [Orbix95b] IONA Technologies Ltd. **Orbix distributed object technology, Advanced Programmer's Guide**. Release 1.3.1, February 1995.
- [Orfali94] ORFALI, R., HARKEY, D. **Client/Server survival guide with OS/2**. Van Nostrand Reinhold, 1994. Cap. 33: Object Request Brokers. p. 689-722. Cap. 34: Distributed Object Services. p. 723-744.
- [PREMO96a] ISO/IEC 14478-1; **Presentation Environment for Multimedia Presentation (PREMO) - Part 1: Fundamentals of PREMO**. 1996. 42 p.
<ftp://ftp.cwi.nl/pub/premo/PremoDocument/Part1/Part1.ps.gz>
- [PREMO96b] ISO/IEC 14478-2; **Presentation Environment for Multimedia Presentation (PREMO) - Part 2: Foundation Component**. 1996. 69 p.
<ftp://ftp.cwi.nl/pub/premo/PremoDocument/Part2/Part2.ps.gz>
- [PREMO96c] ISO/IEC 14478-3; **Presentation Environment for Multimedia Presentation (PREMO) - Part 3: Multimedia System Service Component**. 1996. 98 p.
<ftp://ftp.cwi.nl/pub/premo/PremoDocument/Part3/Part3.ps.gz>
- [Pyarali96] PYARALI, I., HARRISON, T., SCHMIDT, D. C. **Design and Performance of an Object-Oriented Framework for High-Speed Electronic Medical Imaging**. In: PROCEEDINGS OF OBJECT-ORIENTED TECHNOLOGIES AND SYSTEMS, USENIX, June 1996, Toronto.
<http://www.cs.wustl.edu/~schmidt/corba-research.html>.
- [Rumbaugh91] RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., LORENSEN W. **Object-Oriented Modeling and Design**. New York: Prentice Hall International, 1991. 500 p.
- [Schmidh95] SCHMIDT, D. C., HARRISON, T., AL-SHAER, E. **Object-Oriented Components for High-Speed Network Programming**. In: PROCEEDINGS OF FIRST CONFERENCE ON OBJECT-ORIENTED TECHNOLOGIES, USENIX, June 1995, Monterey.
<http://www.cs.wustl.edu/~schmidt/corba-research.html>.

- [Williams91] WILLIAMS, N., BLAIR, G. S., DAVIES, N. **Distributed Multimedia Computing: An Assessment Of The State Of The ART.** <ftp://ftp.com.lancs.ac.uk/pub/mpg/MPG-91-41.ps.Z>
- [Williams92] WILLIAMS, N., BLAIR, G. S. **Distributed Multimedia Applications Study.** <ftp://ftp.com.lancs.ac.uk/pub/mpg/MPG-92-11.ps.Z>

Anexo A : Esquema da *VirtualConnection*

<i>VirtualConnection</i> _{abstract}	
<i>VirtualResource</i> redef (<i>acquireResource</i>)	
<i>connect</i>	
<i>deviceMaster</i> _{in} : <i>RefVirtualDevice</i> <i>portMaster</i> _{in} : <i>Port</i> <i>deviceSlave</i> _{in} : <i>RefVirtualDevice</i> <i>portSlave</i> _{in} : <i>Port</i> <i>exceptions</i> : { <i>ConfigurationMismatch</i> , <i>PortMismatch</i> }	
A operação conecta duas portas de dispositivo virtual. A negociação do detalhes da porta inicia com a porta especificada por <i>deviceMaster</i> _{in} e <i>deviceSlave</i> _{in} . A porta de entrada e de saída é determinada pelo tipo da porta conectada.	
Exceções retornadas:	
<i>ConfigurationMismatch</i>	As configurações não podem ser conciliadas. O dado da exceção lista os objetos de configuração conflitantes. Contém uma sequência de tuplas, cada uma sendo uma par de <i>ConfInfo</i> (i.e. nome semântico e tipo do objeto). O primeiro elemento da tupla refere-se ao objeto de configuração <i>master</i> , e o segundo ao <i>slave</i> .
<i>PortMismatch</i>	Ambas as portas são de entrada ou de saída.
<i>disconnect</i>	
As conexões são encerradas. A operação realiza um <i>release</i> implícito.	
Exceções retornadas: Nenhuma.	
<i>getEndpointInfoList</i>	
<i>info</i> _{out} : seq (<i>RefVirtualDevice</i> × <i>Port</i> × <i>Boolean</i>)	
Uma informação é retornada sobre todas as portas conectadas: referência para o dispositivo virtual, identificação de sua porta e um <i>flag</i> para identificar se a porta é <i>master</i> (valor <i>TRUE</i>) ou não (valor <i>FALSE</i>)	
Exceções retornadas: Nenhuma.	
<i>VirtualConnection</i>	

Propriedades definidas:

Chave	Tipo do Valor	R.O ou R/W	Descrição
<i>TransportTypeK</i>	seq <i>ObjetcType</i>	R/W	Objeto Protocolo usado pela instância do objeto

Capacitações definidas:

Chave	Tipo do Valor	Valores
<i>TransportTypeCK</i>	seq <i>String</i>	<TCPTranport, UDPTranport, RTPTranport, DMATransport, NETBIOSTransport, ATMTransport>

Tipo *VirtualConnectionMulticast*

VirtualConnectionMulticast _{abstract}	
VirtualConnection	
attach	
<i>device_{in}</i> : RefVirtualDevice	
<i>port_{in}</i> : Port	
<i>exceptions</i> : { ConfigurationMismatch, PortMismatch }	
A operação adiciona uma porta à conexão. Esta operação pode ser usada uma vez que a conexão inicial foi estabelecida usando <i>VirtualConnection.connect</i> .	
Exceções retornadas:	
<i>ConfigurationMismatch</i>	As configurações não podem ser conciliadas. O dado da exceção lista os objetos de configuração conflitantes. Contém uma sequência de tuplas, cada uma sendo uma par de <i>ConfInfo</i> (i.e. nome semântico e tipo do objeto). O primeiro elemento da tupla refere-se ao objeto de configuração <i>master</i> , e o segundo ao <i>slave</i> .
<i>PortMismatch</i>	A nova porta é de um tipo incompatível (deveria ser de entrada se o <i>master</i> é de saída, ou de saída se o <i>master</i> é de entrada).
detach	
<i>device_{in}</i> : RefVirtualDevice	
<i>port_{in}</i> : Port	
<i>exceptions</i> : { PortMismatch }	
A operação retira uma porta conexão. Retirar a porta fonte derruba a conexão	
Exceções retornadas:	
<i>PortMismatch</i>	A porta não estava ligada à conexão.
VirtualConnection	