- Universidade Estadual de Campinas -

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E
AUTOMAÇÃO INDUSTRIAL

Caracterização de um Serviço de Gerência Distribuído para Objetos Multimídia Persistentes

Autor: **André Luís Vasconcelos Coelho**

Orientador: **Prof. Dr. Ivan Luiz Marques Ricarte**

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica.

Dezembro 1998

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

C65c

Coelho, André Luís Vasconcelos
Caracterização de um serviço de gerência distribuído para objetos multimídia persistentes. / André Luís Vasconcelos Coelho.--Campinas, SP: [s.n.], 1998.

Orientador: Ivan Luiz Marques Ricarte
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1.Processamento eletrônico de dados – Processamento distribuído. 2.Sistemas de recuperação da informação. 3.Banco de dados orientado a objetos. 4.Programação orientada a objetos I.Ricarte, Ivan Luiz Marques. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

O meu avô, certa vez, antes de vir a falecer, ainda me recordo, confidenciou-me uma simples passagem anedótica, a qual me arrisco transcrever, em essência, aqui:

Estavam o avô e o pequeno neto a caminhar, como de costumeiro o faziam. O segundo, muito atinado a tudo que o envolvia, sempre indagava a quem quer que estivesse a seu alcance acerca daquilo que não compreendia. Desta vez, ao observar como o gado de seu curral se amontoava em pouco espaço e como os diminutos pássaros se serviam de muito do que, à primeira vista, não precisavam, passou a inquirir, em bom tom, ao progenitor sobre como Deus parecia ter errado nesse detalhe tão aparente. O velho senhor, tomado de súbito por essa interpelação, preferiu, ou melhor, se conteve a emitir um vago "porque" como resposta a esse fato do qual, ao longo dos seus idos anos de vida, nunca havia se apercebido. Não mais que de repente, como um subterfúgio divino, ainda que, a meu ver, natural, uma dessas avoantes se pôs em vias de excreção, atingindo o produto do ato consumado a fronte do neto. Foi a deixa encontrada para a justificação de convencimento por parte do avô, o qual, em situação de contemplação, sabiamente, emendou: "Ainda bem que assim o é, pois querias tu, pequeno, que em vez de uma miúda rola estivesse por aí um grande touro a defecar-te do céu?".

É com essa mescla de humor e de observação/inquirição dos fatos que tento pautar a minha vida.

Dedico esta obra a meu avô, José Egydio Coelho, pelo exemplo de homem e pela imagem de artesão que se afixou em minha memória de menino. E, como extensão, à minha família, pelo ambiente de afeto, conforto, humildade, compreensão e, sobretudo, amor (pela acepção do termo), algo próprio de um povo a quem se convencionou tratar por nordestino. À minha família eu devo o que sou, os meus valores, a minha perseverança e a minha esperança.

Apólogo

CENA (mata):

É um belo e ensolarado dia na floresta e um pequeno coelho, com pose de intelectual, se encontra sentado fora de sua toca, dactilografando à sua máquina de escrever pessoal (ou seria um computador portátil?). À sua frente, perambulando na sua caminhada matinal, se apresenta uma lépida e vivaz raposa, quiçá a mais perspicaz do seu bando.

RAPOSA:
— Em que você está trabalhando?
COELHO:
— Em minha Tese.
RAPOSA:
— Humm.... Do que se trata?
COELHO:
— Ah, estou discorrendo sobre como coelhos se alimentam de raposas...

(**PAUSA INCRÉDULA!**)

RAPOSA:
— É ridículo!!! Qualquer tolo sabe que coelhos não comem raposas.
COELHO:
— Claro que o fazem, e eu posso provar. Venha comigo.

Ambos desaparecem rumo à toca. Após uns poucos minutos, volta o coelho, sozinho, à sua atividade de escritor e recomeça a trabalhar junto ao seu equipamento. Daí em breve, entra em cartaz um lobo, em trajes de gala (beca), o qual, mesmo muito açodado, se detém ao ar de compenetrado em que parece se deixar absorver o coelho.

LOBO:
— Sobre o que você tanto escreve?
COELHO (absorto):
— Eu estou elaborando um tratado sobre como coelhos se nutrem de lobos...

(**GARGALHADA DE ESCÁRNIO!**)

LOBO:
— Você não espera, sinceramente, que tal bobagem seja publicada..., ou espera?
COELHO (muito calmo):
— Sem problemas. O senhor quer saber por quê?

Os dois sujeitos, por conseguinte, adentram a toca; e, novamente, o coelho retorna, desacompanhado, ao seu passatempo de criador.

CENA (toca):

Em um canto do recinto, há uma pilha de ossos de raposa. Em um outro, uma ruma de fragmentos cadavéricos de lobo. Percorrendo os olhos ao longo da sala, pode-se aperceber, em seu deleite quase extático, um encorpado leão, palitando os dentes, a arrotar.

FIM

MORAL:

Parece não ser muito relevante o que você escolheu como assunto de tese.
Parece não ser muito importante o que você escolheu como dados para a prova.
Pois, realmente, o que interessa é quem você escolheu como seu orientador.

Agradeço, Ivan, por sua conduta, por seus preceitos, por sua amizade.

Gratidão

Reservo-me, aqui, a expressar o meu agradecimento àqueles que, direta ou indiretamente, em muito me auxiliaram na conclusão deste trabalho.

Aos meus pais, Vera Lúcia e José Pojucan, meus fãs incondicionais: antes de tudo, por me gerarem, criarem e ensinarem a ser o que sou; no mais, pela abdicação constante, pelos conselhos, pelo suporte aos meus estudos e pelo referencial de vida (legado) o qual pretendo seguir.

Às minhas irmãs, Ana Paula (Jaap) e Aline, por compartilharem comigo grande parte dos meus momentos de infância e juventude e pela responsabilidade a que me cabe, como único filho homem que sou, de, mesmo tão distante, protegê-las.

Ao meu orientador, Ivan Luiz Marques Ricarte, pela paciência e confiança devotadas e, mais ainda, pelo caráter e postura nos seus ensinamentos de amigo.

A todos os colegas, professores e funcionários do LCA, pelo convívio, pelos debates e pelo suporte às minhas solicitações. Dentre eles, permito-me destacar: Christian Medeiros, Luiz Affonso Guedes, Naur Janzantti, Cláudia Tambascia e Raquel Schulze.

Ao amigo e companheiro de jornada Alexandre de Souza Pinto, pelas discussões motivadoras, pela audiência enquanto filosofei sobre o tema, e, principalmente, pelo discernimento raro de se encontrar num atleticano são-paulino (ou será o contrário?).

Aos colegas da Engenharia de Computação 92, pelo ambiente de estudos e pela convivência durante os cinco primeiros anos de estada nesta universidade, o que acarretou na minha decisão (inconseqüência, diriam alguns) de permanecer por mais esse tempo.

À comunidade cearense em Campinas, liderada pelo meu amigo Maurício Morais de Lima Júnior (a quem devo pelas aprazíveis conversas e discussões acerca do mundo e de tudo que nos é apresentado) e da qual fazem parte Lucas Marques Morais de Lima, Jorge André Girão de Albuquerque e Rodrigo Gribel Lacerda — este último, adotado pelo grupo.

A George de Oliveira Marques, um amigo a quem tenho um grande apreço, pela sinceridade, pelo incentivo à leitura e à boa música, pela conservação dos valores e, sobretudo, pela índole. Alguém que, mesmo reticente aos avanços por quais passamos, se encontra, pelo lado humano, bem à frente do nosso tempo.

Ao CNPq, pelo apoio financeiro.

A Darwin, por ter elucidado, pelo menos em parte, a natureza a que pertenço, auxiliando-me na compreensão da condição de ser, simplesmente, humano.

Resumo

Uma das tarefas mais desafiadoras e significativas dentro da Multimídia é a de se lidar, adequadamente, com as restrições temporais inerentes a grandes coleções de dados encapsulados em entidades de modelagem conhecidas como objetos multimídia. Essas unidades de informação suprem as demandas de abstração requeridas pelas aplicações, seguindo papéis lógicos unicamente relacionados a um particular cenário de apresentação.

Repositórios dedicados a mídia são componentes de *software* configurados para o atendimento otimizado aos requisitos temporais e espaciais próprios de objetos multimídia codificados sob formatos particulares. São responsáveis por tarefas de mais baixo nível voltadas à manipulação (armazenagem e recuperação) de fluxos de mídia, integrando um agregado estruturado conhecido como Sistema de Gerência de Banco de Dados Multimídia. A dispersão massiva desses repositórios em um ambiente distribuído passa a prover novos benefícios de desempenho concernentes a algumas de suas propriedades funcionais, tais como escalabilidade, confiabilidade, disponibilidade e compartilhamento de dados.

Neste trabalho, as descrições funcional e arquitetural do SGPOM são anunciadas. A motivação por trás dessa abordagem é a de se prover um serviço distribuído bem dimensionado de suporte ao acesso e à gerência de objetos multimídia persistentes, incorporando e, por conseguinte, localizando os repositórios mais adequados ao tratamento de suas peculiaridades. Desse modo, não se propõe uma nova solução baseada em extensões feitas a uma linguagem de programação específica, mas sim uma ferramenta de sistema devotada ao atendimento concorrente de requisições de E/S relativas a esses objetos, o que é feito mediante o contrato estabelecido por sua API pública.

Um conjunto de ponderações de projeto e de implementação se constitui na principal contribuição desta tese, introduzindo um serviço influenciado pela filosofia do padrão RM-ODP e concordante com a arquitetura CORBA. O resultado foi a criação de uma hierarquia enriquecida de componentes distribuídos baseada naquela proposta para o Serviço de Objetos Persistentes — CORBAservice do OMG —, estendendo as suas funcionalidades para o devido atendimento às limitações impostas pela classe de objetos multimídia. Uma análise qualitativa de levantamentos de medidas de desempenho temporal para um protótipo do serviço corrobora a adequação da proposta.

Palavras-chave: Repositórios Multimídia, Persistência, CORBA, POS.

Abstract

One of the most compelling and challenging tasks inside the Multimedia field is that of suitably tackling the temporal restrictions commonly associated to huge collections of bulky data encapsulated in model entities known as multimedia objects. These entities undertake the lacked abstractions required by their owner applications, following logical roles uniquely related to a particular presentation scenery.

Media-dedicated repositories are specialized datastore tools configured with multimedia objects' temporal and spatial requirements in mind. They may be launched as low-level elements responsible for multimedia datastream storage and recovery, taking part into a whole structured assemblage conceived as a Multimedia Database Management System. In order to adhere to the multimedia applications' increasing demands, the massive distribution and placement of these repositories can provide new performance benefits, improving some of their functional properties such as scalability, reliability, availability, and data sharing.

In this work, the functional and architectural description of SGPOM is reported. The motivation behind this new approach is to provide a well-dimensioned distributed service to give support to the access and management of persistent multimedia objects, incorporating and then locating the most suitable repositories to comply with their particularities. Hence, SGPOM does not address any new solution based on the extensions of a specific programming language's capability. Conversely, it is a detached system-environment tool devoted to cater for concurrent I/O requests via its public API.

A set of project and implementation guidelines and tradeoffs encompasses the main contributions of this thesis. It provides for the deployment of a CORBA-compliant service influenced by the philosophy of the RM-ODP and which constitutes an enhancement of the OMG Persistent Object Service hierarchical arrangement of components, overcoming its weaknesses in dealing with multimedia constraints. Qualitative results obtained through measurement-sessions over a developed prototype corroborates the fitness of the whole proposal.

Keywords: Multimedia Repositories, Persistence, CORBA, POS.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário de Acrônimos, Siglas e Abreviaturas

ANSA	*Advanced Networked Systems Architecture*
API	*Application Programming Interface*
AVI	*Audio Video Interrelated*
BD	Base de Dados
BDMM	Banco de Dados Multimídia
BLOB	*Binary Large Object*
BMP	*Bitmap*
BOA	*Basic Object Adapter*
bps	*bits per second*
CAD	*Computer-Aided Design*
CGI	*Common Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
CPU	*Central Processing Unit*
DCE	*Distributed Computing Environment*
DCOM	*Distributed Component Object Model*
DII	*Dynamic Invocation Interface*
DPE	*Distributed Processing Environment*
DSI	*Dynamic Skeleton Interface*
E/S	Entrada/Saída
FS	*File System*
GIF	*Graphic Interchange Format*
GIOP	*General Inter-ORB Protocol*
HTTP	*Hypertext Transfer Protocol*
IDL	*Interface Definition Language*
IETF	*Internet Engineering Task Force*
IIOP	*Internet Inter-ORB Protocol*
IMA	*Interactive Multimedia Association*
IOR	*Interoperable Object Reference*
IPC	*Inter-Process Communication*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunications Union*
JBIG	*Joint Bi-level Image Experts Group*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JMF	*Java Media Framework*
JMP	*Java Media Player*
JPEG	*Joint Photographic Experts Group*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
MIDI	*Musical Instrument Digital Interface*
MIME	*Multipurpose Internet Mailing Extensions*
MPEG	*Moving Picture Experts Group*
MSC	*Message Sequence Chart*
mseg	milissegundo
NFS	*Network File Services*

OA	*Object Adapter*
ODA	*Object Database Adapter*
ODMG	*Object Data Management Group*
ODP	*Open Distributed Processing*
OM	Objeto Multimídia
OMA	*Object Management Architecture*
OMFI	*Open Media Framework Interchange*
OMG	*Object Management Group*
OMT	*Object Modeling Technique*
OO	Orientação/Orientado a Objetos
ORB	*Object Request Broker*
PC	*Personal Computer*
PDF	*Portable Document Format*
PDS	*Persistent Data Service*
PID	*Persistent Identifier*
PIDL	*Pseudo-IDL*
PO	*Persistent Object*
POA	*Portable Object Adapter*
POM	*Persistent Object Manager*
POS	*Persistent Object Service*
PSE	*Persistent Storage Engine*
QoS	*Quality of Service*
RAID	*Redundant Array of Inexpensive Disks*
RAM	*Random Access Memory*
RFC	*Request For Comment*
RFP	*Request For Proposal*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
RTP	*Real-Time Transport Protocol*
SD	*Synchronized Data*
SGBD	Sistema de Gerência de Banco de Dados
SGBDOO	Sistema de Gerência de Banco de Dados Orientados a Objetos
SGBDMM	Sistema de Gerência de Banco de Dados Multimídia
SGBDR	Sistema de Gerência de Banco de Dados Relacionais
SGPOM	Serviço de Gerência da Persistência de Objetos Multimídia
SGPOM_DS	SGPOM - *Data Service*
SGPOM_PID	SGPOM - *Persistent Identifier*
SGPOM_PO	SGPOM - *Persistent Object*
SGPOM_OM	SGPOM - *Object Manager*
SO	Sistema Operacional
SQL	*Standard Query Language*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TINA	*Telecommunications Information Network Architecture*
URL	*Uniform Resource Locator*
VCR	*Video Cassete Recording*
VOD	*Video On Demand*
WWW	*World Wide Web*

Capítulo 1

Introdução

Avanços tecnológicos significativos vêm sendo obtidos em diversas áreas do conhecimento humano, trazendo como conseqüência a necessidade de se avaliar modos alternativos de se gerar, conduzir, distribuir e processar a informação. À medida que o computador se torna uma ferramenta imprescindível no suporte às crescentes demandas por grandes volumes de dados, verifica-se o seu uso em novas frentes antes não previstas.

Por trás desta revolução, esforços estão sendo concentrados em busca de novos modelos de representação da informação que municiem a sua onipresença. Sob essa premissa, novas classes de aplicações e sistemas computacionais justificam o investimento a ser feito no estudo da *Multimídia*[1]. Essa disciplina, a qual congrega pesquisas em ramos bastante distintos, vem ganhando impulso ao passo que suas contribuições e resultados se tornam mais freqüentes. Transformações alcançadas em domínios sociais chave, tais como educação, entretenimento e atividades médicas colaborativas, servem de exemplo.

Aplicações multimídia são construídas com o intuito de explorar e estimular aspectos de percepção multissensorial dos usuários, combinando informações de natureza e forma variadas, como texto, áudio, vídeo, imagens e animações. Dentro dessa expectativa, um mesmo dado pode ter distintas interpretações, conforme o seu caráter subjetivo e implicações culturais intrínsecas. Mais do que isso, a tendência recente é a de se prover meios de interação e agregação pelos quais a comunicação dos fatos possa ocorrer, seguindo uma lógica instintiva (natural). O objetivo é possibilitar que informações sob diferentes formatos sejam conduzidas e apresentadas, paralelamente, para um ou mais usuários.

---

[1] Ao longo deste documento, passa-se a usar uma notação diferenciada quando da introdução de um novo termo, cujo conceito se torna importante dentro do contexto. Da mesma forma, utiliza-se esse recurso para fins de realce ou para se assinalar alguma palavra escrita em outro idioma.

O que se espera é uma rápida evolução da tecnologia multimídia de tal forma a tornar factível a construção de sistemas e aplicações complexos, os quais garantam níveis de consumo desejáveis. Sendo assim, serviços de vídeo sob demanda interativos, trabalhos remotos baseados em cooperação, telepresença e sistemas de distribuição de mensagens multimídia virão a servir como substrato sobre o qual novos processos e novas atividades poderão ser construídos e modelados. Por ser uma área de recente interesse, a Multimídia se serve de esforços já amadurecidos realizados em outros domínios e é profundamente marcada pela forte tendência de integração e sinergia entre aspectos computacionais e de comunicação.

Dentro dessa ótica, quando se passa a caracterizar serviços multimídia, deve-se recorrer a padronizações existentes que promovam a *interoperabilidade* entre as aplicações. Devido à existência de discrepantes especificações de soluções proprietárias executando sobre plataformas heterogêneas, é gradualmente necessária uma filosofia que promova a interconexão dos sistemas de modo a assegurar que a distribuição dos dados seja mantida. Trabalhos na área de *Sistemas Distribuídos Abertos* estão sendo desenvolvidos, almejando-se garantir transparências e funções de suporte à criação de aplicações distribuídas complexas, entre elas as de caráter multimídia.

A manipulação de dados multimídia impõe vários requisitos referentes aos recursos de *software* e *hardware* a serem empregados como infra-estrutura de suporte. Progressos em redes de computadores mais rápidas, bem como em processadores e mecanismos de armazenagem de dados mais poderosos, se tornam vitais para que se consiga projetar serviços de informação multimídia de melhor desempenho e mais abrangentes. Além disso, novas abstrações computacionais e estruturas de dados devem ser propostas, levando-se em consideração a natureza dos dados multimídia. Nesse caso, atualmente, o *Paradigma de Orientação a Objetos* se mostra o mais indicado para esse tratamento.

Objetos encapsulam estruturas de dados e servem como unidades básicas de informação e conteúdo sobre as quais serviços e aplicações podem ser construídos. Usualmente, representam entidades do mundo real e reagem a eventos por meio de mecanismos de trocas de mensagens, alterando o seu comportamento em consonância com a lógica de máquina de estados. O poder de abstração embutido nesse conceito permite que a modelagem da aplicação se torne mais próxima da realidade, trazendo a reboque benefícios significativos às fases de análise do problema e de projeto e implementação da solução. *Objetos multimídia* abrangem essa definição e podem ser construídos sobre fluxos de dados multimídia, formando hierarquias de classes e tipos. Tais objetos podem compor estruturas de mais alto nível, como documentos multimídia, assumindo, por conseguinte, um contexto lógico inerente à apresentação.

Com relação aos requisitos de armazenagem, gerência e recuperação de objetos multimídia, surgiu a necessidade de se abordar questões envolvendo novos modelos de estruturação, sincronização, captura e organização dos dados, de tal modo a se atender aos níveis especificados de qualidade dos serviços oferecidos às aplicações. Nesse âmbito, Sistemas de Gerência de Banco de Dados (*SGBDs*) tradicionais, os quais constam de uma coleção controlada de repositórios e bases de dados, passaram a ser reavaliados e incrementados, a fim de lidarem com requisitos e restrições próprios dos dados multimídia.

Dentro desse cenário, propôs-se uma arquitetura de serviços multimídia, construídos em camadas, conhecida como *Plataforma Multiware* [Loyo93]. A motivação passa a ser a de reunir, em uma única plataforma de desenvolvimento, os recursos tecnológicos de sistemas distribuídos abertos necessários ao suporte a aplicações multimídia. Como extensão natural a essa infra-estrutura, foi concebido um modelo de Banco de Dados Multimídia (*BDMM*), sob a alcunha de *Banco de Dados Multiware* [Toba95] [Rica96], o qual especifica um protótipo de banco de dados distribuído voltado à manipulação adequada de objetos multimídia. A sua estratégia de projeto é a de explorar e incorporar, sempre que possível, recursos e padrões (comerciais ou não) já solidificados que possam auxiliar no suporte ao acesso[2] (armazenagem e recuperação) a dados multimídia. Mecanismos complementares devem ser desenvolvidos a fim de que o "acoplamento" entre as exigências e necessidades de aplicações multimídia e os benefícios trazidos com esses padrões possa ser o maior possível.

O presente trabalho tem seu escopo definido à luz da filosofia da proposta Multiware. O seu propósito é reportar a construção e implementação de uma camada de gerência responsável pelo controle no acesso a repositórios de objetos multimídia, os quais se encontram dispersos sob uma arquitetura distribuída.

1.1 Descrição do problema

O primeiro passo rumo ao projeto de sistemas de informação multimídia é prover um modo integrado e homogêneo de descrever, organizar e estruturar objetos de conteúdo multimídia, assim como representar as suas relações temporais e espaciais, em uma única entidade [Karm96]. Seguindo essa perspectiva, os sistemas computacionais estarão armazenando informação no molde de documentos que integram representações diretas do mundo real, sob a forma de texto, fotos, figuras e gravações de áudio e vídeo. Destarte, a unidade de manipulação em um sistema multimídia deixa de ser um simples registro ou tabela de uma base de dados convencional e passa a ser um *documento multimídia*, o qual integra as distintas representações da informação de maneira uniforme.

Quanto à eficácia na manipulação dessas unidades de informação, novas ferramentas serão necessárias para o suporte à sua representação e composição, bem como à sua apresentação ao usuário. Em face dos requisitos temporais e de armazenagem próprios da informação multimídia, novas técnicas de recuperação devem lidar com grandes volumes de dados e com aspectos de sincronização e interatividade. Os novos tipos de mídia introduzidos, como áudio, vídeo e animações, apresentam características peculiares e heterogêneas que devem ser tratadas conveniente e separadamente. Algumas dessas características se transformam em restrições a serem satisfeitas [Nara96]:

---

[2] Em todo o documento, usar-se-á a palavra *acesso* com o sentido de *ingresso*, *passagem*, *contato*, como fruto de um anglicismo introduzido pelo jargão computacional. Outros vocábulos (como *Externalização*), mesmo não seguindo convenção alguma, são introduzidos, haja vista a necessidade de se manter o sentido original do conceito.

- Objetos multimídia são complexos e, portanto, não podem ser completamente capturados (representados) via esquemas em um banco de dados.
- Objetos multimídia são de natureza audiovisual e, portanto, suscetíveis a múltiplas interpretações.
- Objetos multimídia são sensíveis a contextos.
- Objetos multimídia possuem estruturas de informação irregulares e de caráter nebuloso.

Com isso em mente, passou-se a reavaliar os sistemas de banco de dados disponíveis comercialmente com a intenção de classificá-los conforme a sua adequação ao tratamento de dados multimídia. Estudos recentes [Rako95] [Viei95] [Adje97] [Paza97] demonstram que os Sistemas de Gerência de Banco de Dados Orientados a Objetos (*SGBDOOs*) se mostram os mais apropriados à manipulação de dados complexos. Tais sistemas têm por objetivo suportar, eficientemente, as tarefas de armazenagem e recuperação desses dados (encapsulados sob a forma de objetos) como garantia da sua *persistência*, isto é, da sua existência além do período de execução das aplicações.

Segundo Ricarte [Rica96], alguns requisitos impostos por atividades multimídia sobre sistemas de banco de dados são melhor abordados por servidores de mídia dedicados, os quais, por sua vez, são projetados para assegurarem melhor desempenho no tratamento de particularidades inerentes a uma dada mídia sob um certo formato. Como exemplo, um servidor de vídeo pode ter associado a si algoritmos dedicados de compressão e se utilizar de estratégias de alocação de dados ao disco mais oportunas à recuperação dos mesmos em tempo-real. Disso, optou-se por incorporar ao Banco de Dados Multiware servidores de repositórios especializados, os quais ofereçam tarefas otimizadas de acesso e recuperação consoante parâmetros estabelecidos de *QoS* (Qualidade de Serviço), além de guardarem informações descritivas dos dados sob a forma de *metadados*. Tais servidores devem ser construídos segundo padrões estabelecidos de formato das mídias (como os já existentes MPEG, JPEG, JBIG, GIF, AVI [Cost98]), possibilitando a integração do Banco de Dados Multiware a outras plataformas. Ao passo que esses repositórios atendem aos requisitos de mídia simples, o tratamento dos aspectos estruturais e de composição de objetos (documentos) multimídia e hipermídia[3], bem como das propriedades a eles associadas, passa a ser de responsabilidade de um SGBDOO adequado. Tal SGBDOO deve ocupar-se com os *esquemas* evolutivos [Meye96] de objetos multimídia (via controle de versões), preservando as suas inter-relações e dependências. Um protótipo iniciado desse componente é descrito por Janzantti [Janz].

Com efeito, o futuro das aplicações multimídia está intimamente ligado à sua execução em ambientes distribuídos, já que sistemas centralizados têm sua aplicação e desempenho limitados, tanto pela capacidade de armazenagem como pelos requisitos exigidos de processamento [Bufo94]. A arquitetura a ser adotada para a construção de serviços multimídia deve levar em consideração novas abstrações de distribuição dos dados entre domínios de aplicações, com o intuito de se aliviar a sobrecarga entre os

[3] Hipermídia pode ser definida como a aplicação do conceito de hipertexto a documentos multimídia [Fluc95]. Isto é, a informação passa a ser estruturada e apresentada de forma não-seqüencial, a partir do uso de *links* lógicos entre fragmentos de dados relacionados.

componentes do sistema. Nesse sentido, uma questão que se manifesta é a de como integrar os serviços de um SGBDOO e de seus correspondentes repositórios àqueles oferecidos por uma plataforma distribuída.

A Figura 1.1 traz uma visão modificada da proposta apresentada por Tobar [Toba95], qual seja a de estruturação em camadas de um banco de dados orientado a objetos distribuídos para aplicações multimídia. A partir dela, pode-se depreender como um objeto multimídia seria armazenado em uma plataforma distribuída. Na camada mais próxima do nível físico, estariam localizados os servidores dedicados a cada tipo de mídia. Informações sobre a estrutura de objetos e documentos multimídia estariam armazenadas em um nível intermediário, constituído pelo SGBDOO. Finalmente, um terceiro plano suportaria a interface com aplicações multimídia, recebendo e entregando objetos e documentos multimídia. Sob essa perspectiva, pode-se constatar que a um *objeto de exibição* correspondem um único *objeto de informação* (conteúdo) e um ou mais *objetos de apresentação* (arranjos de seus atributos de exibição).

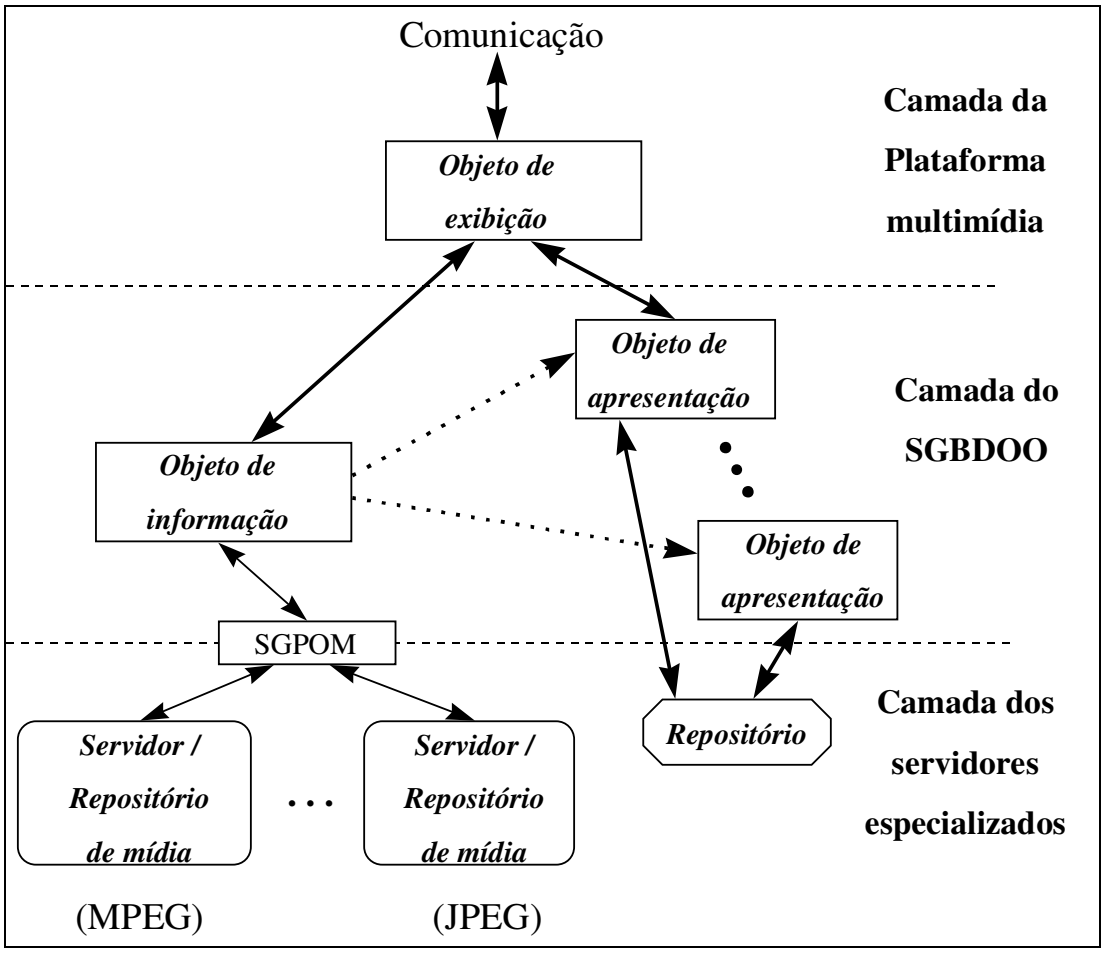

***Figura 1.1**: Estrutura em camadas de um banco de dados orientado a objetos distribuídos para aplicações multimídia.*

Particularmente, a distribuição em larga escala de repositórios multimídia apresenta alguns atrativos, dentre os quais, pode-se citar:

- **Compartilhamento dos dados via replicação**: o que possibilita que mais de uma aplicação multimídia ou partes dela possam estar acessando cópias de um mesmo dado. Mecanismos de consistência desses dados devem ser providos em separado.
- **Confiabilidade e disponibilidade aumentadas**: a primeira implica na probabilidade de um repositório estar funcionando num dado momento, enquanto a última se refere à possibilidade deste estar continuamente disponível durante um certo intervalo de tempo.
- **Maior autonomia e escalabilidade**: onde cada repositório é um componente distinto com características e funcionalidades independentes.

Há, porém, um agravante adicional na implementação de sistemas multimídia em ambientes distribuídos, qual seja o da possibilidade da rede de comunicação não poder dar vazão suficiente à transferência dos dados. O projeto de um banco de dados multimídia assim como o de um serviço de gerência de repositórios multimídia deve estar atento a certas limitações, que podem, nessa situação, ser impostas não apenas pela fonte dos dados (dispositivos de armazenagem secundários e terciários), mas também pelo seu destino (a rede, em primeira instância).

Corroborando essa perspectiva, tornou-se necessária a construção de uma camada de gerência, composta por objetos distribuídos, que servisse de interface aos diversos repositórios especializados disseminados por um ambiente distribuído. Essa camada se constitui em um serviço bem dimensionado, implementado como um *framework*[4] de objetos localizados a partir de suas interfaces públicas.

1.2 Proposta

Objetos persistentes podem ser definidos como elementos de dados que "continuam a existir" mesmo que as aplicações que os criaram venham a terminar a sua execução. De uma outra forma: quando se faz alusão à persistência, está-se referindo à propriedade de se guardar o *estado* de um objeto — o qual é, por sua vez, basicamente definido por seus atributos e pelos *links* ou associações com outros objetos — num repositório ou base de dados, de tal forma a possibilitar a sua recuperação *a posteriori*.

A solução tradicional para o problema da persistência de dados recai sobre a utilização de sistemas de arquivos, os quais armazenam as informações (conteúdo dos dados) em discos; esta é a solução comumente adotada em diversas implementações de aplicações multimídia. Contudo, um dos principais problemas no uso direto de arquivos para o suporte à persistência está no forte acoplamento existente entre as aplicações e os dados. De um outro modo: a incumbência do tratamento (manipulação) dos dados recai, diretamente, sobre a aplicação. Isso afeta sobremaneira o compartilhamento das informações (problema de concorrência), à proporção que compromete a garantia de que os requisitos temporais inerentes às aplicações multimídia sejam cumpridos.

[4] Estrutura.

Em ambientes distribuídos, o uso da persistência está centrado basicamente na simplificação de aplicações que tenham que processar grandes volumes de dados (as ferramentas e aplicações multimídia distribuídas são um bom exemplo). Nesse caso, em vez de se implementar diferentes estruturas para a organização e consistência desses dados (sob a forma de objetos), tanto em memória como em disco, pode-se lançar mão de um serviço especializado. Tal serviço torna-se responsável por guardar os estados dos objetos (conhecidos como *estados persistentes*), promovendo, ao longo do tempo, a sua consulta por diversas aplicações (conhecidas como *clientes* desse serviço), inclusive de uma forma concorrente.

Para a construção de uma camada de gerência de repositórios multimídia, objetivo desta tese, passou-se a avaliar, qualitativamente, estruturas de serviços distribuídos já existentes que tratassem da persistência dos dados sob a ótica da orientação a objetos. Nessa direção, um maior destaque é dado, neste documento, à arquitetura *CORBA* (*Common Object Request Broker Architecture*) [OMG95a] [Bake97], a qual vem ganhando amadurecimento e destaque no suporte robusto à construção de aplicações complexas sobre plataformas e ambientes heterogêneos. Tal arquitetura foi proposta a partir de um Modelo de Objetos conceitual e formal, conhecido como *Core Object Model* [Sole95b], e é continuamente incrementada e reavaliada pelo OMG — consórcio criado por empresas e usuários que visa promover a adoção de padrões para a gerência de objetos distribuídos. A premissa escolhida por essa entidade é a de adotar especificações de interface e protocolo que permitam a interoperação de aplicações criadas como ou por objetos distribuídos. Isso garante uma tecnologia firmada sobre anseios mais pragmáticos, quando comparada a outras anteriormente desenvolvidas.

Dentro dessa arquitetura, são projetados serviços básicos, conhecidos como *CORBAServices* [OMG95b], os quais funcionam como alicerce para a construção de aplicações ou outros serviços. Para os propósitos deste trabalho, um estudo da especificação proposta para o *Serviço de Objetos Persistentes* (o qual será, doravante, chamado de POS, advindo de *Persistent Object Service*) serviu de base para as decisões de projeto e de implementação do serviço de gerência no acesso a repositórios multimídia. O grande benefício trazido com essa abordagem está na sua estruturação, a qual é feita em camadas, e no seu caráter intrinsecamente distribuído.

Por se basear no POS, o serviço a ser reportado nesta dissertação recebeu o acrônimo SGPOM[5]. Grosso modo, tal serviço assume o papel de responsável pela manipulação e tratamento dos estados persistentes de objetos que compõem aplicações (ou documentos) multimídia. É localizado, dentro de um ambiente distribuído, via um conjunto de interfaces públicas, permitindo a uma ou mais aplicações recuperarem, segundo restrições temporais e de qualidade de serviço, os estados de seus componentes multimídia. Sua arquitetura é composta por uma coleção de objetos gerentes, integrando uma estrutura hierárquica e extensível, relacionada a um certo domínio ou classe de aplicações.

O SGPOM tem como meta oferecer as seguintes funcionalidades aos repositórios especializados a mídia:

---

[5] Serviço de Gerência da Persistência de Objetos Multimídia.

- **Interagir com diferentes tipos de repositórios**: desde aqueles confeccionados sob o modelo relacional até os construídos sobre sistemas de arquivos ou produtos de banco de dados orientados a objetos disponíveis comercialmente.
- **Atender a necessidades de armazenagem e recuperação de diferentes aplicações multimídia**: o que acarreta, necessariamente, na construção de um serviço genérico e aberto.
- **Transparência de acesso aos dados**: onde aplicações, clientes do serviço, não precisam saber como o mesmo funciona internamente, ou seja, como os dados são adequadamente manipulados e alocados a repositórios específicos.
- **Suporte à concorrência**: onde mais de uma aplicação pode e deve fazer uso dos recursos providos pelo serviço.
- **Transparência de localização dos repositórios**: escondendo das aplicações-cliente a política de distribuição dos repositórios pelo ambiente. Isso garante um balanceamento da carga de atendimento às requisições entre os repositórios disponíveis, promovendo a sua autonomia e escalabilidade (ver Seção 1.1).
- **Robustez e flexibilidade**: o sistema deve ser tolerante a falhas, atendendo ao maior número possível de requisições, mesmo que parte da sua estrutura não esteja funcionando adequadamente ou esteja corrompida. Além disso, deve ser extensível a ponto de suportar novas restrições que apareçam com o tempo.

1.3 Organização do documento

Este texto tem como objetivo reportar os passos seguidos quando do projeto e da implementação de um serviço distribuído de suporte a objetos multimídia persistentes. Porquanto, como fonte de embasamento para as decisões tomadas durante a sua especificação, foi realizado um estudo prospectivo do estado-da-arte, tornando autocontida, na medida do possível, esta dissertação.

O documento está dividido em sete capítulos. O primeiro deles traz o cenário onde se insere o projeto, a sua motivação, bem como a descrição e proposta de solução do problema. No Capítulo 2, é realizado um estudo contemplativo de dois padrões de Sistemas Distribuídos orientados a objetos, *RM-ODP* e CORBA, que serviram de base teórica para a construção do serviço. Nesse capítulo, são introduzidas as funcionalidades de alguns dos CORBAservices, os quais, direta ou indiretamente, influenciaram o projeto do SGPOM.

No terceiro capítulo, é feita uma revisão dos principais requisitos inerentes à classe de aplicações multimídia, os quais devem ser convenientemente abordados quando da construção de BDMMs. No ensejo, são discutidos, brevemente, dois padrões: o primeiro, de representação uniforme de formatos de mídias distintas (*MIME*), e o segundo, de descrição estrutural de objetos multimídia na forma de *containers* (*Bento*). Uma amostra de alguns dos esforços realizados na área de Sistemas Distribuídos Multimídia construídos sobre objetos conclui a avaliação.

O Capítulo 4 é reservado para uma discussão crítica do Serviço de Persistência de Objetos da CORBA (POS), o qual serviu de modelo para a proposta do SGPOM.

No início, são introduzidos os principais conceitos relativos à persistência de dados. Posteriormente, passa-se a investigar a arquitetura do POS e as funcionalidades e interações existentes entre seus componentes. Finalmente, serão destacados os "prós" e os "contras" dessa especificação, conforme as limitações impostas por objetos multimídia.

No decorrer dos três capítulos anteriores, são feitas algumas considerações sobre como a nova plataforma de programação *Java* — linguagem usada na implementação do serviço — se comporta como ferramenta de suporte a objetos multimídia, objetos distribuídos e objetos persistentes.

No quinto capítulo, o SGPOM é detalhado. Passa-se a descrevê-lo segundo as suas perspectivas funcional (contexto lógico dentro da proposta Banco de Dados Multiware) e arquitetural (seus componentes, o conjunto de interfaces e suas interações), ressaltando as principais decisões de projeto e de implementação tomadas durante a sua idealização.

Para validação da proposta desta exposição, o Capítulo 6 apresenta os resultados obtidos quando da implementação de um protótipo do SGPOM. Esses resultados são demonstrados sob a forma de gráficos e tabelas, apresentando medidas e parâmetros de desempenho que comprovam o comportamento esperado. Por outro lado, um conjunto de figuras ilustra o ambiente de execução (*workspace*) do protótipo, exibindo as interfaces gráficas usadas no controle e monitoração do serviço, bem como no suporte a apresentações multimídia.

O Capítulo 7 traz as conclusões sobre o trabalho, enaltecendo as suas contribuições. São discutidas sugestões de trabalhos futuros, os quais possam incrementar as funcionalidades do serviço e propiciar a sua adequação a outros contextos ou áreas de pesquisa. O Apêndice A traz o código das interfaces IDL dos componentes, com comentários pertinentes. Finalmente, o último adendo (Apêndice B) traz diagramas adicionais mostrando os fluxos de interação (modelo de troca de mensagens MSC) entre os componentes do SGPOM.

Capítulo 2

Objetos Distribuídos

Neste capítulo, far-se-á uma análise contemplativa sobre a base tecnológica de suporte à construção de aplicações e sistemas distribuídos baseados em objetos, a qual serviu de referencial para a caracterização e posterior implementação do SGPOM. Serão discutidos os pontos mais relevantes que estão por trás da especificação da arquitetura CORBA — introduzindo conceitos, componentes e a sua mecânica de interações —, bem como da filosofia do Modelo de Referência ODP.

Um *Sistema Distribuído* compreende uma coleção de computadores (nós) autônomos conectados através de uma rede física e lógica e equipados com *software* projetado para produzir uma facilidade computacional integrada. Tais sistemas são construídos sobre plataformas de *hardware* geralmente variantes no número e, muitas vezes, heterogêneas. O conceito de *Objetos Distribuídos*, por sua vez, segue os princípios do Modelo *Cliente-Servidor* e do Paradigma de Orientação a Objetos, trazendo o melhor dos dois mundos: a abstração, que leva à redução da complexidade no desenvolvimento das aplicações, e a concentração de dados comuns (compartilhados por diversas aplicações) num único lugar, o que evita replicações desnecessárias e leva à especialização de serviços.

No modelo distribuído, os objetos são distinguidos em duas classes: aqueles que provêem serviços e/ou recursos (chamados de *objetos servidores*) e aqueles que requisitam esses serviços (conhecidos como *objetos clientes*). Tais objetos interoperam normalmente, seguindo a idéia clássica de troca de mensagens; contudo, agora, eles não precisam estar contidos numa só aplicação, nem sequer estarem localizados em uma mesma máquina.

Pode-se identificar seis características-chave responsáveis pela funcionalidade a ser provida por sistemas distribuídos baseados em objetos [Coul94]:

- **Compartilhamento de recursos**: onde recursos podem representar itens de dados ou componentes de *hardware* ou *software*. Pelo fato de tais recursos estarem sendo

visados por mais de um processo, é necessário um mecanismo externo de gerência. Em sistemas Cliente-Servidor, isso é feito por um processo servidor ou por um serviço genérico oferecido por diversos processos cooperantes. Por outro lado, em sistemas baseados em objetos, cada recurso compartilhável é visto como um objeto.

- **Abertura**: o que implica na disponibilidade de *interfaces* bem definidas para os recursos gerenciados. Tais interfaces devem ser, de algum modo, publicadas a fim de serem acessadas por mecanismos de comunicação entre processos (IPCs).
- **Concorrência**: que garante o alto desempenho, pois vários processos servidores podem estar atendendo, simultaneamente, a diferentes requisições.
- **Escalabilidade**: propiciada a partir da replicação dos dados e da distribuição da carga entre os processos servidores.
- **Tolerância a falhas**: o que implica na redundância a ser introduzida a fim de que tarefas essenciais possam ser realocadas, evitando a corrupção do serviço.
- **Transparência**: que assegura que a separação do serviço em componentes disseminados pelo sistema não seja percebida por seus clientes nem altere o desempenho de suas tarefas.

A distribuição dos objetos pela rede (ou sistema) deve levar em consideração alguns aspectos, tais como a freqüência de troca de mensagens, fatores de qualidade de serviço e a granularidade dos objetos. Por exemplo, não seria interessante, nem prático, fazer de um simples atributo inteiro um objeto compartilhável, haja vista que a troca de mensagens poderia se tornar intensa. Por outro lado, para um serviço que provenha algum tipo de busca ou armazenagem de informações (um serviço de nomes, de diretórios ou de gerência da persistência de objetos multimídia), seria adequado que este seja "visto" e acessado por diversos clientes.

2.1 RM-ODP

O Modelo de Referência para Processamento Distribuído Aberto (RM-ODP) [Raym95] é um produto dos esforços conjuntos entre os organismos de padronização ITU-T (*International Telecommunications Union*) e ISO (*International Organization for Standardization*) na busca do desenvolvimento de uma estrutura de suporte à distribuição, interconexão, interoperabilidade e portabilidade de sistemas computacionais. Tal Modelo é baseado em conceitos derivados de pesquisas e implementações na área de processamento distribuído.

A motivação que está por trás dessa padronização envolve os seguintes pontos:

- Avanços na área de Redes de Computadores permitiram a interconexão de diversos equipamentos, possibilitando uma distribuição global de informações. No entanto, à medida que o volume disponível de informações aumenta, crescem também os problemas com respeito à confiabilidade e à consistência dessas informações.
- Seguindo esse mesmo raciocínio, devem ser considerados os problemas de segurança concernentes ao acesso a essas informações, ou seja, deve-se garantir a veracidade dos dados para que os mesmos reflitam o verdadeiro estado do sistema.

- As necessidades de compartilhamento de informações, concorrência no acesso a recursos e cooperação entre tarefas ou aplicações devem ser satisfeitas sem afetar o comportamento de um ou outro sistema.
- Nesse contexto, a heterogeneidade dos recursos (equipamentos, Sistemas Operacionais, linguagens e protocolos de comunicação) e a portabilidade das aplicações devem ser levadas em consideração, ainda mais com o grande número de plataformas disponíveis atualmente no mercado. Um sistema distribuído deve ser extensível, assegurando, por conseguinte, a coexistência desses recursos.

De uma maneira sucinta, pode-se dizer que a proposta do padrão ODP está calcada sobre um conjunto de objetivos básicos, sendo alguns deles [Tsch93]: alcançar a portabilidade de aplicações entre plataformas heterogêneas; levar à interconexão de Sistemas ODP, ou seja, garantir uma troca significativa de informações e o uso conveniente de todas as funcionalidades por todo o sistema distribuído; e proporcionar uma transparência de distribuição, isto é, esconder as conseqüências da distribuição tanto do programador de aplicações como do usuário.

Para esse fim, a especificação do RM-ODP é composta de:

1. **Um guia de uso e de visão geral do modelo de referência** [ITU-T94a]: o qual contém a motivação do padrão, dando o escopo, a justificação e a explicação dos conceitos-chave, além de algumas linhas gerais sobre a sua arquitetura. Aqui se explica como o ODP deve ser entendido e aplicado pelos usuários, sendo inclusos, nessa categoria, arquitetos e padronizadores de sistemas concordantes.
2. **Modelo Descritivo** [ITU-T95a]: que contém a definição, a estruturação e a notação de conceitos para uma descrição normatizada de sistemas de processamento distribuído. Compreende um nível de detalhe no suporte ao Modelo Prescritivo e no estabelecimento de requisitos a novas técnicas de especificação.
3. **Modelo Prescritivo** [ITU-T95b]: que contém o conjunto das características que qualificam um sistema de processamento distribuído como aberto. Nesse modelo, são descritos os requisitos do Modelo de Referência ODP.
4. **Semântica Arquitetônica** [ITU-T94b]: que contém a formalização dos conceitos básicos de modelagem ODP definidos no Modelo Descritivo. A formalização é atingida interpretando-se cada conceito em termos da construção de diferentes técnicas padronizadas de descrição formal.

Consoante a filosofia proposta por esse Modelo de Referência, em vez de se lidar com toda a complexidade de um sistema distribuído, pode-se observá-lo sob *pontos de vista* distintos, cada qual refletindo um conjunto de atributos diferentes, com ênfase sobre um determinado assunto. Cada ponto de vista representa um nível de abstração particular do sistema distribuído original, suprimindo a necessidade de se criar um grande modelo que descreva todos esses níveis em detalhes.

A caracterização do sistema sob cada ponto de vista permite a verificação tanto da completude da sua descrição geral como da consistência entre as diferentes abstrações possíveis. Portanto, é interessante notar que esse Modelo não é hierárquico, visto que não estipula uma dependência seqüencial, tampouco uma hierarquia, entre os pontos de vista, considerados ortogonais. No padrão RM-ODP, reconhecem-se cinco pontos de vista [Tsch93] [Lini95] [Raym95] [Ferr96]:

- **empresarial** ou **organizacional**, que se detém às políticas de negócios e de gerência e às regras do usuário em relação ao sistema e ao ambiente com o qual interage. O uso da palavra "empresa" aqui não implica numa limitação a uma única organização; o modelo construído pode descrever muito bem a interação entre um grande número de organizações distintas. Esta visão procura caracterizar os requisitos empresariais.
- **de informação**, o qual enfoca sobre o modelo semântico da informação, provendo uma interpretação geral e consistente do sistema e cobrindo as fontes, os destinos e os fluxos dos dados. Recursos abstratos são modelados, bem como as regras e restrições que governam a manipulação da informação.
- **computacional**, o qual focaliza sobre os algoritmos e estruturas de dados, bem como sobre os requisitos de transparência e distribuição.
- **de engenharia**, o qual salienta os mecanismos e funções requeridos para o suporte à interação distribuída entre os objetos do sistema.
- **tecnológico**, que se detém aos detalhes de componentes e *links* com os quais um sistema distribuído é construído. Aqui são feitas considerações de *hardware* e *software* que devem compor o sistema.

Os pontos de vista mais relevantes à construção do SGPOM são o computacional e o de engenharia, haja vista especificarem, em suas *linguagens* definidas no Modelo Prescritivo, uma lista de funcionalidades a serem proporcionadas por uma plataforma distribuída (multimídia ou não) aderente ao padrão ODP. Tais funcionalidades compreendem instrumentos para a gerência de, e a interação entre, objetos, além de um conjunto de transparências e funções. Quanto às transparências, as mais significativas para o propósito do novo serviço são:

1. **Transparência de Acesso:** escondendo os diferentes mecanismos que garantem que recursos (objetos de informação) locais e remotos sejam acessados usando operações idênticas. Provê a interoperabilidade entre arquiteturas de computadores e linguagens de programação heterogêneas.
2. **Transparência de Localização:** ocultando a topologia (estrutura) do sistema. Mascara a localização de interfaces.
3. **Transparência de Replicação:** assegurando que múltiplas instâncias dos objetos de informação sejam usadas para incremento da confiabilidade e do desempenho, sem que isso seja percebido pelos usuários ou programas-cliente.
4. **Transparência de Migração:** encobrindo a heterogeneidade dos componentes do sistema a fim de possibilitar a migração de funções e aplicações.
5. **Transparência a Falhas:** contornando os efeitos de falhas e erros em componentes do sistema e naqueles de comunicação.
6. **Transparência de Recursos (Persistência):** escondendo a distinção entre componentes ativos e passivos [Davi93]. Mascara a alocação/desalocação e a ativação/desativação de recursos para os componentes do sistema.
7. **Transparência de Federação:** não revelando os limites entre domínios administrativos e/ou técnicos.
8. **Transparência de Grupo:** ocultando o uso de um grupo de objetos no suporte a uma interface computacional.
9. **Transparência de Transação:** escondendo a coordenação de operações transacionais. Possibilita a manutenção da integridade e consistência dos dados, aumentando a confiabilidade e a tolerância a falhas, e a melhoria do desempenho via a execução de operações em paralelo.

No caso das funções, o RM-ODP oferece as de: *Gerência*, para controlar o ciclo de vida dos objetos, os quais são agrupados numa hierarquia formada por *nós*, *cápsulas*, *clusters* e *objetos básicos de engenharia* [Arau96]; *Coordenação*, para orquestrar as atividades de grupos de objetos distribuídos; *Repositório*, para manutenção de bases de dados gerais e específicas de informação, provendo serviços de armazenagem e persistência de objetos; e *Segurança*, para o estabelecimento de um marco padrão de segurança que garanta restrição de acesso aos objetos.

Quanto à interação entre os objetos, o Modelo ODP prevê que esta se dê através do estabelecimento de uma conexão denominada de *canal*. O conceito de canal é importante para aplicações multimídia distribuídas, pois tal mecanismo pode transportar fluxos contínuos de dados, ter suas grandezas (tais como taxa de transmissão e atraso) monitoradas, ter seu fluxo sincronizado com o de outro de mesmo jaez, além de oferecer, adicionalmente, comunicação multiponto (de uma fonte para muitos destinos). Uma discussão aprofundada desse recurso, trazendo detalhes de implementação, é apresentada por Cardozo e outros [Card98].

Dentro de um modelo de arquitetura de uma plataforma multimídia distribuída [Loyo93], as funcionalidades aqui descritas estão mais diretamente relacionadas a uma camada conhecida como *middleware*. Pode-se dizer que seu propósito é o de prover serviços de processamento distribuído aberto às camadas superiores, como as de *groupware* e de aplicação. Contudo, sendo RM-ODP um padrão de referência, este somente especifica quais funcionalidades devem estar disponíveis às aplicações e não como estas devam ser implementadas. Para isso, padrões adicionais devem surgir. CORBA, descrita na próxima seção, por intermédio do seu conjunto de serviços, parece atender, pelo menos em parte, a essa demanda.

Para maior aprofundamento sobre as nuanças desse Modelo, pode-se recorrer a trabalhos introdutórios anteriormente desenvolvidos sobre a Plataforma Multiware [Gunj95] [Arau96], bem como a artigos clássicos sobre o assunto [Lini95] [Raym95].

2.2 CORBA

CORBA (*Common Object Request Broker Architecture*) [OMG95a] [Vino93] é uma arquitetura desenvolvida a partir da filosofia do OMG (*Object Management Group*), sendo o principal produto desse consórcio. Tal arquitetura se insere no contexto de Sistemas Distribuídos e foi criada com a finalidade de satisfazer os requisitos de *portabilidade* e *interoperabilidade* de aplicações orientadas a objeto.

A padronização CORBA assume um caráter diferenciado, haja vista ser o OMG uma associação de âmbito mundial composta por entidades empresariais. Esse consórcio possui como princípios a promoção da abordagem de orientação a objetos para a engenharia de *software* em geral e o desenvolvimento de especificações a partir de implementações e produtos comerciais já existentes ou em projeto. CORBA se insere no contexto da *OMA* (*Object Management Architecture*), que se concretizou como o modelo de referência do OMG e que será introduzida a seguir. Os

componentes da CORBA, bem como as suas inter-relações, serão investigados em uma segunda fase.

Sucintamente, pode-se referir à CORBA como a especificação da funcionalidade de um condutor lógico de mensagens — conhecido como *ORB* — responsável pelo transporte de invocações a operações remotas e de seus correspondentes resultados. Objetos CORBA residem em qualquer lugar de um ambiente distribuído, não interessando o modo como são implementados. Desse modo, CORBA suporta a heterogeneidade existente entre plataformas de *hardware* e *software*.

2.2.1 OMA

OMA [Sole95a] é uma estrutura que engloba toda a tecnologia adotada pelo OMG. Para isso, provê dois modelos fundamentais nos quais se baseiam CORBA e as outras interfaces padrão, quais sejam o *Modelo de Objetos Básico* (*Core Object Model*) e o *Modelo de Referência*.

Para a gerência de objetos distribuídos, é necessário que se atenda às requisições e aos aspectos dos objetos, proporcionando-se os meios:

- para que se aceite requisições, isso via um protocolo ou interface que descreva as mensagens que eles aceitam;
- pelos quais tais objetos devam ser construídos (via uma fábrica ou construtores);
- para gerenciar os seus conteúdos e extensões (por exemplo, capacidade de *Ciclo de Vida*).

A abordagem usada para implementar tais aspectos de um objeto é referida como um Modelo de Objetos. O Modelo de Objetos do OMG [Sole95b] define a semântica comum (interpretação) do que seja um objeto, mediante a especificação de suas características visíveis externamente, isso de um modo totalmente independente de padrões e implementações. Nesse Modelo Básico, clientes requisitam serviços (de servidores) por meio de uma interface bem definida. Para esse fim, o Modelo define os conceitos que permitem que o desenvolvimento de aplicações distribuídas seja facilitado por um ORB (*Object Request Broker*).

Os principais objetivos a serem alcançados com o Modelo de Objetos do OMG são a portabilidade e a interoperabilidade. O primeiro se traduz na habilidade de se construir aplicações cujos componentes não se fiem à existência ou à localização de uma particular implementação. Por isso, o Modelo não define a sintaxe de descrição de interfaces, mas sim as semânticas de tipos e seus relacionamentos.

A interoperabilidade, por outro lado, se baseia na invocação a objetos, negligenciando-se a linguagem de implementação. Isso é obtido pelo ORB, o qual se apóia nas semânticas de objetos e operações descritas no Modelo Básico. O ORB requer extensões adicionais que ofereçam especificações para protocolos específicos de comunicação, além de uma sintaxe de definição de interfaces e serviços básicos para a implementação dos objetos. CORBA provê tal ampliação.

O Modelo de Objetos Básico não se constitui num metamodelo [Voge97]. Isso significa que não se pode ter mais de uma instância concreta dos seus conceitos básicos. Tais conceitos servem apenas para fins de compreensão dos objetos e suas interfaces, não se podendo, todavia, redefini-los ou substituí-los, somente estendê-los para torná-los mais concretos.

Na extensão proposta pela CORBA, *tipos* são definidos usando uma lógica de predicados e formam uma hierarquia baseada em herança, com o tipo *Object* como raiz. Diz-se que se um tipo de objeto é derivado de um outro, então o segundo pode substituir[1] o primeiro. Tipos de objetos podem ser especificados como parâmetros e tipos de retorno para operações, assim como podem constituir outros tipos estruturados. Um exemplo de tipo especializado em CORBA é a *exceção*, que contém campos opcionais (compostos por outros tipos de dados) para descrever informação adicional acerca da causa da conclusão anormal (extraordinária) de uma operação.

No modelo estendido, um cliente acessa um Objeto[2] (servidor) por intermédio de uma requisição. Tal requisição é um evento e leva informações que incluem uma operação, uma referência (conhecida como *Object Reference*) ao provedor do serviço e parâmetros (se algum). Aqui é necessário que se faça uma distinção explícita entre *Implementação de Objeto* e *Referência a Objeto*. Resumidamente, a primeira é o código que implementa as operações descritas por uma definição de interface feita sob uma linguagem específica (*IDL*), enquanto a segunda compreende a identidade do objeto, a qual o destingue dos demais. A Referência a Objeto é um nome (na verdade, uma abstração) disponível aos clientes para fins de caracterização confiável de um objeto servidor específico.

Uma Implementação de Objeto é a parte de um objeto CORBA providenciada por um programador. Ela inclui, geralmente, um estado interno e, freqüentemente, interage com outras entidades do ambiente, tais como bases de dados, monitores ou elementos de rede. Já Referências a Objetos são identificadores opacos — pois contêm informação suficiente para que o ORB envie uma requisição a uma particular Implementação, sendo tal informação inacessível aos seus clientes — construídos sob certa sofisticação, para que indiquem com precisão a localização e o tipo de um Objeto. Isso municia ao ORB subsídios para redirecionar a requisição, caso o Objeto tenha migrado, ou ativar a Implementação, caso a mesma não esteja em execução.

Como se observa, trata-se de um modelo do tipo Requisição/Resposta, onde o ORB é o componente capaz de conduzir uma invocação a seu alvo e retornar os resultados dessa invocação. A Figura 2.1 [Yang96] dá uma dimensão de como seria um objeto CORBA conforme o Modelo estendido[3] proposto pelo OMG.

---

[1] O Modelo de Objetos do OMG define *Substitutabilidade* (de *Substitutability*) como a propriedade de se usar um tipo (ou interface) no lugar de outro sem que se tenha "erro de interação".

[2] A desinência *Objeto*, neste capítulo, faz alusão a um objeto CORBA.

[3] A alguma combinação entre o Modelo de Objetos Básico (*Core*) e uma ou mais de suas extensões dá-se o nome de *Profile*.

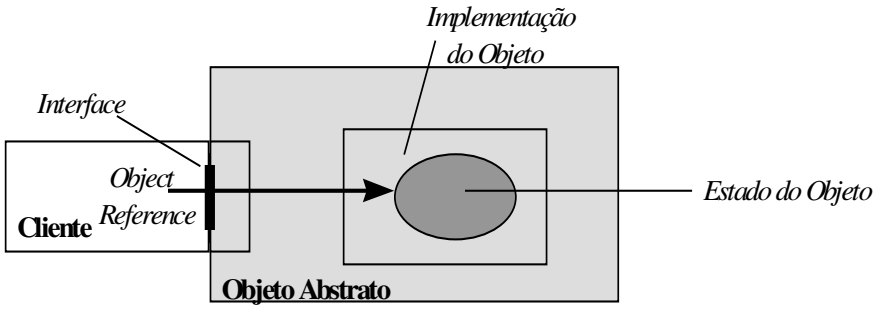

***Figura 2.1**: Modelo de um objeto CORBA.*

Por sua vez, o Modelo de Referência da OMA se apresenta como uma visão em alto nível de um ambiente distribuído completo. É uma estrutura para a padronização de interfaces e serviços a serem usados pelas aplicações. Consiste de cinco componentes que podem ser divididos em duas partes — a dos orientados ao sistema e a daqueles orientados à aplicação:

1. **Orientados ao sistema:** parte composta pelo ORB e por serviços adicionais usados na gerência dos objetos distribuídos (*Object Services*).
2. **Orientados à aplicação:** parte composta por serviços que deverão ser usados em comum por diversas e diferentes aplicações (*Common Facilities*), localizadas em vários domínios industriais (*Domain Interfaces*), e pelo conjunto de objetos desenvolvidos para aplicações distribuídas específicas (*Application Interfaces*).

Desses componentes, o mais importante é o ORB, o qual se constitui em um barramento lógico de mensagens responsável, diretamente, pela interação entre os objetos. Esse componente provê os mecanismos básicos de comunicação pelos quais os objetos, transparentemente, fazem requisições e recebem respostas. É responsável pela mediação entre aplicações, pela gerência das conexões e pela entrega confiável dos dados, mesmo para aquelas aplicações localizadas em plataformas heterogêneas e/ou implementadas sob diferentes técnicas. CORBA é, desse modo, a especificação padronizada de um ambiente que fornece alto grau de interoperabilidade.

O conjunto de interfaces para serviços básicos usados no desenvolvimento de objetos distribuídos são chamados de CORBAservices (antigos *Common Object Services*). Tais serviços devem ser usados por desenvolvedores de aplicações para gerenciar a criação de objetos e seus dados, para controle de acesso a informações sobre estes, para localizá-los, nomeá-los ou coordená-los. Já as *Common Facilities* (atualmente sob o nome de CORBAfacilities) provêem interfaces padronizadas para serviços de aplicações comuns construídas entre domínios. Compreendem serviços a serem usados pelos objetos de aplicação e não para a manutenção destes. As outrora conhecidas *Vertical Facilities*, que recebem atualmente a designação de *Domain Interfaces*, focam sobre domínios particulares de aplicação, tais como Internet, Telecomunicações e Objetos de Negócios. Finalmente, as *Application Interfaces*, por serem privativas a uma dada aplicação e pelo fato do OMG não desenvolver especificações de aplicações, não são padronizadas. A Figura 2.2 identifica esses componentes.

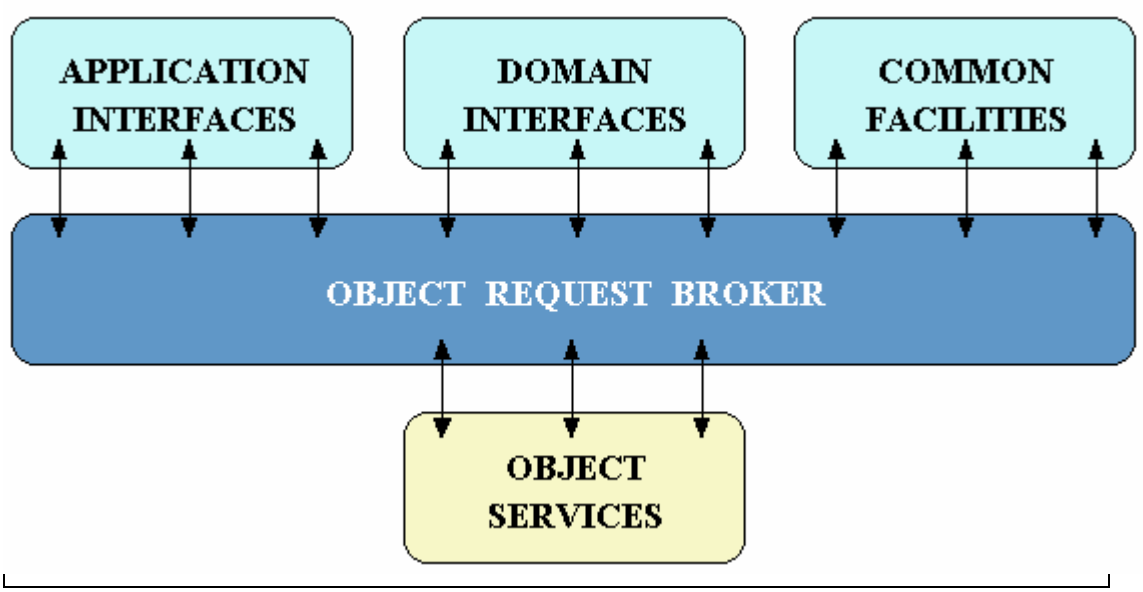

***Figura 2.2**: O Modelo de Referência da OMA.*

2.2.2 Interface *versus* Implementação ➔ IDL

Segundo o Modelo de Objetos do OMG, uma interface de um Objeto especifica o comportamento do mesmo, bem como declara o tipo de serviço que oferece. Esse artifício esconde os detalhes de implementação via encapsulamento (os atributos — estado — do Objeto só podem ser acessados por seus métodos visíveis externamente). Portanto, pode-se dizer que o *comportamento de um Objeto é independente de sua implementação*. Este é um princípio fundamental para o "universo" CORBA: basta que o Objeto especifique o seu comportamento (mediante a sua interface), para que tenham definidas as suas funcionalidades, não sendo exigido que se saiba como é a sua implementação.

Para tomar vantagem desse princípio, a arquitetura CORBA especifica a sua linguagem padrão de definição consistente de interfaces (IDL — *Interface Definition Language*). Uma interface descrita em IDL atua como um *contrato* entre desenvolvedores de Objetos e seus eventuais usuários. Dessa forma, o que um cliente precisa saber para enviar uma requisição a um Objeto é a sua interface. É interessante notar que a IDL não é mais uma linguagem de programação, sendo independente de qualquer uma existente. É somente uma linguagem neutra, uma língua franca para expressar tipos, especificamente os de interfaces [Bake97].

Dentre os "ingredientes" da Orientação a Objetos, CORBA não suporta o *polimorfismo*. Suporta *encapsulamento* e *herança* de interfaces (incluindo herança múltipla), por meio da IDL. Não suporta *overriding* (redefinição), nem especialização e sobrecarga de operações.

Pelo Modelo de Objetos do OMG, cada objeto pode ter somente uma única interface (ao contrário, por exemplo, do Modelo de Objeto Computacional ODP, onde é possível a existência de múltiplas interfaces). Por essa razão, os termos objeto e interface são geralmente considerados equivalentes. Caso seja necessária mais de uma

interface, o mecanismo de herança pode ser usado. Todavia, uma proposta está sendo submetida para que uma próxima especificação CORBA (CORBA 3.0) introduza algum mecanismo ou modelo para que um Objeto suporte múltiplas interfaces [Vino98].

O código escrito em IDL serve de fonte para um tradutor, chamado de *compilador IDL*. A função desse tradutor é fazer o mapeamento de IDL para alguma outra linguagem (conhecida como linguagem-alvo), notadamente uma linguagem de implementação. Atualmente, a especificação CORBA 2.0 adota os seguintes mapeamentos possíveis:

- IDL para C;
- IDL para C++;
- IDL para Smalltalk;
- IDL para Ada;
- IDL para Cobol;
- IDL para Java.

A sintaxe da IDL é derivada da de C++, suportando tipos básicos e compostos; no entanto, remove as construções relativas a implementações e adiciona novas palavras-chave, não ambíguas, para especificar sistemas distribuídos. Os tipos de construções suportados são *constantes*, *atributos*, *operações*, *interfaces* e *módulos* [Voge97]. A sua hierarquia de tipos é apresentada na Figura 2.3 [Yang96].

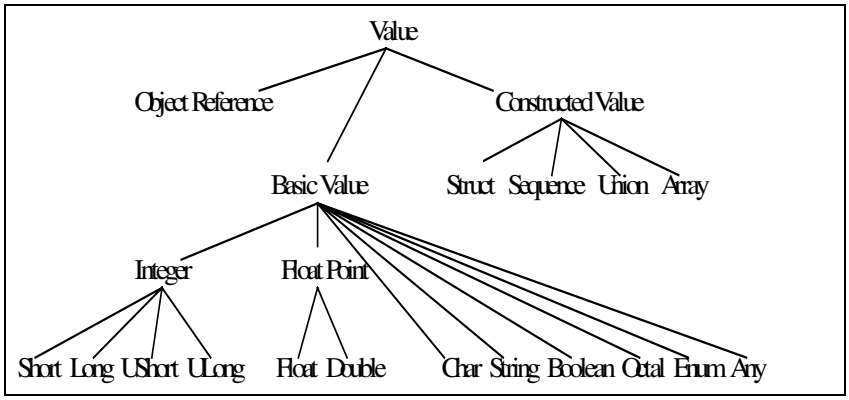

***Figura 2.3**: Hierarquia de tipos em IDL.*

O código produzido pelo compilador IDL ocorre em pares: mapeamentos para o lado do cliente e para o lado do servidor. O primeiro é chamado de *Stub* e o segundo de *Skeleton*[4]. Esses componentes devem ser estaticamente ligados (*linked*), em tempo de compilação, respectivamente, aos códigos das aplicações cliente e servidor. Porém, antes da compilação do servidor, o *Skeleton* necessita estar atrelado ao código de implementação de cada método seu. Já o *Stub* está basicamente completo. A fim de usá-lo, basta a aplicação-cliente invocar um ou mais de seus métodos, o que irá gerar uma ou mais requisições ao serviço remoto. Em linguagens de implementação

[4] O *Stub* é um pedaço de código responsável pela tradução dos tipos de dados inerentes da linguagem de programação usada no lado do cliente para um formato padrão, neutro a plataformas, de composição de mensagens. O *Skeleton* faz justamente o contrário para o lado do servidor. Por conseguinte, diz-se que são responsáveis pelas operações de *marshalling* e *unmarshalling* (dos parâmetros) das requisições.

orientadas a objetos, o *Stub* funciona como um objeto local representante (*proxy*) do servidor. A Figura 2.4 esboça, coesamente, esse raciocínio.

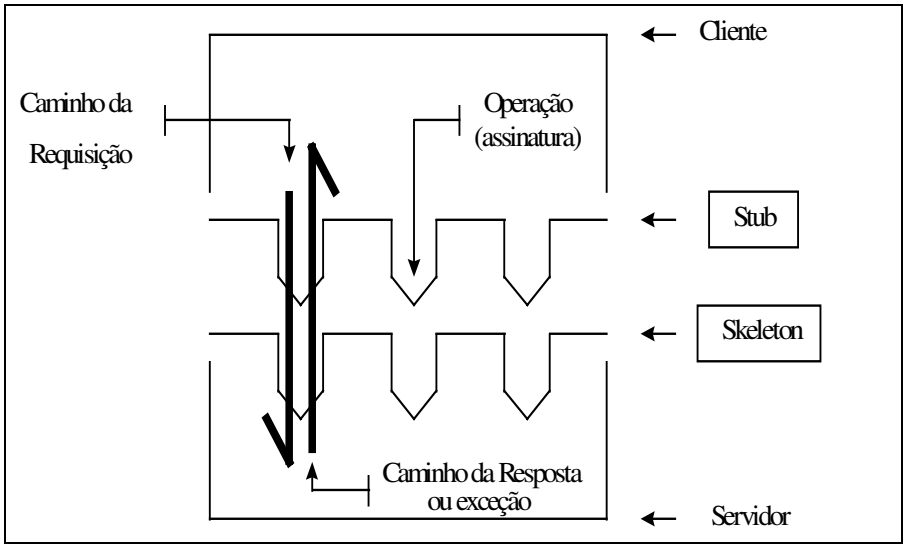

***Figura 2.4**: Requisições e Respostas entre um cliente e um servidor.*

2.2.3 Componentes da Arquitetura e Mecânica da Invocação via ORB

O diagrama da Figura 2.5 mostra os principais componentes da arquitetura CORBA e suas interconexões:

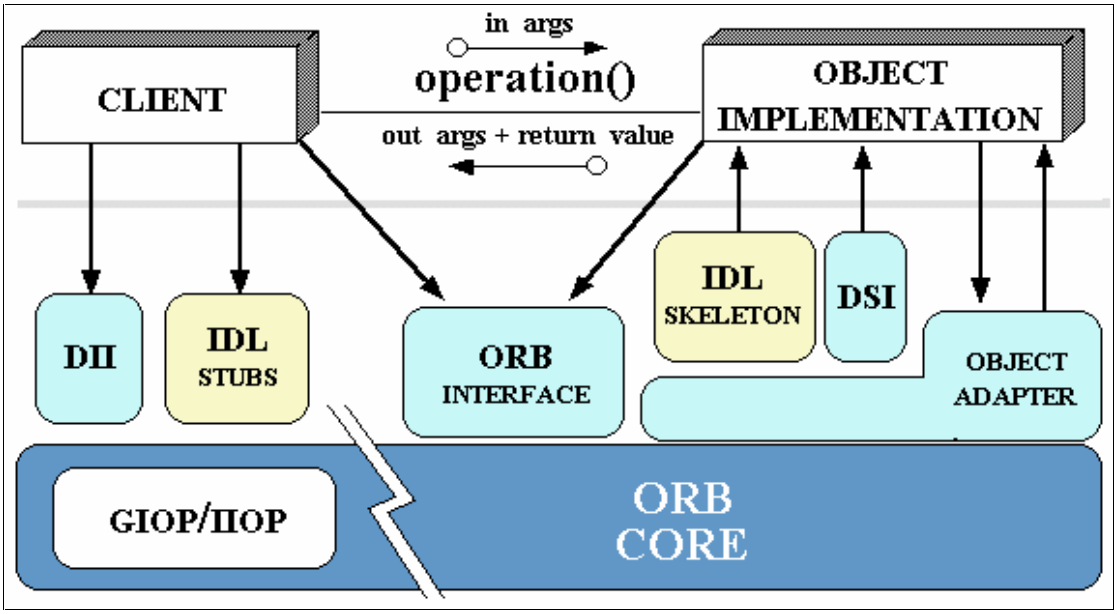

***Figura 2.5**: Arquitetura CORBA e seus componentes de interação estática e dinâmica.*

O componente central da arquitetura é o ORB (cujo cerne, responsável pela comunicação das requisições, é chamado de *ORB Core* - Núcleo ORB). A sua função básica consiste em repassar as invocações (requisições) dos clientes para as Implementações de Objeto correspondentes, escondendo as diferenças impostas pelo Sistema Operacional (característica de um *middleware*).

A fim de fazer uma requisição, o cliente pode se comunicar com o *ORB Core* por intermédio do *Stub* IDL ou do *Stub* da Interface de Invocação Dinâmica (*DII*). O *Stub* passa a representar, então, a interface entre a linguagem de implementação do cliente e o *ORB Core*. O *ORB Core* passa a requisição do cliente para a Implementação (objeto servidor ativo) via uma chamada feita ao *Skeleton* IDL ou à Interface de *Skeleton* Dinâmica (*DSI*). Os códigos do *Stub* e do *Skeleton* são usados para invocações estáticas, ou seja, para os casos onde a declaração IDL de um Objeto já tenha sido definida em tempo de compilação. Por outro lado, requisições podem ser construídas e respondidas, dinamicamente (em tempo de execução), mediante o uso das interfaces padronizadas (DII e DSI).

A DII define outra semântica de execução para as operações invocadas, usando *pseudo-objetos*[5] do tipo *Request*. Além da semântica síncrona usual, conhecida como *at-most-once* (que diz que se uma operação retorna um valor ou é abortada, pode-se deduzir que a mesma foi executada pelo menos uma vez), uma outra opção não-bloqueante, conhecida como *deferred-synchronous*, passa a estar disponível. Sob essa semântica, fica a cargo do código-cliente saber se o servidor respondeu à sua invocação corretamente ou não (pelo uso de um mecanismo de *polling*).

De uma maneira análoga, a DSI permite que os objetos servidores sejam implementados sem se ter disponível o *Skeleton*. Nesse caso, o servidor pode lançar mão dessa interface para avisar, dinamicamente, ao ORB que se alguma requisição chegar possuindo uma Referência a Objeto específica (cuja Implementação de Objeto foi colocada em execução por esse processo servidor), que a mesma seja roteada diretamente para ele. Ao contrário dos outros componentes da arquitetura, a DSI foi introduzida apenas a partir da especificação CORBA 2.0. Ela foi criada para suportar a implementação de *gateways* entre ORBs que utilizam protocolos de comunicação díspares; ao mesmo tempo, auxilia na construção de aplicações que interligam componentes CORBA àqueles fora desse ambiente.

Como se observa pela Figura 2.5, a arquitetura CORBA define uma interface de comunicação com o ORB, acessível tanto para o lado do cliente como para o do servidor. Essa interface trata, principalmente, de aspectos de iniciação do ORB como, também, de manipulação de Referências a Objetos.

A comunicação entre a Implementação do Objeto e o *ORB Core* é efetivada pelo Adaptador de Objetos (*Object Adapter — OA*). Este manipula serviços especiais, tais como geração e interpretação das Referências a Objetos, segurança das transações (pela autenticação da invocação), ativação e desativação dos Objetos e das Implementações, ao mesmo tempo que gerencia processos do Sistema Operacional. Enquanto uma interface é responsável pelo tipo de implementação, um OA fica incumbido pelo seu "estilo".

---

[5] As interfaces para componentes do ORB são todas especificadas em IDL. Isso provê uma representação neutra da sua interface computacional. Contudo, certas partes dessas definições são designadas em *pseudo-IDL* (PIDL), o que significa que suas implementações não serão feitas, necessariamente, via tipos de dados ou objetos CORBA (mas, também, via bibliotecas). Qualquer definição de interface comentada como "pseudo-IDL" pode ser implementada como um pseudo-objeto.

A especificação CORBA 2.0 somente define um único tipo de Adaptador de Objetos, conhecido como *BOA* (*Basic Object Adapter*), cujo propósito é o de gerar e interpretar Referências a Objetos e ativar/desativar as Implementações de Objeto. Para o BOA, são descritos quatro estilos de ativação:

1. **Ativação por método:** neste caso, um novo processo é iniciado sempre que há uma nova requisição (invocação) a um dos métodos (implementação das operações) do Objeto.
2. **Ativação compartilhada**: onde múltiplos Objetos podem ser lançados por um mesmo programa servidor (e estarem ativos num mesmo instante).
3. **Ativação não compartilhada**: onde somente um único Objeto pode estar ativo. Múltiplas invocações a métodos podem ser tratadas pelo único servidor, já que eles pertencem ao mesmo Objeto.
4. **Ativação persistente**: onde os objetos servidores estão sempre ativos (não requerem, portanto, ativação).

Pelo que foi posto, caso uma requisição seja feita, o Objeto deve ser ativado pelo Adaptador (exceto para a última configuração). Para tanto, o OA necessita ter acesso às informações relativas àquele Objeto (tais como, sua localização e o seu modo de operação). Tais informações estão guardadas numa base de dados conhecida como *Repositório de Implementações* (não retratada no diagrama), a qual atua como peça-chave quando do *binding* a *Object References* persistentes [Henn98].

Uma próxima especificação CORBA introduzirá, provavelmente, dois OAs adicionais. O primeiro seria o ODA (*Object Database Adaptor*), a ser formalizado pelo ODMG (*Object Data Management Group*), o qual está trabalhando na padronização de um adaptador que possibilite a integração entre o ORB e um SGBDOO. Tal adaptador deverá levar em conta certos requisitos desse último, tais quais: ativação por longos períodos de tempo, suporte à concorrência e recuperação (*Recovery* e *Rollback*) depois de falhas. O segundo adaptador seria o POA (*Portable Object Adaptor*), o qual estabeleceria mecanismos de suporte à mobilidade dos objetos servidores [Vino98].

As interfaces dos Objetos podem ser compiladas para serem adicionadas ao *Repositório de Interfaces*, outro componente da arquitetura, não retratado no diagrama. Como dito anteriormente, a DII permite ao cliente especificar requisições a Objetos cujas definições de interface sejam desconhecidas por ele durante a sua compilação. Para usá-la, o cliente deverá compor uma requisição, na qual devem ser incluídas, dinamicamente, a Referência ao Objeto, a operação e a sua lista de parâmetros. As especificações dos Objetos e dos serviços que eles provêem são recuperadas pela DII a partir do Repositório de Interfaces, o qual mantém uma armazenagem persistente das definições de suas interfaces. Analogamente, tal mecanismo é, também, passível de ser acessado no lado servidor pela DSI.

Finalmente, uma forma temporal de invocação que demonstra as interações entre parte dos componentes acima descritos é dada pelo diagrama da Figura 2.6.

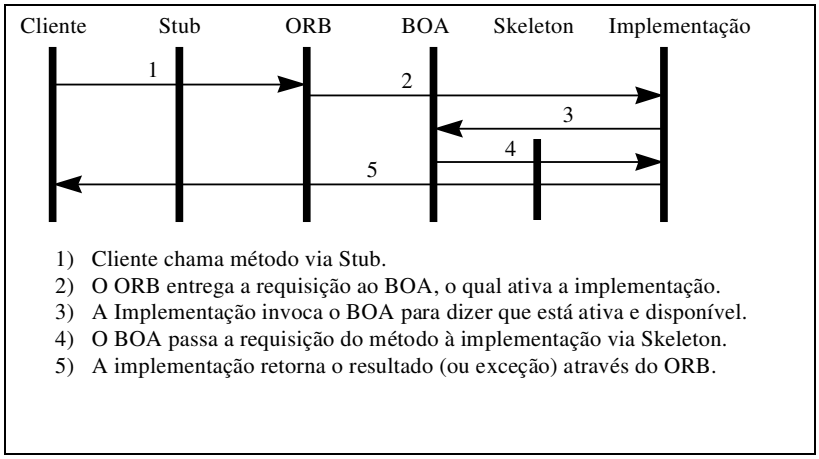

***Figura 2.6**: Diagrama temporal de uma invocação estática.*

2.2.4 Interoperabilidade entre ORBs

Antes da especificação CORBA 2.0, um dos grandes problemas acerca de produtos ORB era que os mesmos não poderiam interoperar. A carência de interoperabilidade era devida ao fato de que a especificação anterior (CORBA 1.2) não estipulava em quais formatos de dados e/ou protocolos tais implementações deveriam se basear. Era necessária uma abordagem compreensiva e flexível de suporte a redes de objetos — estes distribuídos e gerenciados por meio de vários (e heterogêneos) produtos ORB — que assumisse um caráter *universal*: seus elementos poderiam ser combinados de diversas formas, a fim de satisfazerem um grande número de produtos e usuários.

A especificação CORBA 2.0 introduziu uma arquitetura geral de interoperabilidade que provê uma interação direta ORB a ORB baseada no uso de *pontes* (*bridges*). Tal arquitetura é calcada num protocolo chamado de *GIOP* (*General Inter-ORB Protocol*), o qual especifica uma sintaxe de transferência e um conjunto padrão de mensagens para a interação entre ORBs através de qualquer conexão. Resumidamente, tal protocolo é especificado como uma interface comum (de tipo e de formato de mensagem) requerida para uma interoperabilidade geral entre ORBs, sendo, por conseguinte, projetada para ser de fácil implementação.

Existe, ainda, uma camada semântica provida por um protocolo conhecido como *IIOP* (proveniente de *Internet Inter-Orb Protocol*), o qual especifica como o GIOP deve ser construído sobre a pilha TCP/IP (padrão *de facto* da Internet). Implementações isoladas do IIOP, por sua vez, podem ser embutidas em aplicações que estejam fora do ambiente CORBA (aplicações Internet, por exemplo). Tais aplicações poderiam, de uma forma transparente, requisitar operações em objetos CORBA, sem serem projetadas, inicialmente, para esse fim (aplicações legadas).

Para efeito de caracterização, pode-se dizer que a relação entre o IIOP e o GIOP se assemelha àquela entre uma definição de interface em IDL para um Objeto e a sua Implementação. Ou seja, o GIOP especifica o protocolo, tal qual faz uma definição de interface em IDL quando define um contrato entre um objeto e seus clientes. Já o IIOP determina como o GIOP pode ser implementado usando o TCP/IP, tal como uma

Implementação de Objeto em CORBA determina como um protocolo de interface é concebido (instanciado).

Além de padronizar os protocolos, a especificação CORBA 2.0 também definiu os formatos das Referências aos Objetos, a fim de que se garantisse a interoperabilidade entre ORBs. Tais referências são usadas pelos ORBs no encaminhamento das requisições aos Objetos. Foi especificado um formato padrão de Referência conhecido como *IOR* (*Interoperable Object Reference*) [Henn98], o qual contém a informação necessária para a localização e o posterior estabelecimento da comunicação com um Objeto, passando ou não por um ou mais protocolos intermediários. Tipicamente, um IOR para o IIOP conteria informações sobre a máquina e o número da porta TCP/IP.

2.2.5 CORBAservices e CORBAfacilities

Os CORBAservices são definidos pelo OMG como "aquelas interfaces de serviços a serem usadas, largamente, na construção de aplicações baseadas no padrão CORBA" [OMG95b]. Em contraste com as CORBAfacilities, tais serviços oferecem um alicerce de suporte às aplicações e têm as seguintes características e objetivos:

- As operações providas por tais serviços são definidas para servirem como "blocos de construção", tanto para as *Common Facilities* e objetos de aplicação, como para outros serviços de objetos.
- As especificações dos serviços devem ser completas e não devem conter detalhes de implementação.
- A API de tais serviços é *modular*, ou seja, objetos de aplicação podem usar alguns ou vários desses serviços.
- As operações providas por esses serviços são especificadas em IDL. Além disso, onde aplicável, deve-se especificar a seqüência de utilização e o comportamento esperado dessas operações.
- Os serviços são independentes e consistentes, ou seja, deve ser possível se especificar e implementar cada serviço separadamente, assegurando-se a sua coexistência dentro do mesmo ambiente.
- Deve-se, sempre que possível, minimizar a duplicação de funcionalidades. Ou seja, cada serviço deve ser construído, quando apropriado, tendo como base a existência de outros já desenvolvidos. Isso assegura que uma dada funcionalidade pertença ao serviço mais apropriado. Este é um princípio muito importante quando se pensa em se estender ou especificar novos serviços e é conhecido como *Princípio de Bauhaus*.

CORBA define um grande conjunto de Serviços de Objetos como extensão às funcionalidades do ORB. Alguns são executados em servidores localizados sobre o ORB, enquanto outros devem ser construídos, pelo menos parcialmente, como parte dele. A seguir, uma lista parcial dos CORBAservices atualmente disponíveis, os quais, de uma forma ou de outra, são relevantes ou foram avaliados quando da construção do SGPOM (para uma lista mais completa, deve-se recorrer à respectiva especificação do OMG [OMG98a]):

- **Serviço de Persistência**: provê interfaces comuns para os mecanismos usados para se reter e manipular os estados persistentes de objetos, isso de uma forma independente do tipo de repositório de dados a ser empregado.

- **Serviço de Ciclo de Vida**: representa uma estrutura responsável pela criação, remoção, cópia e movimentação de um objeto CORBA, baseando-se na sua localização. Para este serviço, qualquer pedaço de código que inicie uma operação de *lifecycle* é considerado um cliente, sendo o modelo de criação definido em termos de Objetos *factories*. Uma fábrica é um Objeto que cria outros Objetos; contudo, não há uma interface padrão para essa entidade, mas sim uma genérica. Este serviço também define Objetos conhecido como *factory finders*, os quais retornam uma seqüência de fábricas para interação com o cliente.
- **Serviço de Nomes**: permite a associação de um nome simbólico a um objeto CORBA por meio de uma operação de *binding*, a qual é, por sua vez, definida dentro de um *contexto*. Cada contexto delimita um escopo de nomes na forma de um diretório e pode se agrupado com outros de mesmo gênero, formando *federações*. A partir deste serviço, um cliente pode obter a Referência a um Objeto, desde que saiba o seu nome lógico.
- **Serviço de Controle de Concorrência**: descreve como um Objeto pode mediar os acessos simultâneos feitos por mais de um cliente, de tal modo a garantir *consistência* e coerência. Tal serviço é projetado para ser usado juntamente com o de Transações, a fim de coordenar as atividades de transações concorrentes.
- **Serviço de Relações**: provê mecanismos para criação, remoção, navegação e gerência de relações entre objetos CORBA. Define dois tipos de objetos: *relações* (*relationships*) e *papéis* (*roles*). O segundo representa um típico objeto CORBA numa relação; já o primeiro é criado passando-se um conjunto de papéis a uma fábrica de relações. Este serviço permite que relações de níveis arbitrários sejam criadas.
- **Serviço de Segurança**: suporta mecanismos de *autenticação* e *autorização* pelos quais as aplicações podem restringir certas ou todas as operações de um Objeto a clientes específicos. É projetado para satisfazer uma variedade de políticas de segurança relativas a diferentes necessidades.
- **Serviço de Transações**: oferece às aplicações que se comunicam com múltiplos servidores e que atualizam diversos bancos de dados um modo garantido para que se efetue (*commit*) ou se aborte as mudanças em curso. De um lado, define interfaces simples a serem usadas pelos clientes para criarem ou abortarem as transações. De outro, especifica interfaces internas a serem suportadas por implementações desse serviço, seguindo o protocolo de duas fases (*two-phase commit protocol*).
- **Serviço de Externalização**: define protocolos e convenções para a *externalização* e a *internalização* de Objetos. A primeira operação se traduz em gravar o estado de um Objeto em um fluxo (*stream*) de dados (em memória, em um arquivo em disco, através da rede, etc.), de tal modo que seja possível, em uma segunda fase, a conversão desse fluxo em um novo Objeto, isso ocorrendo dentro do mesmo processo ou não. Este serviço traz um modo alternativo para se copiar um Objeto — a operação de cópia cria um novo Objeto e o inicia a partir do estado de um outro já existente.
- **Serviço de Consulta**: define interfaces para operações sobre coleções de Objetos, permitindo a especificação de *queries* via linguagens derivadas do SQL ou específicas para manipulação direta. Este serviço dá ênfase ao acesso refinado de Objetos.

- **Serviço de Gerência de Mudanças**: suporta a identificação e a evolução consistente de Objetos mediante a manutenção eficiente do histórico de suas versões e a gerência de configurações de sistemas complexos.

Já as CORBAfacilities (que integram a *Common Facilities Architecture*) provêem interfaces padronizadas de alto nível para serviços de aplicação bastante comuns. Assim como os Serviços de Objetos, tais *Common Facilities* são acessadas por intermédio de interfaces padrão em IDL. Há duas categorias de tais *facilities* (uma maior descrição pode ser obtida na especificação do OMG [OMG98b]):

1. **Horizontal *Facilities*:** incluem funções que cobrem grande parte ou a maioria dos sistemas, sem levar em consideração o conteúdo da aplicação. Compreendem quatro grupos de facilidades:

- Interface com usuário (*User Interface*);
- Gerência da Informação (*Information Management);*
- Gerência de Sistemas (*Systems Management*);
- Gerência de Tarefas (*Task Management*).

2. **Vertical *Facilities* (*Domain Interfaces*):** representam a tecnologia que suporta vários segmentos de mercado, compreendendo:

- Saúde (*Healthcare*);
- Sistemas Financeiros (*Financial Systems*);
- CAD (*Computer-Aided Design*).

Dentro deste documento, em um capítulo à parte, é dada uma maior ênfase a um desses CORBAservices, o Serviço de Persistência de Objetos, o qual serviu de base para a construção do SGPOM.

2.3 Integração entre Java e CORBA

Nesta seção, serão apresentados os esforços que têm sido empreendidos para que as tecnologias Java [Gosl96] [Flan97] e CORBA possam ser integradas, a fim de que se consiga o melhor dos dois mundos. A idéia aqui é discutir como Java suporta CORBA e vice-versa, sempre dentro do escopo Cliente-Servidor. Em seguida, abordar-se-á a proposta *RMI* (*Remote Method Invocation*), um modelo de objeto distribuído para a linguagem Java, comparando-o com o protocolo IIOP. Por fim, será mencionada a ferramenta ORB usada na implementação do SGPOM.

2.3.1 Java como suporte à CORBA

De certo modo, a infra-estrutura Java começa quando a da CORBA termina. Como já visto, CORBA municia um substrato de objetos distribuídos que permite que as aplicações estendam o seu alcance através e além de redes, linguagens, fronteiras de componentes e Sistemas Operacionais. Enquanto isso, o novo paradigma por trás de Java oferece suporte à implementação de objetos portáveis que trabalham sobre os principais Sistemas Operacionais existentes. CORBA trata da transparência de acesso e localização (transparência de rede), ao passo que Java lida com a transparência de

implementação. Uma lista do que Java provê para CORBA é dada a seguir [Evan97] [Orfa97]:

- Java está fazendo com que se integre CORBA a *World Wide Web* (*WWW*), mediante o desenvolvimento de ORBs em Java, os quais, por sua vez, podem ser embutidos em *browsers* (interfaces gráficas de navegação), fornecendo suporte a aplicações-cliente (conhecidas como *applets*).
- Java simplifica a distribuição de códigos em grandes sistemas CORBA. Para tanto, códigos em Java podem ser ativados e gerenciados a partir de um servidor central. Atualizações feitas no servidor são refletidas em cada um dos clientes, o que auxilia na administração de grandes redes.
- Java complementa a infra-estrutura de agências do ambiente CORBA. Para CORBA, está-se definindo um alicerce de suporte a *agentes móveis* (*Mobile Agent Facility* [OMG98d]), ainda em fase de especificação. Tal estrutura irá permitir que objetos possam vagar através dos nós de uma rede, a partir de certas regras. Tais agentes levam consigo o seu estado, o seu itinerário (caminho a ser percorrido) e o seu comportamento. Os *bytecodes* Java (códigos de execução para a Máquina Virtual Java[6]) se tornam ideais para representar e carregar esse comportamento. Portanto, nesse ponto, será possível fazer com que as *applets* Java possam assumir um caráter mais dinâmico, exercendo o papel de pequenos agentes móveis.
- Java se torna uma grande linguagem na implementação de objetos CORBA, já que parece ser ideal para se escrever códigos cliente e servidor. Isso por causa do suporte a *multithreading*, à "coleta de lixo" (*garbage collection*), a operações de *networking* e ao tratamento de exceções. Além disso, o modelo de objeto Java parece complementar o da CORBA; ambos usam o conceito de interfaces para separar a definição de um objeto de sua implementação.

2.3.2 CORBA como suporte à Java

Ao se estender as funcionalidades da WWW com CORBA, passa-se a oferecer, pelo menos, três benefícios em imediato: evitam-se os gargalos oferecidos por soluções do tipo *CGI* (*Common Gateway Interface*), principalmente quanto ao tempo de atendimento às requisições; oferece-se um ambiente Web para comunicação servidor a servidor bastante robusto e escalável; e estende-se o próprio paradigma Java com o substrato de objetos e serviços distribuídos. Isso posto, pode-se concluir que:

- Ao se lançar mão do ambiente CORBA, permite-se que os clientes (por exemplo, as *applets*) invoquem diretamente métodos a um servidor. Para tanto, o cliente passa os parâmetros usando *Stubs* pré-compilados ou os remete, dinamicamente, usando o componente de invocação dinâmica da CORBA (DII). Em ambos os casos, o servidor recebe a chamada, diretamente, via um *Skeleton* pré-compilado. Pode-se passar com essas invocações, além de tipos básicos (como *string*), qualquer outro definido pelo usuário (na forma de *structs*, *enums, sequences* e *arrays*).
- Uma das vantagens dessa integração (CORBA e Java), com relação a serviços que empregam a tecnologia CGI, é o provimento de suporte à manutenção do estado do servidor ao longo de uma sessão de invocações oriundas de um ou mais clientes. Uma aplicação em CGI não dispõe de recursos para manter um estado contínuo de execução entre as requisições. Já um servidor em forma de objeto CORBA pode

[6] *Java Virtual Machine*.

atender a várias requisições, usando, caso seja necessário para esse fim, o Serviço de Concorrência.

- Com o advento do paradigma CORBA, pode-se conseguir um melhor balanceamento da carga do sistema, distribuindo melhor as funcionalidades (e responsabilidades) entre os servidores e, por conseguinte, auxiliando no atendimento mais rápido aos clientes. Relativo a isso, CORBA fornece os CORBAservices que permitem que objetos servidores se tornem mais específicos em suas funcionalidades (modularidade).
- Afora essa integração, *applets* Java não podem se comunicar além dos espaços de endereçamento usando métodos de invocação remota. Isso significa que não há um modo fácil de tal aplicação ambulante fazer uma invocação a um método de um objeto remoto (RMI, como se verá adiante, só permite que aplicações em Java possam conversar). Com CORBA, tais *applets* podem se comunicar com outros objetos escritos em linguagens diferentes, estes localizados em outros espaços de endereçamento .

2.3.3 RMI

O Método de Invocação Remota, RMI [Orfa97] [Voge97], foi projetado para suportar invocações a métodos remotos, entre objetos localizados em diferentes Máquinas Virtuais Java. Tal proposta torna a figura do ORB quase transparente para o cliente pela adição de novos requisitos no lado do servidor. De uma maneira concisa, pode-se dizer que essa abordagem provê mecanismos semelhantes ao RPC (*Remote Procedure Call*), sendo, no caso, voltada a objetos Java.

Um objeto RMI é um objeto Java remoto cujos métodos podem ser invocados a partir de uma outra Máquina Virtual Java, mesmo através de uma rede. Pode-se fazer invocações a esse objeto, como se o mesmo estivesse disponível localmente (transparência de invocação). Sob esse aspecto, a proposta RMI é bastante similar à da CORBA. E como a da CORBA, ela permite que se passe uma referência a um objeto remoto em um argumento ou que se retorne essa referência como resultado de uma invocação.

A idéia que está por trás desse mecanismo é muito simples (ver Figura 2.7): classes *Stub* e *Skeleton* são geradas diretamente a partir de uma classe Java denotada como remota. A classe *Stub* lida com o *binding* (ligação) para com o objeto remoto e com a *serialização* (*marshalling*) dos dados/parâmetros para o lado do cliente. A classe *Skeleton* manipula as invocações que chegam.

A principal vantagem da abordagem CORBA sobre a do RMI é que ela suporta múltiplas linguagens de programação. Embora Java seja mais amigável para implementação do que C, C++, Cobol e, portanto, seja uma boa candidata a tomar o lugar dessas linguagens no desenvolvimento de *software* comercial, haverá sempre aplicações já consolidadas implementadas em linguagens mais antigas (*códigos legados*). Aplicações em Java podem apenas acessar essas antigas aplicações se elas estiverem encapsuladas (empacotadas) sob um objeto CORBA. Na verdade, RMI é um mecanismo interessante quando se está tratando de pequenas e/ou médias aplicações completamente implementadas em Java.

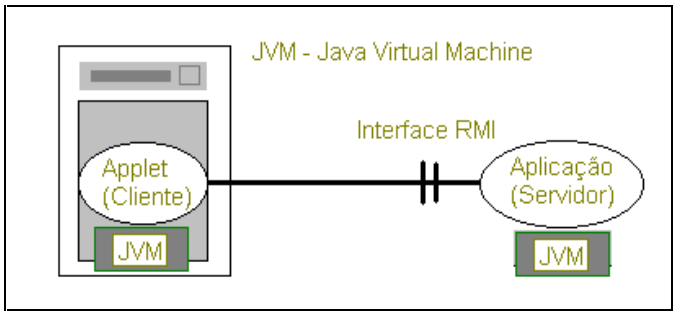

***Figura 2.7**: Método de Invocação Remota.*

Comparando-se RMI e IIOP — o protocolo de interoperabilidade da especificação CORBA 2.0 —, há determinadas razões que tornam este último um substrato de transporte aparentemente inconveniente para o sistema RMI-Java. Algumas dizem respeito a considerações de projeto; outras são devidas a divergências técnicas entre as duas abordagens. Dentre essas incompatibilidades, podem ser citadas:

- RMI faz uso marcante da habilidade do paradigma Java de realizar a baixa (*download*) de código de uma maneira segura e de possibilitar a passagem de objetos por valor (cópia da estrutura do objeto). Isso ainda não é possível dentro do contexto CORBA, onde, atualmente, apenas é permitida a passagem de objetos por referência. Portanto, não há condições para o IIOP de mover outra coisa que não seja dados ou *pseudo-objetos* CORBA. Destarte, códigos não podem ser transportados em um sistema CORBA, já que esta é uma arquitetura independente de qualquer linguagem de programação[7].
- Objetos RMI são alvo do mecanismo de *garbage collection* suportado por Java, ao passo que objetos CORBA não podem (e, de certo modo, não devem) lançar mão desse artifício. Logo, o protocolo RMI necessita suportar mensagens que tenham que ver com tal mecanismo, o que não é atendido e nem especificado pelo IIOP.
- Com relação à identificação de tipos, o IIOP reconhece tipos e métodos com referência ao Repositório de Interfaces da arquitetura CORBA. RMI, por sua vez, não apresenta mecanismo de suporte algum do gênero desse repositório. Pelo contrário, para ele, um tipo de um objeto é identificado pelo nome qualificativo geral da classe, além de um segredo criptográfico baseado na estrutura do objeto Java.
- O IIOP define tamanhos específicos para os tipos de dados primitivos (tais como, *char*, *long*, *boolean*). Portanto, às vezes, não se consegue fazer o mapeamento desses tipos para os seus correspondentes em Java.
- Finalmente, o IIOP (seguindo o modelo CORBA) provê suporte explícito a três tipos diferentes de argumentos: parâmetros *in*, *out* e *inout*. Java somente suporta a noção análoga aos parâmetros *in*. Portanto, um sistema RMI não seria capaz de atender a chamadas IIOP que fizessem uso dos outros tipos de parâmetros.

De qualquer forma, as expectativas quanto à integração desses dois mecanismos deverão ser atendidas com uma nova versão do *kit* de desenvolvimento Java (*JDK 1.2*), a qual se encontra em fase de avaliação e testes. Um novo pacote de

[7] Não obstante, há, correntemente, uma proposta de especificação de um Serviço de Objetos que atuará como responsável pelo suporte à passagem *por valor* dos objetos CORBA pelas operações. Para descrição dessa proposta, o leitor deve recorrer ao documento do OMG [OMG98c].

classes e interfaces, baseado numa primeira implementação de ORB conhecida como JOE, provê suporte à construção de objetos distribuídos Java segundo o Modelo de Objetos CORBA. Esse novo ORB, sob o cognome de *Java IDL*, traz um ambiente de desenvolvimento composto por um compilador IDL (*idltojava*) e por um Servidor de Nomes (*tnameserv*) concordante com aquele especificado pelo OMG [OMG98a]. Nesse mesmo rumo, uma tendência que parece se solidificar é a de sinergia entre as abordagens *JavaBeans*[8] e CORBA — referida como *Enterprise Beans* —, que deverá introduzir como alicerce de suporte o *JTS* (*Java Transaction Service*), o qual é baseado no Serviço de Transações (ver Subseção 2.2.5).

Contudo, até o momento, Java IDL só provê suporte a Objetos transientes (cujos tempos de vida estão limitados àqueles dos processos servidores que os contêm), o que, muitas vezes, não é interessante — o próprio *Name Server*, que pode ser visto como um Objeto ordinário, tem a sua estrutura (diretório) de nomes lógicos removida (e, por conseguinte, perdida) toda vez que pára a sua execução. Mais ainda, não há implementação do Repositório de Interfaces, o que elimina a opção de invocação dinâmica (DII).

A partir dessas considerações, quando se pensa na configuração e implementação do SGPOM, é natural que se dê mais atenção a produtos que integrem ORBs e Java, em detrimento da abordagem RMI; ainda mais que esse serviço está fortemente atrelado à especificação do POS, a qual, por sua conta, foi projetada para trabalhar em um ambiente concordante com CORBA.

2.3.4 OrbixWeb

O produto OrbixWeb3.0™ [IONA97a] [IONA97B] — o qual é acompanhado, desde a sua versão anterior, por um compilador IDL que provê um mapeamento total tanto para o lado do cliente como para o lado do servidor — foi o escolhido para suporte ao desenvolvimento do SGPOM. Tal produto vem passando por um processo de amadurecimento constante, sendo empregado em outros projetos acadêmicos. Como oferece uma implementação robusta do IIOP, além de recursos e facilidades adicionais à construção de objetos CORBA, e por atender ao recente Mapeamento Padrão de IDL para Java [OMG97a] — o qual garante a portabilidade de *Stubs* e *Skeletons* —, esse produto se mostrou adequado para os propósitos de implementação do protótipo do SGPOM.

2.4 Considerações Finais

Ante o que foi citado até aqui, pode-se concluir que o modelo CORBA fornece um flexível substrato de comunicação e de suporte à construção de serviços dentro de um ambiente distribuído orientado a objetos. Por ser projetada para resolver os

[8] Um *Bean* pode ser definido como um componente reusável (e modular) de *software*, manipulável, visualmente, a partir de uma ferramenta gráfica de desenvolvimento, podendo ser, opcionalmente, embutido em qualquer aplicação [Flan97].

problemas de interoperabilidade e heterogeneidade de aplicações (distribuídas), CORBA se apresenta como uma tecnologia robusta de integração de componentes (programas, objetos, funções, bases de dados...) que ultrapassa os limites impostos por algumas fronteiras conhecidas [Bake97] [Vino97], tais como:

- **Fronteiras da rede**: um dos objetivos basais para CORBA é prover um acesso transparente aos Objetos. Basicamente, isso significa que um programador pode lançar mão de um Objeto sem precisar saber onde este está localizado na rede. Ou seja, os detalhes relativos à posição dos Objetos ficam escondidos, evitando-se, pois, a interferência nos contratos de serviço os quais se propõe oferecer ao ambiente. Um Objeto pode "existir" em qualquer lugar na rede, mas aparenta ser local a quem o invoca. Essa transparência, conhecida como *Transparência de Rede*, absorve duas outras, propostas pelo RM-ODP: Transparência de Localização e Transparência de Acesso.
- **Fronteiras da linguagem de implementação e do Sistema Operacional**: o uso da IDL para definir as interfaces dos Objetos permite que essas interfaces possam ser usadas por um grande número de linguagens de programação (C, C++, Smalltalk, Ada, Java, Cobol) por meio de diferentes plataformas (desde PCs até estações de trabalho, passando inclusive por *mainframes*). Assim, é possível que um mesmo serviço (provido por um único tipo de interface) possa ser implementado em linguagens diferentes, o que ficará totalmente transparente para quem o requisita.
- **Fronteiras do modelo de objetos**: embora o Paradigma de Orientação a Objetos venha se tornando um padrão *de facto* para desenvolvimento em escala de sistemas, há ainda diferenças entre os modelos de objeto em uso. Como exemplo, os modelos de programação de Java e C++ apresentam pequenas disparidades que impedem a integração entre códigos implementados nas duas linguagens. O Modelo de Objetos e o Modelo de Referência propostos pela OMA definem as regras para a interação entre objetos CORBA, garantindo que essa interação seja independente de protocolos, aspectos de implementação e mecanismos de rede. Aplicações desenvolvidas conforme essa filosofia abstraem-se dos detalhes de rede (comunicação), podendo ser empregadas em ambientes de programação diferenciados.
- **Fronteiras do legado**: o que se traduz, sob os auspícios do OMG, em se possibilitar mecanismos de integração do novo com o já existente. Como CORBA não entra em detalhes de implementação, um bem projetado ORB não requer que componentes e tecnologias já em uso sejam abandonados. Pelo contrário, a especificação é flexível o suficiente para garantir a incorporação e integração de protocolos e padrões (como DCE — *Distributed Computing Environment* — e DCOM — *Distributed Component Object Model*) já consolidados.
- **Fronteiras de paradigmas**: paradigmas de projeto e programação, muitas vezes discrepantes, podem gerar soluções contendo conjuntos de componentes a serem implementados em diferentes linguagens ou, talvez, sob considerações e regras distintas. Essas diferenças podem ser ocultadas por uma interface IDL.
- **Fronteiras de venda**: a própria arquitetura CORBA e aplicações construídas sobre ela são melhor projetadas mediante os princípios de orientação a objetos. O fato de que as interfaces dos Objetos devam ser definidas em IDL ajuda aos projetistas e programadores a pensarem em termos de componentes reusáveis que interagem entre si, abreviando os seus esforços e garantindo uma maior distribuição de funcionalidades em aplicações de porte. Isso garante, como efeito paralelo, um padrão de concordância entre especificações de produtos lançados por fabricantes

distintos, da mesma forma que assegura a sua manutenção por um período mais duradouro.

Uma breve discussão sobre alguns dos esforços voltados para a caracterização de sistemas multimídia orientados a objetos distribuídos, os quais se servem das especificações padronizadas para RM-ODP e CORBA, será introduzida no final do próximo capítulo, após serem analisados os requisitos impostos pela classe de objetos multimídia.

Capítulo 3

Objetos Multimídia

A meta deste capítulo é a de se apresentar uma análise concisa dos fatores que influenciam, essencialmente, a representação, a recuperação e a armazenagem de Objetos Multimídia (OMs). Nesse sentido, uma revisão bibliográfica de apoio foi realizada, cobrindo algumas das contribuições mais recentes dadas ao estado-da-arte. Dentre estas, as que mereceram maior atenção são aquelas relacionadas à definição, construção e implementação de Bancos de Dados Multimídia (BDMMs) e de servidores de repositórios multimídia.

Este capítulo está organizado da seguinte forma. Num primeiro momento, alguns conceitos serão introduzidos. Em seguida, será conduzida uma avaliação dos tipos de dados multimídia e de seus principais formatos e características. Isso serve de base para se levantar as funcionalidades a serem fornecidas por um SGBDMM modelado como um SGBDOO estendido. Posteriormente, um guia de recomendações para a troca de dados multimídia, fruto dos trabalhos da IMA, servirá de alvo para uma breve discussão sobre um modelo de estruturação de dados multimídia fundamentado no conceito de *container* de objetos. Um enfoque maior será dado sobre uma lista de requisitos a serem satisfeitos quando da manipulação de objetos multimídia por um BDMM (ou repositório multimídia). Em última instância de contribuição, será analisada uma ferramenta de suporte à criação de apresentações multimídia provida pela plataforma Java e serão levantados alguns dos esforços empreendidos no âmbito de objetos multimídia distribuídos.

3.1 Conceitos

Os métodos empregados para a elevação da qualidade do intercâmbio de informações entre as pessoas eram, antes do uso maciço do computador, bastante comprometidos, devido às limitações impostas pelas mídias comumente utilizadas. A

informação era ordenada em estruturas estáticas (livros, capítulos, etc.) de difícil modificação e atualização. Com os avanços tecnológicos alcançados frente aos dispositivos de armazenagem, vídeo, memória e processamento, tornou-se factível explorar novos meios de estruturação e organização dos dados que fugissem a essa rigidez, dando espaço para o surgimento da Multimídia.

A Multimídia digital pode ser definida como o campo interdisciplinar de estudos voltado para a construção de aplicações que exploram a natureza multissensorial (ou *multimodal*) do ser humano e a capacidade computacional de armazenar, manipular e conduzir informações numéricas e não-numéricas de gêneros variados — como texto, áudio, vídeo, imagens e animações — sob o mesmo formato de representação binária [Fluc95]. Desse modo, tais aplicações passam a fixar novas restrições a serem satisfeitas pelo sistema de suporte ou plataforma computacional. Para o seu uso em larga escala, um maior amadurecimento das tecnologias de *hardware* e *software* passa a ser fundamental, e isso só é atingido à medida que se evoluem os modelos de armazenagem, recuperação, gerência, organização e representação dos dados.

Algumas mídias (como áudio e vídeo) são classificadas como *contínuas*, *dinâmicas* ou *temporais*, por consistirem de uma seqüência de unidades, chamadas de *quanta* (amostras e quadros, respectivamente), que só trazem significado quando apresentadas no instante de tempo apropriado [Gemm95]. Outras (como texto e imagens) envolvem apenas componentes de estruturação espacial, sendo, por isso, classificadas como *discretas* ou *estáticas*. Documentos multimídia, por sua vez, definem estruturas lógicas organizadas como uma composição de dados encapsulados em objetos, onde cada um destes pode ser de um tipo de mídia diferente (numérica ou contínua).

As aplicações multimídia apresentam diferentes desafios a serem enfrentados. Os seus requisitos e os tipos de dados multimídia que necessitam manipular podem determinar quais funcionalidades um BDMM deve oferecer. Com aplicações de vídeo sob demanda (*VOD*), usuários podem acessar informações, sob a forma de vídeo, a partir de um ou mais servidores remotos, seguindo um estilo de interação unidirecional. Nesse caso, não é interessante que se dê suporte à edição, captura e modelagem de tipos de mídias simples, bem como de suas composições (documentos multimídia). Ao mesmo tempo que isso é verdade, aplicações VOD necessitam de mecanismos que garantam alto desempenho para as tarefas de armazenagem e transmissão dos dados, já que estes apresentam grandes volumes. Por outro lado, as aplicações de *vídeo-conferência* enfocam sobre aspectos de tempo-real relacionados à comunicação e sincronização de fluxos de áudio e vídeo. A sincronização é inflexível, haja vista que se deve reproduzir, da maneira mais fiel possível, os fluxos de dados produzidos em nós remotos. Não se necessita, nesse contexto, de suporte à armazenagem de dados. Finalmente, aplicações voltadas a tarefas de treinamento e aprendizagem mediadas por computador (tais como as enciclopédias digitais) requerem técnicas complexas de modelagem e apresentação, as quais garantam o suporte à interatividade em ambientes multi-usuários.

Por ter um caráter variado, a informação multimídia apresenta uma estrutura irregular, podendo ser tão simples quanto um único tipo de dados a tão complexa quanto um documento multimídia. Dentro dessa perspectiva, Sistemas de Informação

tradicionais precisam ser incrementados com novos recursos para tratamento dessas características. Comumente, tais sistemas definem uma camada de gerência que isola a aplicação dos detalhes de armazenagem dos dados, de modo a garantir a sua independência e o seu compartilhamento. Uma classe desses sistemas é formada pelos Sistemas de Gerência de Banco de Dados (SGBDs), os quais se constituem em coleções de dados inter-relacionados (na forma de bases de dados) controladas por um conjunto de programas responsáveis pela definição, criação, armazenagem, gerência e consulta a tais dados. Seguindo esse raciocínio, uma extensão natural a esses conceitos nos leva aos *Sistemas de Informação Multimídia* [Chri96] e aos *SGBDMMs* [Adje97].

Dentro do contexto de SGBDs, quando se alude a um repositório, passa-se a compreender um componente do sistema responsável por simples tarefas de gerência dos dados, tais como segurança e *backup*. Repositórios não precisam reconhecer os formatos dos dados que armazenam, caso não operem sobre eles. Suporte a *transações* é possível, mas operações de atualização requerem a substituição do objeto como um todo. No caso dos repositórios multimídia, estes não estão a par das restrições de sincronização inerentes aos dados, já que são projetados para simplesmente os enviar aos clientes (aplicações) sem nenhum tratamento *a priori*. Em uma publicação recente [Paza97], são apresentados, dentre outros conceitos, dois tipos de repositórios multimídia interessantes para o escopo do protótipo "Banco de Dados Multiware":

- **Pseudo-repositório**: este tipo de repositório contém *metadados* multimídia, na forma de nomes, tamanhos, codificações, descrições e palavras-chave. Valores adicionais descrevem o caminho lógico para os OMs, localizados fora do repositório e dentro de um sistema de arquivos local ou remoto. Esse último fato faz com que o repositório tenha um controle limitado dos dados, não podendo garantir, por exemplo, que não sejam corrompidos quando acessados por outras entidades.
- **Repositório simples**: um repositório deste gênero pode prover acesso restrito a dados, garantindo um *backup* mais eficiente. Este tipo de repositório gerencia os dados, armazenando-os em discos secundários locais ou em dispositivos terciários, tais como discos ópticos. O uso desses repositórios está centrado basicamente em um cenário onde aplicações recuperam os dados multimídia, modificam-nos e depois os retornam para uma nova armazenagem.

3.2 Natureza dos dados multimídia

A Multimídia surgiu como uma tentativa de diminuir o *gap* semântico entre as aplicações e seus usuários, de modo a simplificar e a tornar universais comandos para a obtenção das tarefas desejadas, buscando, com isso, um processo de interação mais natural [Viei95]. Para tanto, a informação é conduzida por diferentes meios (mídias), a fim de que se aumente o seu poder de expressão. Porém, cada tipo de mídia apresenta propriedades particulares a serem eficientemente tratadas pelos SGBDs. Os tipos de mídia (e suas características inerentes) comumente encontrados num BDMM incluem [Fluc95] [Adje97] [Paza97]:

- **Textos**: os quais apresentam uma estrutura lógica e de *layout* que deve ser preservada na apresentação e que serve como fonte de identificação no processo de busca.

- **Gráficos**: que incluem rascunhos, desenhos, ilustrações e objetos 3D. Este tipo de dados pode ser armazenado de modo particionado num banco de dados, garantindo que seu conteúdo seja facilmente recuperado a partir de consultas feitas sobre os seus metadados de estruturação (tais como linhas, arcos e círculos).
- **Imagens**: que incluem figuras e fotografias codificadas sob padrões definidos no mercado, como BMP, JPEG e GIF [Cost98]. Ao contrário dos gráficos, as imagens são armazenadas sem o uso de metadados, haja vista que aqui não existe o conceito bem definido de arcos, linhas e círculos. Porquanto alguns desses formatos de padronização (JPEG, JBIG, GIF) usam técnicas e algoritmos de compressão baseados em redundância para reduzir o volume dos dados a serem armazenados.
- **Animações**: que se constituem em seqüências temporais voltadas à apresentação ordenada de imagens e gráficos gerados de forma independente. Ao contrário de simples imagens, as quais podem ser recuperadas e apresentadas durante qualquer intervalo de tempo, as animações têm restrições temporais a serem satisfeitas, pois o seu conteúdo conduz a uma interpretação cujo significado é dependente da sua taxa de apresentação.
- **Áudio**: como as animações, representa uma seqüência de componentes independentes orquestrados temporalmente. Se cada componente é representado pelo uso de um descritor, tais como nota, tom ou duração, tem-se o que se chama de *áudio estruturado*. Dados de áudio em um aplicativo multimídia apresentam a característica da continuidade no tempo; por essa razão, são considerados como de mídia contínua. Um fluxo (*stream*) de áudio pode ser manipulado por operações estáticas de edição, tais como recortar, copiar e colar. Porém, operações de reprodução (*playback*) e de gravação (*recording*) estão sempre associadas a uma escala de tempo. Operações de busca, tais como encontrar uma palavra em um documento falado, podem ser definidas sobre dados de áudio. Para tanto, há a necessidade de se ter algum meio de abstração que desempenhe um papel similar às técnicas tradicionais de indexação usadas em operações de busca a dados convencionais. Da mesma forma que para as imagens e animações, outra característica desse tipo de mídia é que dados de áudio ocupam quantidades significativas de espaço de armazenagem, acarretando no uso de técnicas de compressão.
- **Vídeo**: que se define como um conjunto em seqüência de dados fotográficos, representando algum evento real capturado por um dispositivo especializado (como, por exemplo, uma câmera digital de vídeo). Os dados são divididos em quadros (*frames*), os quais representam unidades semânticas básicas de edição correspondentes a simples imagens fotográficas capturadas a taxas que variam entre 24 e 30 fps (*frames per second*).
- **Multimídia composta**: criada pela combinação dos tipos básicos de mídia apresentados acima, os quais podem estar física ou logicamente entrelaçados. O entrelaçamento físico leva à formação de um novo tipo sob um outro formato de armazenagem; o lógico define um novo tipo, todavia preserva os formatos individuais dos tipos básicos. Como exemplo do segundo, um novo tipo AV (áudio-vídeo) seria composto por duas partes distintas, as quais, quando da sua apresentação, deveriam ser sincronizadas a fim de manterem as suas dependências temporais — o que caracterizaria a percepção do novo tipo pelo usuário. O padrão AVI segue esse modelo.
- **Apresentações**: que compreendem opulentos agregados, sob a forma de orquestrações de dados multimídia feitas por intermédio de roteiros (*scripts*)

temporais, estes ajustados aos propósitos de modificação e apresentação. Os documentos multimídia constituem-se em um bom exemplo desta classe.

Para a representação desses tipos de mídia, formatos divergentes apareceram como resultado. Tais formatos estão associados a algoritmos que tratam de particularidades específicas de cada mídia, ou seja, são independentes, e portanto não apresentam mecanismos de interoperação. Como decorrência, a busca de padrões para transferência e codificação de documentos e mensagens multimídia (compostos por dados sob diversos formatos), dentro do âmbito da *Internet*, levou à especificação do MIME (*Multipurpose Internet Mailing Extensions*) [Bore93]. Esse protocolo de representação oferece uma maneira padronizada, robusta, aberta e extensível de suporte à organização e ao transporte de mensagens multimídia complexas.

A abordagem de codificação trazida com o MIME é mais robusta quando cotejada a aquelas anteriormente existentes (como o *uuencode* e o *BinHex*), as quais não são totalmente imunes à corrupção dos dados. Mais ainda, o MIME provê uma descrição estrutural de documentos complexos, suportada por esquemas portáveis de codificação, além de garantir uma maior automação para com operações básicas, tais como a de conversão e compressão de múltiplos arquivos sob uma árvore de diretório em uma só unidade (técnica conhecida como *Archiving*).

O padrão MIME procura prover um mecanismo aberto que cubra tanto os tipos de arquivos (e mídias) já existentes, como aqueles a serem especificados futuramente. Para tanto, associa-se a um modelo hierárquico baseado em conjuntos de tipos e subtipos vinculados. A proposta inicial contém sete tipos MIME, cada qual relacionado a alguns subtipos (dos quais exemplos típicos são dados na Tabela 3.1 [Cost98]).

3.3 Banco de Dados Multimídia

Mídias que requerem grandes volumes para sua armazenagem (como áudio e vídeo) podem ser introduzidas em um sistema multimídia mediante métodos de integração de tipos de dados contínuos a linguagens de programação e a modelos de dados, trazendo, como conseqüência, a necessidade de uma gerência eficiente desses dados. Muitas aplicações ainda usam sistemas de arquivos para armazenar dados multimídia, embora se saiba que tanto eles como os SGBDs tradicionais não são apropriados às tarefas de organização, manipulação e gerência. Os conceitos de SGBDs, todavia, não devem ser descartados; devem, sim, ser reavaliados, pois parecem ser adequados para o tratamento uniforme de dados complexos. Tais sistemas provêem facilidades que garantem integridade, proteção e consistência aos dados, bem como um controle de concorrência no seu acesso por múltiplos usuários [Boll96b].

Tipo/subtipo MIME		Descrição
text		*Textos usando o conjunto ASCII básico ou uma de suas extensões padronizadas*
	text/plain	*Texto sem anotações de formatação*
	text/richtext	*Texto com anotações (textuais) de formatação (RTF)*
image		*Arquivos binários contendo imagens estáticas*
	image/JPEG	*Dados de imagem sob o formato de compressão JPEG*
	image/GIF	*Dados de imagem sob o formato de compressão GIF*
audio		*Arquivos binários contendo dados de áudio ou voz*
	audio/basic	*Dados de áudio armazenados sob o formato básico*
video		*Arquivos binários contendo dados de imagem em movimento (incluindo, possivelmente, trilhas de áudio)*
	video/mpeg	*Dados de vídeo armazenados sob o formato de compressão MPEG*
application		*Há duas principais categorias de subtipos de aplicações: (1) arquivos contendo aplicações ou dados binários não especificados; e (2) arquivos, usualmente binários, que podem ser interpretados por aplicações específicas*
	application/octet-stream	*Um fluxo de dados binários não especificados, tais como um arquivo binário geral ou inerente a uma aplicação*
	application/postscript	*Um arquivo Postscript*
	application/pdf	*Uma arquivo PDF*
message		*Para encapsular uma outra mensagem*
multipart		*Para mensagens complexas consistindo de vários outros tipos simples*
	multipart/mixed	*Uma mensagem complexa contendo tipos de dados diferentes*
	multipart/alternative	*Conjunto de diferentes representações dos mesmos dados — usuários (ou ferramentas) podem escolher o formato mais adequado para seus propósitos*
	multipart/parallel	*Um conjunto de dados diferentes solicitados para serem apresentados em paralelo, tais como dois vídeos mostrando a mesma ação sob ângulos diferentes*

***Tabela 3.1**: Descrição dos sete conjuntos de tipos e subtipos básicos do padrão MIME.*

Um SGBDMM pode ser definido como um sistema de *software* que gerencia uma coleção de dados multimídia (localizados em um ou mais BDMMs) e provê suporte aos usuários nas tarefas de consulta, armazenagem e recuperação de objetos multimídia. O projeto de um SGBDMM deve se beneficiar de outros realizados em áreas correlatas, as quais já providenciam suporte a operações multiusuário (controle de concorrência), independência (abstração) de dados, tolerância a falhas (transação e recuperação), neutralidade a plataformas (abertura), consultas flexíveis (acesso aos dados) e segurança (controle de acesso).

A modelagem de dados contínuos pode ser realizada de diferentes modos. Em modelos de bases de dados tradicionais, a solução encontrada foi a de inserir BLOBs (*Binary Large Objects*) como forma de armazenar grandes volumes de dados. Tais entidades são geralmente implementadas como ponteiros em um campo de registro de uma base de dados relacional e podem encapsular volumosos objetos binários vistos como simples unidades. Essa abordagem, porém, apresenta limitações (discutidas na Seção 3.5), pelo fato dos dados serem acumulados sem interpretação do seu conteúdo. Isso afeta sobremaneira a propriedade de *independência dos dados* para com as aplicações, a ser oferecida por um BDMM, bem como a manutenção de mídias interativas [Nara96].

Por sua vez, SGBDOOs parecem formar uma base adequada à construção de BDMMs. Com esses sistemas, pode-se modelar dados contínuos por meio de tipos de

dados abstratos [Gann96]. Ou seja, podem ser definidas classes com atributos e métodos para a representação de fluxos de dados. Isso possibilita a captura da semântica específica associada a cada tipo de dado não convencional. Por exemplo, uma classe poderia, explicitamente, refletir a estrutura de um vídeo codificado sob o formato MPEG via atributos de classificação dos quadros (em I, B e P) e operações de corte das cenas. Embora isso permita um suporte significativo ao processamento de mídias contínuas, ainda não atende a requisitos de gerência dos objetos, haja vista ser esse tipo de modelagem apenas uma aplicação do modelo de dados.

Em BDMMs baseados em objetos, instâncias dos tipos de mídia são chamados de *objetos de mídia*[1]. Um objeto de mídia representa uma mídia simples, por exemplo, um fluxo de vídeo ou de áudio. Composições multimídia podem ser construídas a partir de várias dessas unidades, as quais possuem relações espaciais e temporais entre si a serem cumpridas durante a fase de apresentação. Objetos de mídia, portanto, são blocos básicos para a composição de apresentações e documentos multimídia.

Dentro desse cenário, as funções de um BDMM passam a estender aquelas introduzidas por um SGBDOO convencional, envolvendo [Adje97]:

- **Integração**: que assegura que os itens de dados não sejam duplicados entre as diferentes invocações.
- **Independência dos dados**: que permite que mudanças internas às bases de dados e funções de gerência não sejam propagadas às aplicações.
- **Controle de concorrência**: que garante a consistência dos dados multimídia por meio de regras ordenadas de execução em transações concorrentes.
- **Persistência**: que se traduz na habilidade dos objetos de dados de persistirem a diferentes transações e invocações.
- **Privacidade**: que restringe acesso não-autorizado para modificação ou leitura dos dados.
- **Recuperação**: que garante que os dados persistentes não sejam afetados por eventuais falhas durante as transações.
- **Suporte a consultas**: que assegura a existência de mecanismos apropriados de realização de *queries* voltadas à recuperação de dados multimídia.
- **Controle de versões**: que permite a coexistência de várias versões de objetos persistentes.

No suporte ao controle de concorrência, o conceito de transação é vital. Uma transação define uma seqüência de instruções a ser executada ou abortada completamente. No segundo caso, a base de dados restaura o seu estado anterior. Para os propósitos da Multimídia, um dos desafios é o de se definir os níveis de granularidade de concorrência apropriados. Em sistemas tradicionais (tais como os SGBDRs), um simples registro ou tabela serve como unidade de concorrência. Todavia, tais sistemas são projetados para atenderem aos requisitos de recuperação de estruturas simples e repetitivas. Como os documentos multimídia possuem estruturas lógicas e de *layout* irregulares [Karm95], além de tamanhos significativos, uma nova abstração deve ser oferecida. No caso dos BDMMs baseados em SGBDOOs, a unidade lógica de acesso passa a ser um simples (ou complexo) objeto de mídia, garantindo, com isso, maior flexibilidade.

---

[1] No escopo deste documento, Objetos Multimídia (OMs) correspondem a desde objetos de mídia simples até objetos complexos, tais como os documentos multimídia.

O suporte à persistência de dados multimídia compreende um conjunto de métodos que possibilita a armazenagem de arquivos multimídia em repositórios ou sistemas de arquivos dedicados. Tal abordagem deve lidar com altos volumes de dados, característica intrínseca aos tipos de mídia contínua, além de aspectos de representação e armazenagem de metadados e de composição dos objetos multimídia. Isso acarreta no suporte adequado à *persistência seletiva* dos dados: para evitar sobrecargas de processamento e de armazenagem, os dados (objetos) devem estar sendo constantemente classificados em persistentes ou transientes, de acordo com o seu período de existência.

O suporte a controle de versões se torna importante quando um objeto multimídia persistente é acessado por várias aplicações, sendo atualizado ou modificado freqüentemente. Entretanto, os grandes volumes de dados restringem o uso não-controlado e indiscriminado dessas versões, principalmente quando da manipulação de objetos complexos.

Pazandak e Srivastava [Paza97] mostram, numa tabela comparativa, o quadro atual formado por produtos e protótipos de SGBDOOs, colocando-os sob uma análise qualitativa que os classifica conforme a sua adequação ao tratamento de dados multimídia. Consoante essa tabela, a maioria das implementações ainda não suporta as funcionalidades necessárias, já que se restringem a oferecer um conjunto básico de classes com opções limitadas para armazenagem de imagens e trechos de áudio e vídeo.

3.4 Prática Recomendada para a troca de dados multimídia

A intenção, aqui, é a de se discorrer, brevemente, sobre um (dentre outros) modelo padrão dedicado à estruturação de OMs complexos.

3.4.1 Apresentação

"A troca de dados multimídia leva, freqüentemente, a complexas transações de múltiplos formatos de dados, o que aumenta a complexidade das ferramentas de suporte, os custos e o tempo de desenvolvimento, além de limitar o mercado de títulos e conteúdo multimídia". Sob essa premissa, e a fim de reduzir essas barreiras, a IMA (*Interactive Multimedia Association*) traçou um conjunto de métodos para a troca de dados multimídia entre plataformas heterogêneas, recomendado para produtores de ferramentas, de títulos e de conteúdo multimídia. O resultado foi a Prática de Recomendação IMA950701.1 [IMA95], a qual define um formato flexível de *container* (recipiente ou encapsulador) como unidade de troca de dados multimídia. Define também *composições* para expressar as relações entre os objetos contidos nesse *container* e destes com o tempo, além de *descrições fonte* para expressar as relações dos dados com fontes digitais e analógicas. Para tanto, são apresentados propriedades fundamentais e tipos de dados que provêem a base para o intercâmbio destes.

Tal Recomendação é um documento composto por três partes, a saber:

1. **Fundamentos da Prática de Recomendação**: uma sinopse do documento, dando seu escopo, justificação e explanação dos conceitos-chave.
2. **Formato do *container***: descrição de um formato para um encapsulador portável de dados, o qual contém objetos de dados multimídia. Tal formato de *container* é uma abstração chamada de Bento [Harr93] e foi especificado para armazenar o maior número possível de tipos de dados.
3. **Composições**: relato de composições que expressam as relações entre objetos multimídia contidos num *container* e definição de descrições que fornecem referências a fontes analógicas e digitais de origem ou destino dos dados dos objetos multimídia. Essas composições e descrições são providas via um modelo de objetos conhecido como OMFI — *Open Media Framework Interchange*.

Em particular, quando se faz alusão ao formato de *container*, quer-se referir a um gênero de arquivo composto por diferentes tipos de dados, estes sob formatos divergentes. Isso significa que um simples arquivo Bento pode ter múltiplas partes funcionais (por exemplo, uma parte de texto e outra de desenho) e cada uma dessas partes pode ter distintas versões das mesmas interfaces para diferentes plataformas (por exemplo, para PCs ou para estações de trabalho).

Já o modelo de objetos para troca de composições entre plataformas heterogêneas, projetado como OMFI, o qual não será discutido aqui, define uma estrutura para englobar todas as informações requeridas para o transporte de uma variedade de mídias digitais, tais como áudio, vídeo, gráficos, imagens, bem como das regras de combinação e apresentação dessas mídias. Por seu intermédio, pode-se representar informações concernentes aos atributos físicos que descrevem um dado objeto de mídia, guardando-se as condições em que se deu a sua captura ou geração.

3.4.2 Considerações sobre o Formato do Recipiente (*Container*)

Na segunda parte do documento [IMA95], a qual descreve o formato do Bento, são identificados os componentes que compõem esse recipiente, cuja função é a de integrar numa só unidade um ou mais objetos relacionados a um ou mais formatos de dados. A idéia que está por trás é a de possibilitar que aplicações inteiras (compostas por vários objetos de dados) possam ser inseridas numa espécie de invólucro, homogêneo e acessível a outras aplicações por meio de um conjunto bem definido de interfaces. Para tanto, são introduzidos conceitos, os quais serão sucintamente discutidos nesta subseção. O interesse é o de se conseguir caracterizar a estrutura de um OM, a partir do modelo desse *container*.

A Figura 3.1 traz os elementos mais importantes que particularizam o formato de recipiente em questão. Seguindo a abordagem proposta, todo objeto deve estar associado (contido) a um *container*. Mais ainda, cada objeto pode consistir de um conjunto de *propriedades*, sendo que uma propriedade define o papel, a função de um grupo de *valores* com *tipos* distintos dentro do *container*. Por exemplo, uma simples propriedade relativa a datas poderia ter associada a si valores representados por uma *string*, uma data segundo o calendário ocidental (Juliano), bem como uma data seguindo o padrão OSI. Tanto as propriedades como os valores não necessitam estar em uma ordem particular, sendo estes últimos compostos por seqüências variáveis de

zero ou mais *bytes*. Um valor não pode ter mais de uma propriedade e representa um dado real de um objeto.

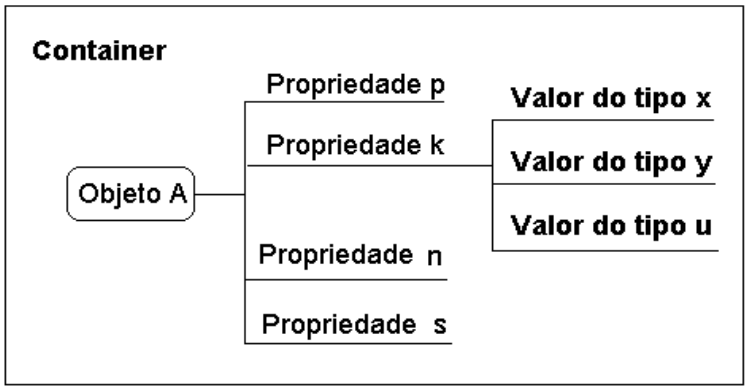

***Figura 3.1**: Elementos de um container Bento.*

Um *container* pode estar inserido em outro; sendo esse o caso, deve-se ter um valor dentro do último que represente o primeiro. Cada objeto contém pelo menos um identificador (ID), o qual é único dentro do *container* que o envolve, e não deve possuir informações além daquelas armazenadas em suas propriedades. Referências a objetos são valores persistentes que devem ser armazenados como dados do usuário. Tais referências permitem que um objeto se refira a outro via o seu identificador.

Outro conceito importante introduzido é o de *tipos* estruturados. Um tipo define o formato de um valor, podendo informar sobre a sua estrutura, se ele está comprimido ou não, e sobre a sua ordem de *bytes*. Tipos estruturados, por sua vez, permitem a ocorrência de transformações entre representações diferentes. Usando esse recurso, é possível se ter múltiplos tipos associados a um só valor, todos eles independentes. Sendo o caso, o tipo visível para uma aplicação passa a ser aquele que codifica o formato dos dados sob o ponto de vista mais apropriado.

O modelo desse *container*, como dito anteriormente, serve-nos como base para a compreensão de como é a estrutura de um OM. A lista de características e requisitos dada a seguir é fruto da compreensão desse modelo e de ponderações acenadas em outras referências.

3.5 Caracterização de Objetos Multimídia

Nesta seção, serão apresentados alguns dos fatores que identificam e, de certo modo, particularizam a classe de Objetos Multimídia. O interesse é o de se caracterizar tais objetos sob três pontos de vista:

- **representação**;
- **recuperação**; e
- **armazenagem**.

3.5.1 Fatores que influenciam a representação e a apresentação

Os seguintes fatores devem ser alvo de consideração para a representação e a apresentação apropriadas de OMs em um BDMM:

Modelo de dados

Os dados multimídia são de natureza audiovisual. Por isso, uma representação acurada e adequada do seu conteúdo será sempre algo muito difícil de se obter. Porquanto os dados armazenados em um BDMM deverão ser uma representação aproximada (e pobre) dos verdadeiros objetos.

O Modelo de dados é um dos principais pontos dentro do projeto de desenvolvimento de qualquer SGBD e assim o é para os SGBDMMs. A sua função é a de isolar dos usuários dos dados os detalhes de gerência de dispositivos e de estruturas de armazenagem. Modelos de dados multimídia devem capturar propriedades estáticas e dinâmicas associadas ao conteúdo das bases de dados multimídia. As propriedades estáticas incluiriam os objetos que fazem parte do conjunto de dados, as suas relações lógicas e seus atributos. Já as dinâmicas incluiriam as interações entre os objetos, bem como as operações possíveis de serem efetuadas sobre eles. Nesse âmbito, Nwosu e Thuraisingham [Nwos94] descrevem as nuanças envolvidas na extensão de um modelo de dados orientado a objetos para aplicações que gerenciam dados multimídia. Na ocasião, eles endereçam uma nova perspectiva de modelagem baseada naquela proposta pelo OMT [Rumb91].

Um modelo de dados trata de aspectos de representação de objetos multimídia, definindo, mediante abstrações de mais alto nível, operações possíveis de serem realizadas sobre eles. Essas operações podem incluir seleção, inserção, edição (*clipping*, *cutting*, *pasting*...), indexação e navegação (*browsing*).

Sob esse aspecto, tal modelo deve possuir o que Narasimhalu [Nara96] se refere como um adequado *poder de expressividade*, este definido como a habilidade para se expressar as muitas variedades existentes de objetos multimídia em termos de sua *estrutura*, *comportamento* e *função*. A estrutura incluiria os atributos e conteúdo. O comportamento de um objeto definiria o conjunto de mensagens que ele entende, responde e inicia. Já a função explicitaria o seu papel lógico dentro do mundo de uma base de dados. Nesse âmbito, objetos similares poderiam ser agrupados em classes; cada OM seria um membro *nebuloso*[2] de uma dada classe geral. Isso por não satisfazer todas as propriedades da classe, com a qual teria associada uma certa *função de pertinência* [Wu+95], indicando o seu grau de similaridade para com esta.

Dentro desse contexto, o modelo deve encapsular diferentes tipos de objetos. Num nível mais básico, deve representar tipos de dados fundamentais (tais como imagem, gráfico, texto e áudio). Num patamar mais alto, deve ser capaz de identificar, explicitamente, objetos multimídia complexos com características espaciais e temporais próprias, além de prover suporte à representação de objetos de mídia de tempo-real.

Quanto às relações entre OMs, aquelas que expressam composição, como as de "É-uma" (*Is-A*) e "Parte-de" (*Part-Of*), são bastante interessantes: a segunda captura a

[2] *Fuzzy*.

noção de objetos complexos, enquanto a primeira detém a idéia de hierarquia entre classes. Uma outra relação interessante aqui seria a "Similar-a" (*Similar-To*), a qual permitiria que assertivas do tipo "*A* é similar a *B*" pudessem ser representadas. Isso abre espaço para uma classificação difusa nas hierarquias de classes.

Finalmente, um modelo de dados multimídia deve capturar o mapeamento entre as representações física e lógica de OMs. No caso de um livro, por exemplo, a sua organização lógica seria em termos de seus componentes, tais como capítulos, seções e subseções, ao passo que a sua representação física se reduziria a páginas, parágrafos e frases. Para esse exemplo, um mapeamento possível seria aquele feito por uma tabela comum de conteúdos (índice).

Estrutura e representação complexa da informação multimídia

Como se viu pela abstração de *container* apresentada na seção anterior, esse gênero de informação tem uma estrutura irregular, podendo ser tão simples quanto um único dado a tão complexo quanto um documento formado por diversos tipos de mídia — cada mídia poderia estar associada a um tipo dentro do *container* ou, senão, ao próprio *container*. Essa estrutura deverá ser bem representada (via um modelo de dados conveniente) e bem manipulada (mediante a utilização de um ou mais repositórios dedicados).

Ademais, objetos multimídia são em geral *multidimensionais*. Enquanto as dimensões temporal e espacial do conteúdo podem ser tratadas via uma representação conveniente, aquela de tempo-real impõe requisitos à sua apresentação que tanto influenciam a disposição dos dados em dispositivos de armazenagem especializados, como fixam restrições aos engenhos de recuperação, demandando, por conseguinte, um suporte apropriado a ser dado pela infra-estrutura de comunicação (rede), assim como pelo próprio BDMM.

Objetos multimídia podem ser classificados como *complexos*, se construídos por uma "mistura" de pequenas entidades individuais (um *container* pode encapsular um ou mais congêneres). Objetos complexos são diferentes daqueles considerados como *compostos*. Os componentes destes últimos seriam objetos "fundidos", os quais não poderiam assumir uma identidade própria. No caso de um trecho de filme, o seu fluxo de áudio pode ser considerado como um objeto complexo, integrando diversas trilhas sonoras. Cada uma dessas trilhas pode ser considerada como um simples objeto composto, o qual consiste de fala, música e efeitos especiais, todos eles integrados e inseparáveis.

Sistemas atuais de bases de dados, como analisado anteriormente, provêem um único tipo de representação para tratamento de objetos complexos, os BLOBs. Algumas desvantagens de se usar BLOBs para a armazenagem de informação multimídia podem ser endereçadas:

- Qualquer tipo de dados é armazenado como uma seqüência de *bytes*. Isso proíbe a diferenciação lógica de dados pertencentes a tipos diferentes.
- Nem todas as aplicações usando um BDMM necessitam ter os seus objetos multimídia sob esse formato. Freqüentemente, elas irão necessitar de que apenas partes de um objeto, mas não todo ele, sejam armazenadas dessa maneira.
- Perde-se a informação sobre a estrutura interna dos dados.

Metadados

A *transcodificação*, ou seja, a representação de um objeto em um novo formato a partir de sua forma nativa, é algo interessante para se discutir quando se trata de objetos multimídia. Freqüentemente, tal mudança de representação é adequada para a criação de metadados, isto é, dados que descrevem dados. Diante de uma representação mais pobre (porém realista e objetiva), pode-se traçar o conteúdo desses objetos qualitativamente. Assim, poderiam ser usadas palavras para se reproduzir as características de uma fotografia. Nesse caso, a forma nativa do objeto "fotografia" seria a imagem, enquanto a transcodificada, o texto.

No caso de metadados para Multimídia, deve-se especificar quais são as estratégias otimizadas para a sua geração automática (ou semi-automática) a partir dos dados originais, bem como para a representação da sua estrutura dentro do banco de dados. Por exemplo, para se permitir a consulta a fragmentos específicos de um trecho de vídeo, é preciso que se saiba, dentre outras, as informações sobre os cortes e sobre os objetos contidos nas cenas. Por outro lado, para se testar se dois segmentos de áudio estão relacionados a uma mesma peça musical, é necessário que se defina poderosos operadores de comparação que atuem sobre os metadados relacionados ao tipo "áudio".

De acordo com o meio empregado na utilização de documentos multimídia, deve-se selecionar o conjunto de metadados necessários e a melhor forma de organizá-los. Metadados para representação de tipos de mídia, para descrição e classificação de conteúdo, para composição, descrição histórica e localização de documentos, assim como para fins estatísticos, servem como base representativa desse conjunto [Viei95].

3.5.2 Fatores que influenciam a recuperação

Idealmente, qualquer repositório multimídia ou BDMM deveria suportar múltiplas técnicas de recuperação, a serem usadas de modo independente ou coletivo. Contudo, a recuperação de objetos multimídia é influenciada por certos aspectos de difícil tratamento, tais como sincronização, linguagens de consulta e estratégias de busca e recuperação baseadas em conteúdo, os quais irão ser tratados em seguida.

Sincronização

Aplicações multimídia requerem um conjunto extensivo de mecanismos de sincronização [Blak96]. Além de noções tradicionais para fins de orquestração de eventos, sincronização de tempo-real é necessária para controlar interações entre atividades multimídia separadas, impondo um suporte adequado por parte do SO no atendimento aos requisitos de QoS [Tezu96]. Nesse universo, objetos multimídia de tempo-real são normalmente volumosos, introduzindo restrições quanto às técnicas e estruturas a serem empregadas para o *layout* (organização) e a distribuição dos fluxos de dados pelos discos; isso se agrava quando múltiplas requisições necessitam ser atendidas concorrentemente (caso de aplicações VOD).

Dado que as bases de dados (ou repositórios) devam estar distribuídas pelo ambiente, tais restrições assumem um caráter não mais local, e sim geral. Com isso, impõem-se novas exigências de sincronização entre objetos relacionados dentro de um

SGBDMM. Analisando os requisitos de aplicações multimídia, dois estilos de sincronização de tempo-real podem ser identificados [Coul95]:

- **Sincronização guiada a eventos**: que ocorre quando é necessário iniciar uma ação (tais como apresentar uma legenda) em um sistema distribuído. O intervalo de tempo dessa ação pode corresponder a um ponto lógico de referência, como, por exemplo, um quadro de vídeo particular sendo exibido.
- **Sincronização contínua**: que surge quando se é preciso coordenar os dispositivos de apresentação dos dados, de modo que consumam os fluxos de dados em taxas fixas. Como exemplo clássico, pode-se citar as restrições de sincronização entre os fluxos de áudio e vídeo que compõem um trecho de filme.

Linguagens de consulta

Usuários freqüentemente descrevem as suas consultas de uma forma vaga e subjetiva, assumindo, portanto, um caráter nebuloso. Um indício dessa subjetividade e incerteza seria a descrição de uma face humana com palavras. Uma assertiva tal como a de um "nariz longo" seria passível de diferentes interpretações por usuários distintos.

Uma abordagem convencional e rígida de avaliação das consultas aos repositórios, na maioria das vezes, não reflete a verdadeira intenção ou razão implícita. A causa disso está na dificuldade de representação do raciocínio que motiva a *query*. Pensando-se no contexto multimídia, seria interessante uma linguagem que possibilitasse a descrição dos atributos ambíguos e subjetivos e que passasse a incluir descritores difusos preocupados com o lado qualitativo da consulta.

Pelo fato de a informação multimídia ser difícil de se delinear, é duvidoso que se possa definir uma linguagem padrão abrangente e de peso, tal qual SQL (*Standard Query Language*) o é. É necessário que sejam propostos métodos mais poderosos de processamento de consultas, os quais envolvam medidas embutidas de incerteza (*fuzzy*) e de descrição incompleta.

Estratégias de busca e recuperação baseadas em conteúdo

Em aplicações multimídia, os dados armazenados em repositórios assumem características e padrões diferentes. Seria interessante, por sinal, que se dispusesse de um mecanismo de busca que provesse a recuperação desses dados (voz, vídeo, áudio) de modo mais flexível. Isso possibilitaria ganhos substanciais — principalmente em se tratando de quesitos como velocidade e qualidade dos dados recuperados — a programas e aplicações que realizam constantes acessos a repositórios remotos.

O conteúdo de um OM pode ser representado por meio de dados, características e interpretações. Recuperação baseada em conteúdo [Wu+95] geralmente usa uma característica ou uma interpretação — as quais devem ser, respectivamente, extraída e gerada — quando do acesso a base de dados. Sendo o caso, esta é uma tarefa difícil, haja vista que a compreensão acurada de informações sob a forma de trechos de vídeo e áudio, ou mesmo imagens, nem sempre é possível.

Coleções de OMs podem se tornar muito extensas. Para um processamento rápido de consultas ou para uma possível navegação visual pelos dados, técnicas de *indexação* sofisticadas (tais como indexação baseada em conteúdo) são requeridas. Buscas usando medidas e relações de similaridade seriam interessantes nessa discussão. Uma boa amostra do uso dessa estratégia seria a de se produzir, como resultado da

consulta, uma lista decrescente de valores que, sob algum critério, seriam similares àquele procurado.

3.5.3 Fatores que influenciam a armazenagem

Objetos multimídia são, por natureza, muito complexos e volumosos, o que dificulta a sua adequada manipulação por um banco de dados ou repositório. Como exemplo, uma imagem estática em padrão de alta definição (2048x1024, *true color*) ocupa dezenas de milhões de *bytes*. Um segmento de filme de apenas poucos segundos, sob a mesma configuração, facilmente ocupa *gigabytes* de espaço em disco. Desse modo, esquemas de codificação e compressão, combinados com os de transformação dos dados, são usualmente apresentados como solução, conquanto não definitiva, pois o volume armazenado ainda é grande[3] e a latência de acesso (devida ao tempo de compressão e descompressão), comumente, aumenta. Esses mecanismos atuam na remoção da *redundância* dos dados e, por isso, são passíveis de envolver perdas de informação, as quais, muitas vezes, são toleráveis.

Em virtude das divergentes características existentes entre mídias dinâmicas e estáticas, deve-se decidir pelo uso de instrumentos de armazenagem distintos para cada tipo. No caso de mídias não-contínuas, a sua armazenagem é feita no próprio SGBDOO (vide Figura 1.1), com o uso de metadados para descrição do conteúdo dos arquivos. Por outro lado, os requisitos temporais e de armazenagem intrínsecos a mídias contínuas demandam a construção de repositórios (servidores de dados) dedicados, cujo projeto se torna uma tarefa complexa. Tal servidor deve [Özde95]:

- prover taxas garantidas para armazenagem e recuperação de dados de mídia contínua;
- suportar o acesso concorrente de múltiplos (centenas, ou mesmo milhares) clientes;
- oferecer suporte a operações VCR (*fast-forward*, *rewind*, *pause*...);
- suportar, quando necessário, a recuperação de dados de mídia não-contínua juntamente com a daqueles de mídia contínua;
- gerenciar recursos de armazenagem, tais como memória e discos, a fim de se maximizar a vazão (*throughput*) e se reduzirem os tempos de resposta.

Um servidor de dados multimídia deve lançar mão de uma hierarquia de dispositivos de armazenagem. Dispositivos *online* de alta velocidade (tais como memórias RAM e discos magnéticos) armazenam dados em uso corrente ou freqüentemente acessados, enquanto àqueles *offline*, de baixa velocidade (fitas magnéticas ou discos ópticos), correspondem os dados armazenados permanentemente. A configuração exata do servidor (número de discos e fitas, quantidade de memória) deve ser determinada com base no número de *clips* e segmentos (de áudio e vídeo) e na freqüência de acesso a estes. O desempenho das tarefas do repositório depende da eficiência dos mecanismos de migração empregados na atribuição dos itens de dados multimídia ao nível mais otimizado dentro da hierarquia.

---

[3] Para cem minutos de vídeo comprimidos sob o padrão MPEG-I, por exemplo, é requerido cerca de 1.25 GB de espaço em disco.

Já que um servidor de vídeo é configurado para usar diversos discos, esquemas para a separação adequada de trechos de filme entre esses dispositivos são cruciais para uma distribuição uniforme da carga de trabalho e para a utilização eficiente da largura de banda (*bandwidth*) agregada disponível. Duas técnicas, conhecidas como *data striping* e *data interleaving* [Gemm95], seguem esse preceito e exploram o paralelismo no acesso a blocos de dados dispersos entre vários discos de tecnologia RAID. No primeiro método, o conjunto formado pelos setores físicos de mesma posição de todos os discos do arranjo formam um único setor lógico, onde se encontra todo o trecho de vídeo procurado. Isso força que o acesso aos discos seja sincronizado. Na técnica de entrelaçamento de dados, por sua vez, blocos sucessivos que compõem o arquivo de mídia são armazenados, consoante um padrão determinado, em discos distintos. Isso facilita a recuperação de pequenos trechos contidos no arquivo, haja vista que, nesse caso, os discos podem trabalhar independentemente.

Além disso, políticas especiais de gerenciamento de *buffers* e de controle de admissão dos recursos empregados na armazenagem e recuperação de um filme extenso, ou mesmo de partes dele, devem ser endereçadas. As soluções (algoritmos) propostas podem ser classificadas em abordagens *determinísticas*, *estatísticas* e *baseadas em medidas* [Shen95], de acordo com as estratégias adotadas para se garantir o desempenho requerido. Abordagens determinísticas garantem a confiabilidade dos serviços na medida em que realizam estimativas baseadas em cenários de pior caso, podendo, como efeito colateral, subutilizarem os recursos. Já as estratégias estatísticas apenas garantem a confiabilidade dentro de alguma margem de probabilidade; fenômenos (atrasos) imprevisíveis podem ocorrer devido à contenção de recursos. Abordagens baseadas em medidas alcançam as mais altas taxas de utilização, embora provenham as piores garantias.

Os excessivos volumes e as restrições de apresentação inerentes aos dados de mídia contínua tornam a política de armazenagem de OMs alvo de relevante consideração. Dependendo do nível de granularidade, um OM pode representar uma seqüência de vídeo inteira, um simples quadro (ou imagem), ou apenas um objeto individual pertencente a esse quadro. Quando diversos OMs precisam ser armazenados em um conjunto comum de dispositivos (repositórios) e um número mínimo de requisições concorrentes de acesso deve ser satisfeito, técnicas de decomposição desses objetos em tais repositórios devem ser desenvolvidas, a fim de se aliviar restrições temporais de apresentação e limitações de banda da infra-estrutura de comunicação [Orji96]. Políticas que assegurem decomposições ótimas ainda estão sendo pesquisadas. Nesse contexto, uma estrutura ou serviço que permitisse localizar e gerenciar o acesso aos diversos repositórios dedicados ao tratamento de mídias particulares seria de interesse dentro do ambiente. Isso tornaria factível a recuperação em paralelo de partes de um grande OM (documento multimídia), garantindo as taxas de disponibilidade inerentes a cada mídia. O SGPOM, montado uma camada acima de repositórios destinados a armazenagem desses dados, serviria como exemplo desse serviço.

3.6 Integração entre Java e Multimídia

Desde o seu lançamento (em meados de 1995, ainda com versões beta), a plataforma Java [Gosl96] vem passando por algumas transformações e está em contínua expansão. Para tanto, vem sendo prometido o desenvolvimento de algumas estruturas (pacotes de classes-base adicionais chamados de *Java Frameworks*), que estendem o núcleo sobre o qual aplicações de usuários são construídas, a fim de se atender a diversas áreas de desenvolvimento, desde segurança de dados até comércio eletrônico. Dentre essas *frameworks*, uma é de particular interesse quando se pensa em desenvolver aplicações multimídia, a chamada *Java Media Framework* (JMF).

Tal conjunto de classes adicionais tem o intuito de permitir que desenvolvedores e usuários se beneficiem de uma coleção bastante rica de ferramentas de suporte a mídias interativas para a WWW. O JMF proporciona uma estrutura neutra a plataformas para a apresentação de mídia temporal. O corpo desse *Java Media Framework* oferece, basicamente, as seguintes funções:

- gráficos 2D;
- animação de objetos 2D;
- relógios de sincronização; e
- *players* (reprodutores) de áudio e vídeo.

Além disso, é proposto um conjunto de serviços de mídia que estende o núcleo para novos tipos de mídia, tais como vídeo, MIDI e gráficos (imagens) em três dimensões. Inclui-se, também, uma estrutura de telefonia que irá permitir que aplicações Java façam, recebam, transfiram e controlem chamadas telefônicas com identificação de usuários. O *Java Share Framework* (contido nesse conjunto de novos serviços) deve oferecer funções de conferência em grupo, incluindo uma espécie de *whiteboard* compartilhado (onde um, por vez, escreve e todos, simultaneamente, lêem).

As APIs do JMF são projetadas para suportarem a maior parte dos padrões de conteúdo e de tipo existentes, incluindo MPEG-1, MPEG-2, AVI e MIDI. Além disso, provêem uma abstração que esconde os detalhes de implementação do programador ou projetista de aplicações. Com o JMF, pode-se sincronizar e exibir mídias contínuas produzidas em fontes distintas.

Como exemplo ilustrativo, um documento recente [Sun97b] traz uma primeira proposta dentro dessa estrutura: os *Java Media Players* (JMPs). Um *player* é uma máquina de *software* programável, que processa um conjunto de dados com o tempo e que suporta três tipos de uso:

- **Em nível de cliente (usuário comum)**: um programador pode criar e controlar um JMP para qualquer tipo de mídia padrão mediante o uso de um conjunto simples de métodos.
- **Em um nível mais realçado**: onde um programador pode modificar um *player* já existente, a fim de adicionar novas funcionalidades. Isso garantiria o reuso das estruturas desses *players*.
- **Em um nível de projeto**: onde agora um programador pode adicionar novos *players* (reprodutores) de suporte a formatos de mídias adicionais (outras mídias ainda não padronizadas).

Consoante a proposta do JMF, os JMPs podem apresentar dados de mídia oriundos de diferentes fontes, tais como arquivos locais ou remotos, ou via *broadcasts*, em consonância com o modelo *Produtor-Consumidor*. O JMF suporta dois tipos de fontes de mídia, envolvendo uma aplicação cliente e outra servidora:

- ***Pull Data-Source***: onde o cliente inicia a transferência dos dados, controlando o(s) fluxo(s) proveniente(s) da(s) fonte(s). Protocolos típicos para este cenário seriam HTTP (*Hypertext Transfer Protocol*) e FILE.
- ***Push Data-Source***: onde o servidor inicia a transferência dos dados e controla o seu fluxo oriundo da fonte que representa. Exemplos deste tipo de fonte incluem mídia de *broadcast* ("para todos"), de *multicast* ("para um grupo") e de vídeo sob demanda (VOD). Para dados de *broadcast*, um protocolo adequado seria o RTP (*Real-time Transport Protocol*), desenvolvido pelo IETF (*Internet Engineering Task Force*).

O grau de controle que um programa-cliente pode oferecer ao usuário depende do tipo de fonte da mídia a ser apresentada. Por exemplo, um arquivo MPEG ou AVI pode ter a sua exibição reposicionada via interação do usuário com uma interface gráfica (*applet*) adequada para tratamento dos eventos. Por outro lado, protocolos de *broadcast* devem oferecer um suporte limitado a esse controle, haja vista que operações VCR não têm sentido nesse cenário.

A proposta inclui que esses *players*, durante a apresentação de uma aplicação de áudio e vídeo, passem por diversos estados, como aqueles mostrados no esboço a seguir (Figura 3.2). A cada estado, o *player* realiza alguma funcionalidade:

- Estado ***Unrealized***: aqui, o *player* é instanciado, mas ainda não reconhece a mídia que tem que manipular.
- Estado ***Realizing***: onde o *player* faz a determinação dos requisitos da aplicação com relação a recursos compartilhados (uma amostra desses recursos seria as caixas de som conectadas a um *host*).
- Estado ***Realized***: aqui, o *player* já tem conhecimento de que recursos necessita e já possui alguma informação sobre o tipo de mídia a ser manipulado.
- Estado ***Prefetching***: o *player* está compondo/preparando a apresentação (*rendering*) da mídia. Durante essa fase, por exemplo, o *player* pode carregar os dados da mídia ou pode ativar recursos exclusivos.
- Estado ***Prefetched***: onde o *player* já está pronto para entrar em execução (embora não o faça ainda).
- Estado ***Started***: aqui o *player* está apresentando os dados da mídia ou se encontra à espera de um tempo em particular (dentro do espaço de tempo da apresentação) para começar a reproduzir.

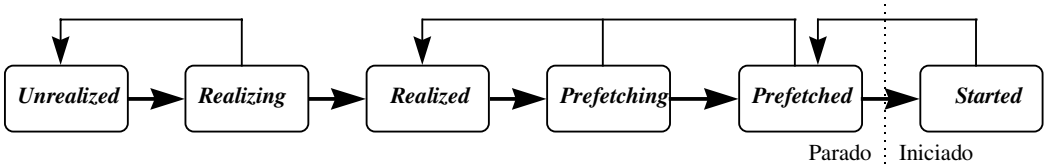

***Figura 3.2**: Estados de um Player.*

Para o projeto do SGPOM, o uso de tal ferramenta é interessante sob o ponto de vista de apresentação. Uma aplicação construída sobre essa estrutura pode manter os seus conjuntos de objetos multimídia (um conjunto para um mídia é associado a um *player*) em um ou mais repositórios especializados, deixando essa incumbência a cargo do serviço de gerência. Assim, quando da exibição dos dados, o acesso (recuperação) a estes poderia ser feito via SGPOM.

3.7 Objetos multimídia distribuídos

Como já destacado, dados multimídia têm estruturas bastante volumosas, o que acarreta na inviabilidade de replicá-los e transmiti-los desmedidamente. Para se garantir o seu uso adequado por várias aplicações, passou-se a pesquisar abordagens alternativas que garantissem a sua distribuição. Estudos na área de *Sistemas Distribuídos Multimídia* surgiram como resposta a essa demanda, introduzindo (ou estendendo) conceitos e modelos e levantando novas necessidades e soluções [Will92]. Nesse enfoque, surgiram propostas de construção de BDMM distribuídos, dentre as quais, a do Banco de Dados Multiware [Toba95] [Rica96]. Esta última, seguindo a filosofia da Plataforma Multiware [Loyo93], descreve um modelo em camadas formado por objetos distribuídos que se comunicam através de uma infra-estrutura de comunicação (Figura 1.1) baseada em um ORB.

A distribuição dos dados multimídia acarreta no surgimento de alguns problemas, os quais vêm sendo apontados e abordados em vários esforços. Dentre esses problemas, pode-se citar:

- **Responsividade**: quando do consumo interativo de dados temporais por vários clientes, problemas de desempenho do substrato de comunicação se tornam críticos. Isso é relevante, particularmente, para repositórios multimídia espalhados em ambientes convencionais que não foram projetados para tratamento adequado de parâmetros de QoS (tais como *throughput*, *delay* e *jitter*).
- **Heterogeneidade**: cada vez maior é a necessidade de se integrar aplicações construídas sobre diferentes paradigmas, abstrações e plataformas.
- **Confiabilidade**: deve-se garantir a disponibilidade contínua dos dados, evitando que se tornem corrompidos. Mecanismos de tolerância a falhas devem ser introduzidos às plataformas e Sistemas Operacionais.

Tendo em mente atender a essa demada, a tendência atual é a de se servir de padrões e especificações típicos de sistemas distribuídos orientados a objetos, como a ANSA, o RM-ODP e a CORBA (estes últimos discutidos no Capítulo 2), passando a estendê-los ou incrementá-los rumo ao tratamento das restrições multimídia. O Grupo de Lancaster (DMRG - *Distributed Multimedia Research Group*) é um dos pioneiros nessa empreitada, realizando trabalhos significativos que norteiam a evolução da área e que servem de baliza para a definição do estado-da-arte.

Numa das primeiras publicações do Grupo [Coul92], investigou-se a adequação da plataforma-modelo ANSA (precursora do RM-ODP) no atendimento aos requisitos multimídia, propondo, como resultado, um conjunto de extensões; as mais relevantes são as que introduziram os conceitos de *stream* (abstração de

protocolos multimídia) e *chain* (interface de controle genérica para dispositivos multimídia).

Coulson et al. [Coul95], por outro lado, apresentam alguns dos requisitos de tempo-real de mídias contínuas mais afetados pelos aspectos de distribuição dos dados e das aplicações, mostrando o seu impacto sobre os pontos de vista computacional e de engenharia do RM-ODP. Nessa frente, os autores sustentam que as especificações de tais pontos de vista não suportam adequadamente essas restrições e passam a sugerir um certo número de extensões mínimas ao modelo. Com relação ao suporte a mídias contínuas, são propostos uma representação explícita de fluxos contínuos, assim como mecanismos de reserva de recursos. Por outro lado, com respeito à sincronização em tempo-real, novas funcionalidades devem ser introduzidas ao Sistema Operacional ou ao subsistema de comunicação [Ferr96]. Nesse sentido, os principais conceitos apresentados para serem adicionados ao ponto de vista de computação são interfaces contínuas (as quais impõem restrições temporais às invocações feitas), núcleos (*kernels*) reativos (os quais implementam *bindings* configurados com características de QoS) e anotações de QoS nas interfaces. Para o ponto de vista de engenharia, são propostos conceitos de suporte a fluxos, gerenciadores de recursos e pilhas de protocolos estendidas para tratamento de QoS.

No tocante à arquitetura CORBA, os trabalhos se dividem em dois rumos: estender o barramento lógico de comunicação (ORB) com recursos típicos de tempo-real [Wolf97] [Schm98] e de tolerância a falhas [Maff97], ou prover serviços estratificados de infra-estrutura (baseados nos CORBAservices) e acessíveis via APIs bem definidas, formando plataformas multimídia distribuídas [Loyo93] [Bufo94] [Yun+97] [Hong98]. Em uma outra direção, vem se empregando CORBA como alicerce tecnológico para a construção e validação (implementação) de padrões/propostas em áreas afins a multimídia distribuída, tais como *Computação Móvel* e *Telecomunicações* Para esta última, a arquitetura CORBA vem servindo de base para a construção de ambientes de processamento distribuído (*DPEs*) [Pint99], os quais funcionam como camadas de suporte à construção em larga escala de serviços inteligentes e avançados de telecomunicação (TINA) [Mage97] [Oliv98].

Capítulo 4

Objetos Persistentes

O intuito deste capítulo é trazer uma análise qualitativa e crítica da especificação do Serviço de Objetos Persistentes (POS) [OMG94a] [OMG98a] da arquitetura CORBA. Serão descritos os objetivos que estão por trás dessa especificação, as funcionalidades e interfaces de seus componentes, bem como levantados os seus pontos fracos e fortes, tendo em mente os requisitos de objetos multimídia apresentados no capítulo anterior. Antes, porém, alguns conceitos e abordagens alternativas no suporte à persistência de objetos serão introduzidos.

4.1 Conceitos

É desejável para muitas aplicações a existência de algum mecanismo ou propriedade que garanta a manutenção de informações acerca de seus objetos entre sessões de execução. Nesse âmbito, o conceito de persistência vem sendo especificado e estudado, já há algum tempo, para cobrir essa lacuna. Ao longo deste documento, as seguintes definições se tornam relevantes:

- **Objeto persistente**: um elemento de dado (objeto) cujo tempo de vida pode exceder àquele da aplicação que o criou.
- **Estado persistente** (de um objeto persistente): uma *tupla* de *n* valores correspondendo aos *n* atributos do objeto. Há a possibilidade de alguns desses atributos — geralmente os com tempos de vida limitados — serem considerados auxiliares, ou seja, seus valores não serem incluídos no estado persistente do objeto. A esses refere-se como *transientes*.
- **Dependências** (de um objeto persistente): o conjunto de todos aqueles objetos referenciados diretamente pelo primeiro.

- **Fechamento transitivo de dependências** (de um objeto persistente): o conjunto de todos os objetos — direta ou indiretamente referenciados — formado pelo grafo transitivo de dependências aninhadas [Meye96].

A persistência é uma propriedade que permite a existência dos dados por limites de tempo bastante variados [Brow88] [Atki96]. Logo, é interessante como suporte a aplicações que tenham que processar grandes volumes de informação, como é o caso daquelas relacionadas à Multimídia. Em vez de se implementar diferentes estruturas para a organização e manutenção da consistência dos dados, tanto em memória como em disco (acarretando, muitas vezes, em sobrecarga às aplicações), pode-se recorrer a um serviço dedicado. A Figura 4.1 [OMG94a] mostra, em suma, os elementos participantes desse serviço. Consoante as definições dadas acima, o estado do objeto pode ser considerado como dividido em duas partes. Uma delas seria o *estado dinâmico*, o qual tipicamente estaria na memória e não seria salvo (mantido) caso ocorresse algum evento de falha no sistema. A outra seria o estado persistente, usado pelo objeto para reconstruir o seu estado dinâmico.

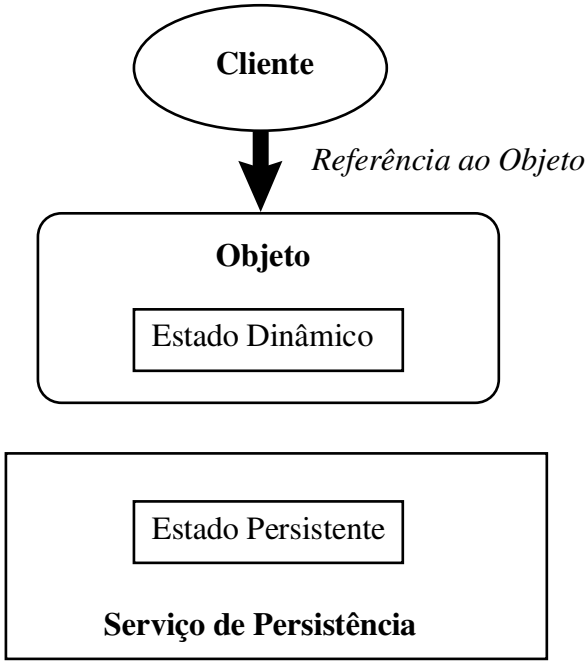

***Figura 4.1**: Elementos de um Serviço de Persistência de Objetos.*

Há variados modos de se proporcionar um suporte adequado à persistência de objetos; os mais comuns seriam mediante o uso de arquivos e bancos de dados relacionais ou orientados a objetos. Independentemente da abordagem escolhida, o ato de se armazenar um objeto inclui a identificação de sua classe e a codificação de seus dados e estrutura (atributos e *links*) em um formato adequado para armazenagem, a fim de que possa ser possível a recuperação fiel do seu estado a qualquer instante de tempo posterior. Ademais, diversas nuanças e decisões de implementação deverão ser resolvidas, tais como, a ativação/desativação da propriedade de persistência do objeto e a política temporal de sua atualização. Para a primeira, Kleindienst et al. [Klei96b] apontam três abordagens:

- **Abordagem estática**: em tempo de compilação, alguns dos objetos da aplicação são estaticamente denotados como sendo persistentes; isso é feito, tipicamente, via

herança de uma classe-base que já provê as condições fundamentais (atributos e métodos) de suporte. Tais objetos irão possuir tal propriedade (a de serem persistentes) durante todo o escopo de tempo definido para a aplicação. De um outro modo: não há como desabilitá-los dessa condição em nenhum momento de sua execução. Essa abordagem é relativamente fácil de ser implementada; no entanto, a rigidez imposta aos objetos nem sempre é conveniente em algumas situações.

- **Abordagem semidinâmica**: essa abordagem é uma modificação da anterior: a persistência de um objeto (denotado estaticamente com tal propriedade) pode ser dinamicamente ativada ou desativada. Embora isso proporcione ao usuário uma maior maleabilidade quando do projeto das aplicações, ainda é necessário que se defina *a priori* quais objetos devem (ou podem) ter o seu estado preservado, o que nem sempre é factível — principalmente quando se trabalha com redes de objetos.
- **Abordagem dinâmica**: essa abordagem é a mais flexível e, por conseguinte, a mais desejável. Ela permite ao usuário dinamicamente decidir quais objetos, dentre todos, podem ter a sua propriedade de persistência ativada em um dado momento (isso é referido como persistência *ortogonal* [Atki96]). Ou seja, objetos podem ativar e desativar a condição de serem persistentes um número arbitrário de vezes dentro do período de execução da aplicação.

Da classificação anterior, pode-se compreender os esforços no suporte à persistência como orientados a dois conjuntos de propostas. O primeiro é o de estender o poder de linguagens de programação orientadas a objeto via conjuntos de classes (bibliotecas ou pacotes) projetados especialmente para essa tarefa. O segundo é o de se oferecer uma estrutura de serviço dedicada e externa à execução da aplicação (isto é, independente), localizada dentro do ambiente por um conjunto de interfaces públicas. Exemplos do primeiro grupo seriam o Pjava [Atki96], o PSE [O'Bri96] e a especificação de objetos *serializáveis* (*Object Serialization* [Sun97a]) para a plataforma Java, analisados, brevemente, na próxima seção. O Serviço de Objetos Persistentes da CORBA e o SGPOM estão sob a segunda categoria de propostas.

4.2 Integração entre Java e Persistência

Java [Gosl96] se mostra uma grande linguagem quando associada à propriedade de persistência de objetos. C++, por exemplo, apresenta problemas relacionados a alguns aspectos de recuperação dos dados, tais como o de ponteiros escondidos (*hidden-pointer problem*), indicado por Channon [Chan96]. Por decisão de projeto, Java provê, em sua estrutura de pacotes, um conjunto de métodos básicos — como *Object.getClass()*, *Class.forName()*, *Class.getName()* e *Class.newInstance()* — de suporte à criação de metaclasses, que tornam as tarefas de persistência e reflexão[1] de objetos mais práticas de serem implementadas. Como extensão a isso, um programador Java pode lançar mão de uma dentre diferentes opções existentes, quando tenta

[1] Reflexão computacional implica na construção de meta-estruturas de informação, usadas na descrição e "inspeção" de hierarquias de classes e objetos, que permitem a um sistema manipular ou modificar seu estado dinamicamente. Uma metaclasse, por exemplo, contém dados adicionais acerca da estrutura particular (conjunto de atributos, métodos e eventos) de uma classe.

satisfazer as suas necessidades de gerência em maior escala de objetos persistentes. As abordagens mais interessantes, hoje, são: Serialização, Engenho de Armazenagem Persistente (PSE), JDBC e Banco de Dados de Objetos.

A abordagem mais simples ainda é o uso direto de sistema de arquivos. Para esse caso, Java oferece um protocolo construído como suporte ao RMI (ver Capítulo 2), conhecido como *Object Serialization*, o qual ajuda na manipulação de objetos, transformando-os em fluxos (*streams*) de dados [Flan97]. Tais fluxos podem ser direcionados para arquivos ou transmitidos via algum mecanismo de IPC (como *sockets*), sendo subseqüentemente lidos, a fim de que se reconstrua os estados dos objetos. Para uso desse protocolo, são definidos pares de interfaces e classes de implementação de alto nível, de suporte à escrita e à leitura dos objetos a partir dos *streams*. Há um novo "atributo de classe", *serializable*, que serve como marca para identificar as classes de objetos passíveis de se tornarem persistentes. Desse modo, pode-se enquadrar essa abordagem, conforme a classificação acima, como sendo semidinâmica, haja vista ficar a cargo do programador a tarefa de projetar o código para ativação ou desativação da propriedade de persistência dos objetos (a marca não implica, necessariamente, que o objeto terá seu estado salvo).

Embora fácil de usar, tal solução pode apresentar problemas de desempenho e de confiabilidade, quando da manipulação de um grande número de objetos. Logo, não se apresenta como a melhor escolha para aplicações que tenham que gerenciar, ou constantemente atualizar, grandes volumes de objetos. Isso porque o seu protocolo se restringe a ler/escrever grafos inteiros de objetos por vez, não podendo fazer nenhuma grande otimização. Quando o fluxo de dados chega à ordem de *megabytes* de tamanho, a armazenagem pode se tornar bastante lenta. Além disso, o protocolo não oferece nenhum suporte a operações de *undo* e *recovery*, o que implica na perda de conteúdo dos objetos quando da ocorrência de falhas.

A fim de atender a essa demanda, uma nova ferramenta de *software* para aplicações Java, chamada de Engenho de Armazenagem Persistente (PSE), oferece um controle mais refinado e explícito sobre as operações de acesso e recuperação. Essa abordagem é interessante quando se quer trabalhar com bases de dados mais robustas a falhas, conquanto mantendo a simplicidade de uso via uma API de suporte a transações. Um PSE deve atender a aplicações que tenham que movimentar dados de dezenas de *megabytes* de magnitude. Porém, não é projetado para atender a múltiplas transações concorrentes provenientes de usuários distintos. Sendo o caso, o seu desempenho começa a ser degradado. A ilustração a seguir (Figura 4.2) [O'Bri96] dá uma dimensão da estrutura de um PSE (um exemplo de produto lançado a partir desse modelo é o ObjectStore PSE™ para Java).

A mais proeminente e madura abordagem proposta para Java no acesso a Sistemas de Gerência de Bases de Dados Relacionais parece ser o JDBC, o qual é um nome registrado e não um acrônimo ou sigla (embora alguns o refiram como "*Java Database Connectivity*"). O JDBC define uma API de acesso a bases de dados que oferece funcionalidades SQL básicas e que permite a interface, via um conjunto de *drivers*, com um número variado de produtos SGBDRs. Funciona, portanto, como um *plug-in*. Com tal mecanismo, Java pode ser usada como uma linguagem hospedeira de suporte à criação de aplicações-cliente de base de dados. No topo do JDBC, APIs de

mais alto nível podem ser construídas: um exemplo seria para o mapeamento de classes e objetos Java em tabelas relacionais, ou vice-versa.

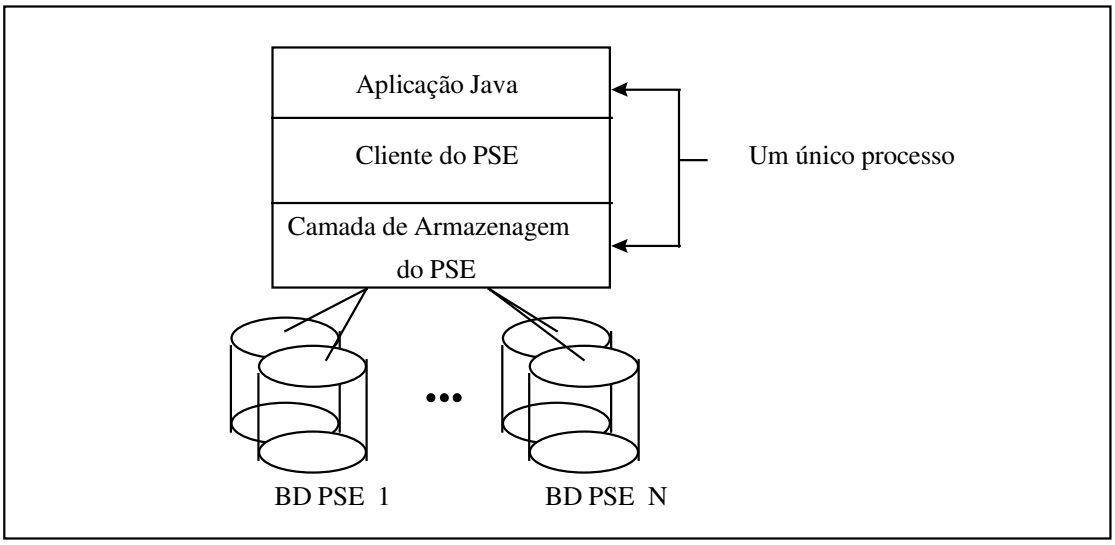

***Figura 4.2**: Engenho de Armazenagem de Persistência.*

Finalmente, com respeito ao acesso a SGBDs Orientados a Objetos, o ODMG está trabalhando em busca de uma API padrão para uso integrado com Java. Tal abordagem almeja prover persistência transparente a múltiplas bases de dados, garantindo, com isso, uma manutenção escalável da concorrência. O seu uso é apropriado em ambientes Cliente-Servidor de multicamadas (*multi-tier*). A idéia é prover às aplicações capacidades próprias de SGBDs, tais como *backup* e recuperação, segurança, tratamento de falhas, transações, etc. A Figura 4.3 [O'Bri96] mostra uma visão da arquitetura sob essa abordagem, proposta como sendo uma extensão de um PSE acessível via um ambiente de rede.

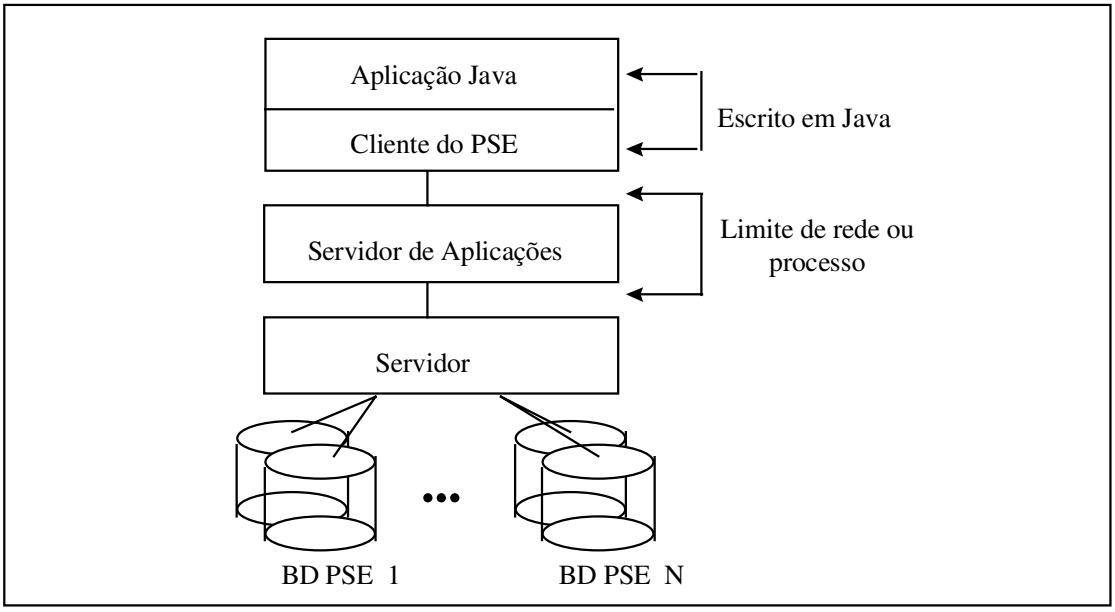

***Figura 4.3**: Arquitetura em camadas de acesso a um PSE.*

4.3 O Serviço de Objetos Persistentes da CORBA

O Serviço de Objetos Persistentes da CORBA (POS) foi especificado para prover suporte à persistência de objetos[2] que seguem o Modelo de Objetos Básico da OMA, de um modo independente do tempo de execução das aplicações (clientes) que os acessam, bem como do tempo de vida das implementações que realizam seus métodos. O POS está diretamente relacionado às operações de armazenagem e recuperação dos estados dos objetos e, para esse propósito, deve lidar com os seus diferentes tempos de vida (desde aqueles que duram o tempo de uma simples tarefa, como as *threads*, até aqueles que precisam ser armazenados ou executados indefinidamente, incluindo os de aplicações típicas de banco de dados e os *daemons*). Mais do que isso, por se tratar de um padrão, deve permitir a coexistência de várias implementações da sua especificação dentro de um mesmo ambiente. Sendo o caso, um simples cliente passa a poder lançar mão, ao mesmo tempo, de uma ou mais dessas implementações, de acordo com as necessidades de seus objetos.

4.3.1 Objetivos

São três os objetivos mais importantes a serem alcançados pelo POS [Sess96]:

1. **Prover suporte a repositórios e bases de dados corporativos**: incluindo, aqui, as bases de dados de todos os tipos e modelos, tais como relacional, hierárquico, sistemas de arquivos...
2. **Prover uma independência com relação a esses repositórios**: o que possibilitaria que, a partir de uma simples API, uma aplicação-cliente pudesse armazenar e recuperar seus dados sem se preocupar com o tipo de repositório a ser usado.
3. **Prover uma arquitetura aberta**: a partir da qual novos produtos (implementações de repositórios) possam ser embutidos com o tempo.

Seguindo a filosofia CORBA, o POS descreve uma arquitetura composta por objetos (componentes), estes localizados em um ambiente distribuído, que se interrelacionam mediante o conjunto de suas interfaces. A especificação do serviço passa, então, a compreender a descrição dessas interfaces, o que é feito usando a linguagem IDL (ver Seção 2.2.2). O POS pode ser pensado como uma "cola" que tenta unir dois mundos (sistemas). O primeiro é aquele composto pelo objeto com propriedade de persistência e seus mecanismos. O segundo é o formado pelo repositório. O que o POS tenta fazer é prover um conjunto de interfaces por intermédio das quais os dois sistemas possam "conversar" (vide Figura 4.4). Um subconjunto dessas interfaces (pelas quais um repositório é acessado) garante a idéia de uma arquitetura aberta. O outro subconjunto (usado pela aplicação-cliente que manipula o objeto) garante a independência com relação ao tipo de repositório.

---

[2] Ao contrário do Capítulo 2, o termo *objeto*, aqui, já assume a conotação de um objeto concordante com a filosofia CORBA.

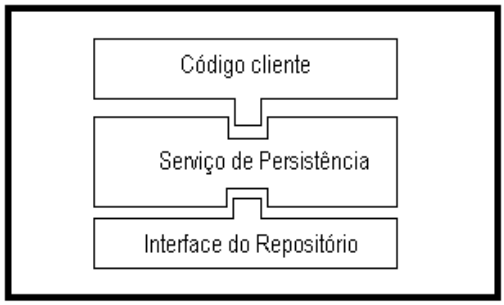

***Figura 4.4**: Visão em alto nível do POS.*

4.3.2 Estrutura do Serviço de Persistência

A Figura 4.5 dá uma dimensão da estrutura do serviço:

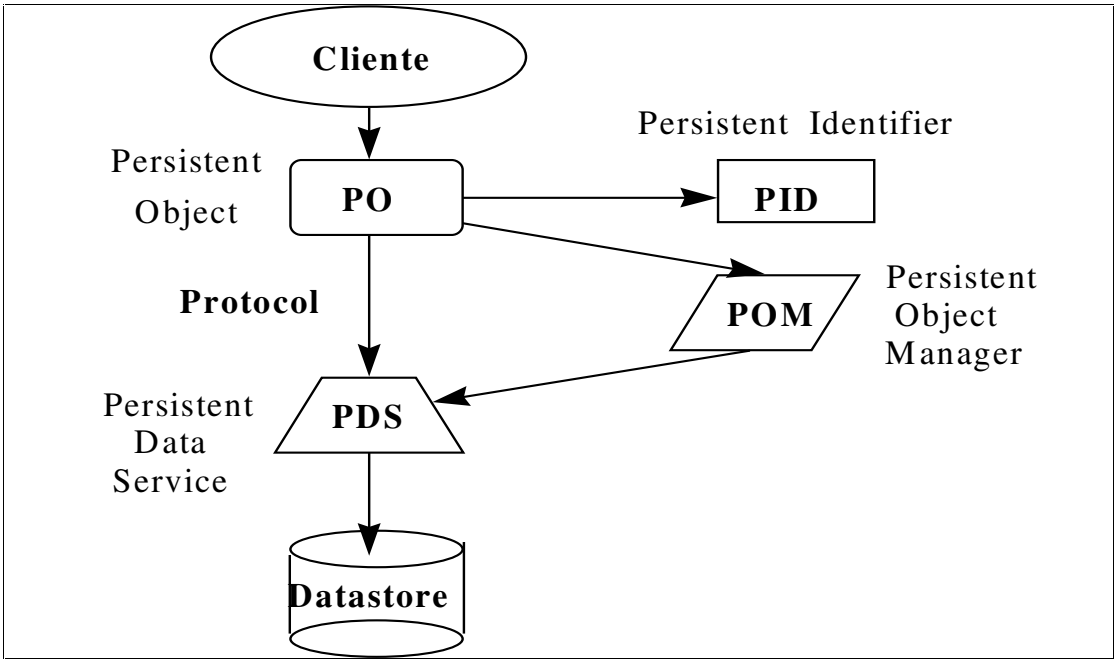

***Figura 4.5**: Componentes do POS.*

De modo conciso, os principais componentes do POS podem ser descritos como:

- ***Persistent Object*** **(PO[3])**: o objeto CORBA cuja persistência é controlada externamente por seus clientes.
- ***Persistent Identifier*** **(PID)**: o qual descreve a localização dos dados de um objeto persistente em alguma fonte de armazenagem (*Datastore*), guardando uma *string* de identificação para esses dados. É um componente abstrato: deverão ser utilizadas especializações de sua interface.

[3] No decorrer desta seção, serão usados, sem distinção de interpretação, os termos *Objeto* e *PO*, ambos se referindo a um objeto CORBA persistente.

- ***Persistent Object Manager* (POM)**: este componente provê uma interface uniforme para a implementação das operações de persistência de um Objeto. Um Objeto tem um único POM para o qual repassa as operações de persistência de mais alto nível.
- ***Persistent Data Service* (PDS)**: este componente oferece uma interface uniforme para qualquer combinação de *Datastore* e *Protocol*, coordenando as operações básicas de persistência para um dado Objeto.
- ***Protocol***: este componente abstrato provê um dentre os diversos modos possíveis de se passar os dados do ou para o Objeto.
- ***Datastore***: este componente (não padronizado) fornece um dos diversos modos de se armazenar os dados de um Objeto, isso feito independentemente do espaço de endereçamento que o contém. Serve como abstração de um repositório.

As relações entre os quatro componentes primários (PO, PID, POM e PDS) podem ser melhor evidenciadas tendo como pano de fundo o cenário da Figura 4.6. Seguindo o fluxo típico de uma requisição de persistência (por exemplo, de uma invocação de armazenagem — *store*), pode-se estabelecer as interações entre esses objetos.

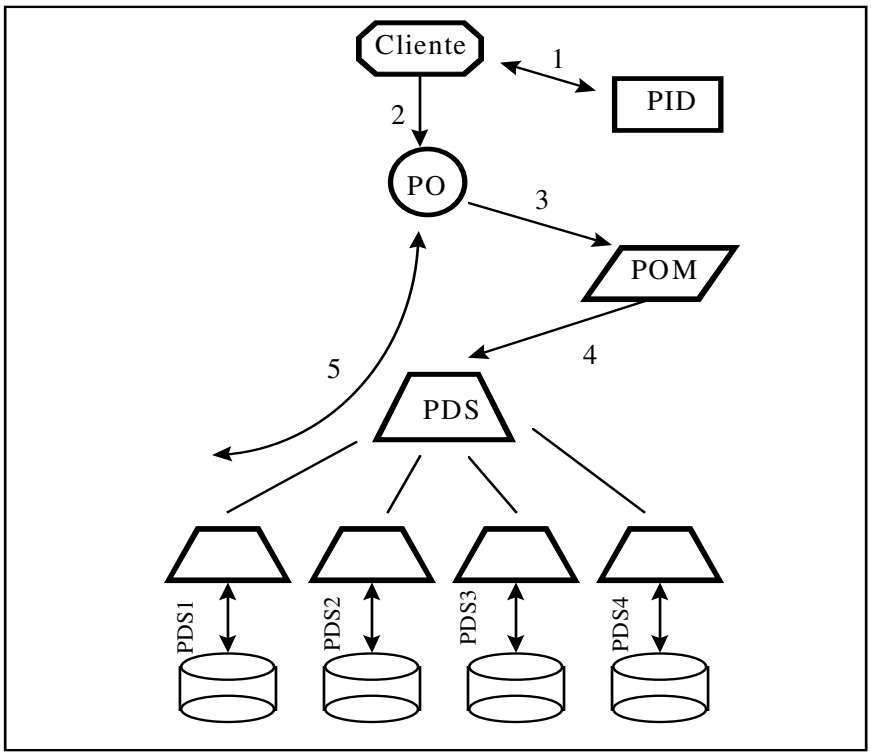

***Figura 4.6**: Fluxo das interações entre os componentes do POS.*

De início, o cliente precisa ter acesso ao PID (passo 1), o qual já deve ter sido instanciado via uma fábrica apropriada (ver Subseção 2.2.5), definindo com ele o tipo de repositório onde deverão ser armazenados os dados que compõem o estado do objeto (espécie de *chave* de identificação). Em seguida, invoca, via interface do PO, uma requisição de armazenagem (passo 2), entregando-o a chave, a qual se encontra ainda vazia. O objeto persistente (PO) encaminha tal requisição para o POM (passo 3). Este, então, baseado no PID e em informações obtidas via PO (por exemplo, a sua

classe ou tipo), caracteriza um protocolo (*Protocol*) de "conversação" para o Objeto e determina qual PDS é o mais indicado para tratar a requisição, repassando-a (passo 4). Finalmente, o PDS passa a trabalhar sobre a armazenagem, interagindo diretamente com o PO e com o repositório (*Datastore*); este último preencherá o PID com a real localização dos dados dentro do seu domínio. Outras interpretações para esse cenário são possíveis; uma delas seria a da não participação da entidade cliente.

4.3.3 A Especificação do Serviço de Persistência

A finalidade desta subseção é a de propiciar uma síntese clara da arquitetura proposta para o POS. Para tanto, passa-se a investigar/analisar as funcionalidades e interfaces dos componentes.

Funcionalidades

A primeira versão do Serviço de Objetos Persistentes é especificada em [OMG94a], onde um conjunto de interfaces em IDL é descrito para os quatro componentes básicos da arquitetura POS. Fundamentalmente, tais interfaces compreendem um conjunto de mesmos métodos, os quais compõem uma família de operações de entrada e saída (E/S): *connect(), disconnect(), store()* e *delete()*. Essa família de operações é suportada em três dos quatro componentes, sendo eles o Objeto Persistente (PO), o Gerente de Persistência (POM) e o Serviço de Dados Persistentes (PDS).

O PO e o PDS representam os dois extremos do serviço. O PO compreende a interface de mais alto nível, aquela acessada pelos clientes (aplicações que manipulam o Objeto) para que se realizem as operações de E/S. Um objeto CORBA, para ser "visualizado" como persistente, deve herdar e/ou implementar essa interface. Já o PDS corresponde à interface de mais baixo patamar, aquela que escreve os *bytes* de informação para o repositório. A idéia é a de se ter diferentes PDSs, cada um especializado e vinculado a um tipo de repositório (conquanto isso não seja uma imposição), o que garantiria o primeiro dos objetivos do POS, qual seja o de suporte a implementações comerciais já existentes.

Um PDS suporta uma coleção de pares <repositório, protocolo>. Um repositório é quem realmente salva (guarda) e recupera os dados do Objeto, enquanto que um protocolo descreve o modo pelo qual um PDS transfere o estado de e para o PO. Tanto o repositório como o protocolo são entidades não padronizadas pela especificação. Contudo, esta última oferece três exemplos de possíveis protocolos e descreve um *Datastore_CLI* como amostra de uma interface uniforme para acesso a diferentes repositórios. Grosso modo, um PDS se comunica com um PO por meio de um protocolo e com o repositório via essa interface (ou outra proprietária).

Um POM realiza, dinamicamente, a ligação (*binding*) entre o PO e o seu correspondente PDS, baseando-se no PID associado ao Objeto e no protocolo suportado por este. É aquele componente que tem a função de um *roteador* de requisições. O PID funciona como um identificador associado à localização do estado derivado do PO armazenado no repositório. Tal PID é basicamente representado por uma tripla de valores: <*datastore_type*, *datastore_type_instance_id*, *key_to_PO*>.

Exemplos desses valores seriam <FS, máquina, *path*>, para um repositório do tipo sistema de arquivos, e <BD, nome da BD, chave>, para um repositório do tipo base de dados.

A partir de tais considerações, passa a ser função do POM conhecer todos os PDSs disponíveis no ambiente, mantendo uma lista de combinações <repositório, protocolo> que cada PDS pode suportar. Uma implementação para essa lista seria, por exemplo, a de se guardar os valores em um arquivo ou registro de configuração; outra seria a de prover uma interface adicional dedicada mediante a qual os PDSs poderiam se registrar. Uma segunda função do POM é a de conhecer o protocolo associado a ou suportado por um Objeto. Em termos de implementação, tal protocolo poderia ser deduzido a partir do tipo ou classe a que pertence a instância.

O intento do POS de ser uma arquitetura aberta é alcançado na medida em que ele permite que quase todas as *n*:*m* combinações possíveis entre os seus componentes (tanto instâncias como tipos) sejam factíveis de se configurarem [Klei96a]. Como conseqüência disso, um Objeto e um PDS podem suportar um ou mais protocolos, um PDS pode trabalhar sobre um ou mais repositórios, ou mais de um PDS pode ter acesso a um dado repositório.

Análise do Conjunto de Interfaces

O quadro a seguir (Tabela 4.1) traz o conjunto básico de interfaces (e módulos) em IDL definido para os componentes do POS.

Módulos	Interfaces	Operações/atributos
CosPersistencePID	PID	*attribute string datastore_type;* *string get_PIDString ( );*
CosPersistencePO	PO	*attribute CosPersistencePID::PID p;* *CosPersistencePDS::PDS connect (in CosPersistencePID::PID p);* *void disconnect (in CosPersistencePID::PID p);* *void store (in CosPersistencePID::PID p);* *void restore (in CosPersistencePID::PID p);* *void delete (in CosPersistencePID::PID p);*
CosPersistencePO	SD	*void pre_store ( );* *void post_restore ( );*
CosPersistencePOM	POM	*CosPersistencePDS::PDS connect (* *in Object obj, in CosPersistencePID::PID p);* *void disconnect (in Object obj, in CosPersistencePID::PID p);* *void store (in Object obj, in CosPersistencePID::PID p);* *void restore (in Object obj, in CosPersistencePID::PID p);* *void delete (in Object obj, in CosPersistencePID::PID p);*
CosPersistencePDS	PDS	*void connect (in Object obj, in CosPersistencePID::PID p);* *void disconnect (in Object obj, in CosPersistencePID::PID p);* *void store (in Object obj, in CosPersistencePID::PID p);* *void restore (in Object obj, in CosPersistencePID::PID p);* *void delete (in Object obj, in CosPersistencePID::PID p);*

***Tabela 4.1**: Módulos e interfaces (em IDL) dos componentes do POS.*

Como se observa, o único dos componentes com uma assinatura diferente das operações é o PID, o qual define um atributo (o *datastore_type*) e uma operação (*get_PIDString()*). O tipo do repositório onde estão localizados os dados persistentes de um Objeto pode ser obtido via PID de dois modos: ou via o atributo

datastore_type, ou pelo próprio tipo (especialização) do PID. O que se nota é que há muito pouca informação disponível num PID genérico. Isso nos leva a acreditar que a especificação, propositadamente, deixou em aberto (delegou aos fabricantes) a especialização desse componente de acordo com a implementação (ou tipo) de repositório associado — Sessions [Sess96] advoga, por exemplo, que tais PIDs poderiam conter atributos adicionais específicos ao repositório. Como extensão a essa interface, poderia ser incluída uma segunda operação, com semântica simétrica àquela padronizada, cuja assinatura seria dada por *void SetPIDString(string)*. Tal operação seria de interesse quando da instanciação de um PID a partir de uma *string* apropriada.

Os outros módulos são compostos pelo mesmo conjunto de operações de E/S. No caso do módulo *CosPersistencePO*, é definida uma interface adicional (chamada de SD — *Synchronized Data*), de caráter opcional, que pode ser suportada por objetos que tenham que manter sincronizadas as suas partições persistente (por exemplo, em um *buffer*) e transiente (em memória). Um caso típico desses objetos é o de um processador de palavras que conserva parte da informação em uma memória *cache* (de uso mais rápido), acessada e modificada mais freqüentemente, e uma outra parte, atualizada de tempos em tempos, em um *buffer*. Quando alguma requisição de armazenagem é feita, somente a informação da partição persistente (no *buffer*) passa a ser salva; entretanto, se ocorrer alguma falha no sistema, dados escritos no *cache* serão perdidos. A fim de evitar esse risco — pois, em se tratando de aplicações complexas, as informações, por restrições de tempo, não podem ser continuamente armazenadas —, tal interface provê um mecanismo (mediante suas operações abstratas *pre_store()* e *post_restore()*) para sincronizar essas partições quando as operações *store()* e *restore()* venham a ser invocadas.

Todavia, suporte à interface SD não implica, necessariamente, em suporte à interface PO e vice-versa. Tais interfaces, ainda que pertencentes a um mesmo módulo, são ortogonais. A PO revela que os clientes poderão controlar quando um Objeto será armazenado ou recuperado, sem se preocupar com a sincronização entre os estados transiente e persistente. Por outro lado, a interface SD indica que os estados serão sincronizados, sem levar em consideração se as requisições de *store()/restore()* serão feitas pelo cliente do objeto ou por ele próprio.

Em se tratando das operações *store()*, *restore()* e *delete()*, para cada um dos componentes, pouco precisa ser mencionado. O comportamento dessas operações é facilmente descrito pelas próprias assinaturas. O parâmetro de tipo *Object* (ver Subseção 2.2.1) é a Referência (*Object Reference* — IOR) ao objeto persistente, enquanto o argumento do tipo PID identifica a localização de seu estado persistente no repositório.

Por outro lado, as operações *connect()* e *disconnect()*, da forma como foram especificadas para a interface de mais alto nível (a PO), não estão sendo bem vistas por especialistas [Sess96]. Esse par de operações foi introduzido para definir uma abstração de *sessão* de diálogo entre o PO e o PDS. Durante esse espaço de tempo, o estado do Objeto se manteria consistente com aquele armazenado no repositório, podendo as duas representações dos dados serem vistas como uma só. Todavia, futuras revisões da especificação do POS (advindas mediante novos RFCs) deverão levar em consideração a semântica (interpretação) pobre que está por trás das

assinaturas dessas operações (principalmente, com relação ao valor de retorno da operação *connect()*). Aparentemente, parece não fazer sentido (a menos para certas classes especiais de bases de dados) que se permita ao cliente do objeto fazer uma invocação de abertura ou fechamento do repositório, algo que poderia ser automaticamente realizado pelos componentes de mais baixo nível (os PDSs) — quando da condução das atividades de armazenagem, recuperação e remoção, *connect()* e *disconnect()* seriam, implicitamente, realizadas.

A partir das funcionalidades descritas no item anterior e do conjunto de módulos analisados nesta parte, pode-se esboçar o modelo de objetos (conhecido como *modelo estático*, seguindo a metodologia OMT [Rumb91]) para o POS, traçando as relações existentes entre os seus componentes. O diagrama da Figura 4.7 traz essa contribuição. Nele, pode-se identificar os módulos e as interfaces (contendo operações padronizadas ou não) correspondentes a cada um dos componentes. No ensejo, retratou-se o *Protocol* como um registro de auxílio ao POM, contendo atributos e métodos de identificação para pares de POs e PDSs. Foram providas algumas especializações possíveis de PIDs e PDSs, de acordo os tipos de repositórios (não revelados na figura) disponíveis no ambiente.

4.3.4 Dependências do POS para com outros CORBAservices e vice-versa

Seguindo o Princípio de Bauhaus (ver Subseção 2.2.5), os CORBAservices apresentam interdependências, de modo a evitar a duplicação de funcionalidades. Ou seja, os serviços são projetados para que suas implementações se utilizem de outras já existentes. Os quadros a seguir (Tabelas 4.2 e 4.3) apresentam, sucintamente, algumas relações de dependência previstas em [OMG95b], apresentando as possíveis razões. Porém, à medida que as especificações evoluem (são reavaliadas ou refinadas), ou novos serviços são propostos, outras dependências aparecerão. O primeiro quadro apresenta as dependências do POS para com outros CORBAservices, enquanto o segundo indica aqueles que podem se beneficiar de uma implementação do POS.

CORBAservice	Causa possível de dependência
Gerência de Mudanças	*para armazenar versões dos estados dos objetos*
Controle de Concorrência	*para suportar compartilhamento dos estados dos objetos*
Externalização	*para transformar o estado para um stream de dados e vice-versa*
Repositório de Implementações	*para determinar informações acerca da implementação do objeto*
Repositório de Interfaces	*para armazenar definições de tipos*
Transações	*para realizar o commit das operações de armazenagem, recuperação e remoção com garantias ACID*
Ciclo de Vida	*para criação e remoção dos objetos*
Nomes	*para a obtenção de referências entre os objetos a partir de nomes lógicos (um PDS, por exemplo, se registra junto ao POM por meio de um nome fictício)*

***Tabela 4.2**: Dependências do POS com relação a outros CORBAservices.*

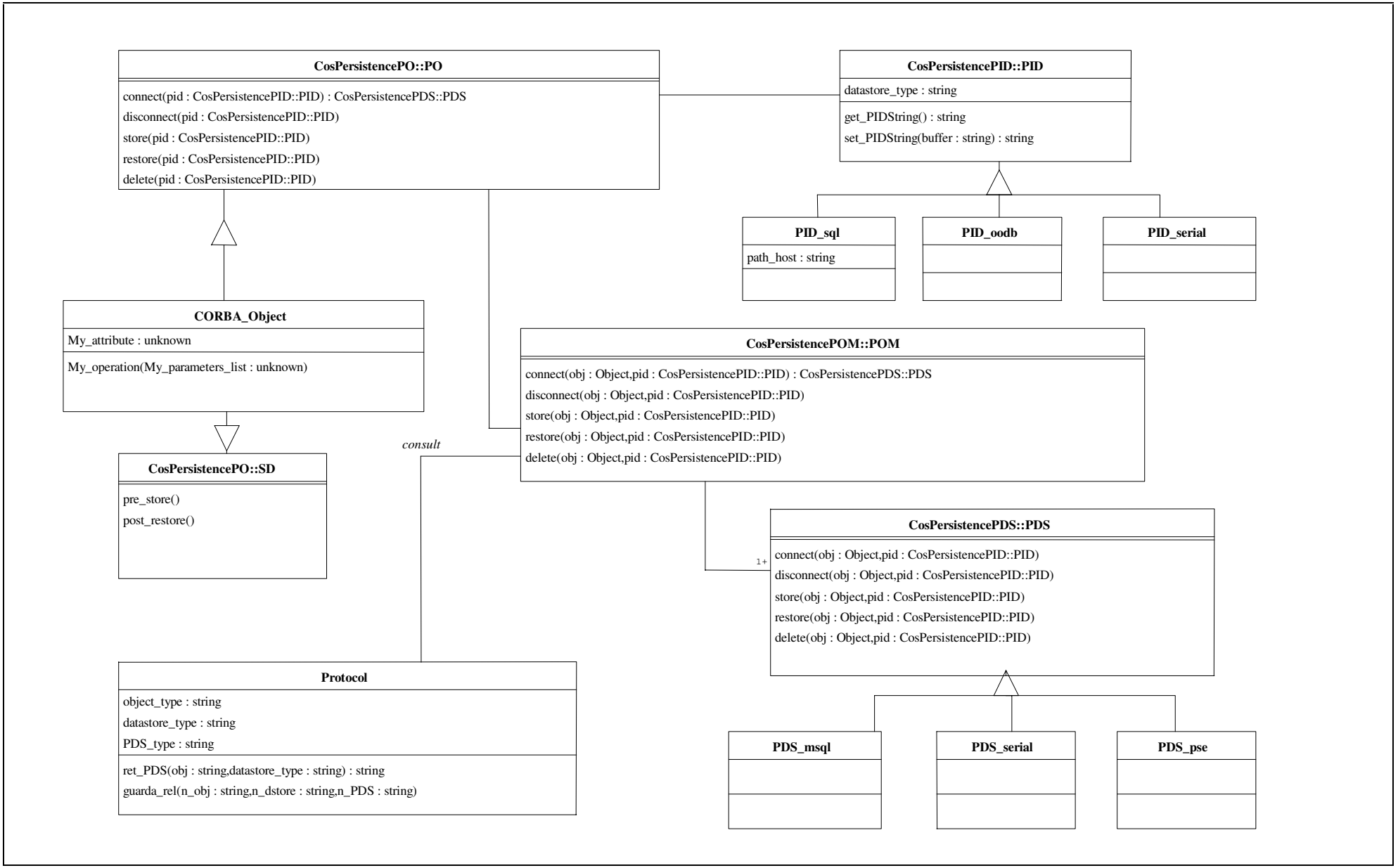

***Figura 4.7**: Modelo de Objetos (OMT) para o Serviço de Persistência da CORBA.*

CORBAservice	Causa possível de dependência
Arquivo (*Archive*)	*para prover persistência aos objetos que serão "empacotados" num só arquivo, possibilitando a sua recuperação no caso de falhas*
Cópia/Recuperação (*Backup/Recover*)	*para armazenar logs*
Gerência de mudanças	*para suportar mudança a objetos persistentes/transientes*
Controle de Concorrência	*para manter uma tabela persistente de locks*
Repositório de Implementações	*para salvar permanentemente objetos*
Repositório de Interfaces	*para salvar permanentemente objetos*
Ciclo de Vida	*se um novo objeto criado deve ser automaticamente apontado como persistente*
Nomes	*para salvar mapeamentos entre nomes e objetos*
Propriedades	*para fazer com que as propriedades se tornem persistentes*
Consulta	*se uma coleção de objetos, fruto de uma consulta, deve se tornar persistente*
Negociação (*trading*)	*para armazenar informação no Repositório de Interfaces*
Transações	*para permitir operações atômicas em objetos persistentes*
Intercâmbio de dados (*Data Interchange*)	*para permitir a troca de estados entre objetos persistentes*
Replicação	*para permitir a manutenção da consistência entre réplicas de objetos persistentes*

***Tabela 4.3**: Dependências de alguns CORBAservices com relação ao POS.*

4.3.5 Alternativa ao POS

Reverbel [Reve96] propõe uma alternativa ao POS para se implementar a persistência de objetos CORBA. A sua abordagem se baseia na integração ORB/SGBDOO a ser provida via um componente intermediário — um adaptador de objetos (OA) especial — conhecido como ODA (ver Subseção 2.2.3). O intuito desse adaptador é tornar disponíveis (visíveis) ao ambiente distribuído os objetos armazenados num SGBDOO. Desse modo, implementações de entidades CORBA podem ser codificadas, diretamente, sob a linguagem de programação provida pelo SGBDOO (a qual já incorpora mecanismos de suporte à persistência), servindo-se, com isso, de outras funcionalidades existentes, tais como consistência de dados na presença de acessos concorrentes, interrupção de transações corrompidas e recuperação a falhas.

O autor defende que essa abordagem é interessante para os casos onde se deseja manter um objeto CORBA persistente acessível a seus clientes, sem, todavia, precisar expor o esquema (*schema*) usado pela BD — os dados e o *layout* do objeto se mantêm privados. Analogamente ao POS, o intuito é o de delegar ao Objeto (à sua própria implementação ou, como extensão, a seus clientes) a responsabilidade pela gerência (ativação/desativação) de seu estado persistente. O enfoque maior é dado sobre o ODA, o qual deve garantir a transparência de armazenagem (persistência) do Objeto, isto é, torná-lo continuamente disponível a invocações, mesmo sendo um elemento comum de BD. Para isso, o ODA deve oferecer um mecanismo extra que garanta também a persistência das *Object References*.

Para os propósitos de construção do SGPOM, essa abordagem foi preterida, principalmente, por duas razões:

- **Pouca flexibilidade**: em contrapartida ao POS, não é proposta nenhuma estrutura normatizada (conjunto de interfaces públicas padrão) que permita adaptar (especializar) a abordagem para os casos onde se deseja trabalhar com repositórios (BDs) construídos sob diferentes modelos ou tipos (de relacionais a proprietários). Ou seja, a proposta está muito atrelada à funcionalidade provida pelo ODA.
- **Desempenho**: todo o acesso a ser feito ao Objeto deve passar, necessariamente, pelo ODA, tornando-o, caso não seja bem projetado, um possível ponto de gargalo. Sessões de interação (várias invocações em seqüência) devem ser evitadas, mormente quando vários Objetos necessitam ser acessados simultaneamente. No caso de OMs de mídia contínua, os quais apresentam grandes volumes de dados, recuperá-los através de um ORB convencional torna-se impraticável em termos de desempenho [Schm98]. Mais ainda, tais objetos, consoante a visão do SGPOM, não apresentam interfaces de acesso (encapsulam somente um *stream* de dados), o que torna dispensável a figura do ODA como ponte de integração ORB/SGBDOO.

4.3.6 Avaliação crítica sobre o POS

Por ser especificado como um padrão aberto e genérico, independente de aspectos de implementação, em nenhum momento o POS faz alguma alusão a como se atender a necessidades específicas e inerentes a algumas classes de aplicações. Por esse motivo, passa-se nesta subseção a avaliá-lo consoante os requisitos multimídia levantados no capítulo anterior (Seção 3.5), discutindo os seus pontos fortes e fracos. Para essa análise, as referências [Klei96a], [Klei96b], [Sess96] e [Tüma97] serviram como fontes de contribuição. Como pontos favoráveis, pode-se citar:

Serviço em camadas

Uma das características do POS é a de ser estratificado. Cada camada esconde de suas anteriores as funcionalidades que municia, garantindo a modularização. Assim, pode-se ter uma visão *top-down* da arquitetura, onde cada nível está mapeado em um componente, cujo serviço pode ser acessado a partir de sua interface. Isso permite também que cada componente se torne *especializável*, ou seja, possa ser remodelado (ou estendido) para atender a requisitos particulares de implementações, sempre cumprindo as suas funcionalidades básicas.

Transparência no acesso aos dados

Como um dos objetivos colocados na Subseção 4.3.1, o POS tenta, realmente, garantir o que seria uma "independência com relação ao repositório". Isso porque para o cliente do Objeto ou para o próprio, é necessário que se mantenha transparente a localização dos seus dados. O que ele precisa portar consigo são um "endereço" na forma de uma *Object Reference* (ou seja, conhecer a interface e o conjunto de operações de um gerente associado — POM) e uma chave identificadora dos seus dados (PID) no repositório.

Serviço intrinsecamente distribuído

Mesmo que seja aparentemente prolixo ou dispensável discutir sobre esta característica, ela é, na verdade, a principal e engloba as anteriores. O POS, seguindo a filosofia CORBA, é especificado para ser um serviço distribuído. Essa distribuição é o

seu atrativo basal e garante que ele possa ser integrado a um ambiente aberto, servindo de infra-estrutura para o desenvolvimento de outros serviços ou de novas aplicações.

Suporte à Persistência Seletiva

Projetado para prover uma abordagem dinâmica de ativação/desativação da persistência (ver Seção 4.1), o POS é centrado em objetos que expõem essa propriedade para clientes CORBA. Com isso, uma aplicação (multimídia, inclusive) pode decidir, a qualquer momento, quais dos seus objetos terão o seu conteúdo (estado) salvo e quais serão (ou permanecerão durante um período) transientes.

Embora possa ser elogiável o esforço por parte de seus projetistas em tornar a especificação do POS a mais aberta possível, essa qualidade, como efeito colateral, acaba por limitar, de certo modo, a compatibilidade entre implementações concordantes [Sess96]. Em certas condições, tais implementações terão precisamente que definir ou especializar alguns aspectos de serviço e, por decorrência, se tornarão incompatíveis com outras. A seguir, são ressaltados alguns dos pontos débeis que estão por trás da especificação e podem ser alvo de futuras modificações:

Semântica da operações connect() e disconnect()

Sob este tópico, deve ser criticada (como mencionado na Subseção 4.3.3) a questão do valor de retorno da operação *connect()* da interface PO. A especificação do POS relata que o PDS é retornado "para uso por parte daqueles protocolos que requeiram que o PO entre em contato com o PDS" [OMG94a]. Não obstante, parece ser ilógica tal interpretação, haja vista que o PO não é o iniciador da chamada e sim o alvo dela. Ou seja, quem tem que principiar a comunicação é o PDS, a partir do momento que o POM lhe passou a requisição e o "endereço" do PO. Mais ainda, pelo fato de tal operação retornar um PDS, subentende-se que é o cliente, e não o PO, quem passa a conversar com o PDS. A questão que surge é: Para quê?

Uma explicação para esse fato seria a de tornar possível para o cliente "fugir" da regra de divisão do serviço em camadas [Sess96]. Ou seja, o cliente passaria também a interagir com o elemento de mais baixo nível, o PDS, a fim de requisitar alguma operação adicional sua, não especificada pelo POS e, portanto, não geral. Isso acarretaria no que irá ser discutido no ponto seguinte.

Problema da especialização dos serviços e de sua visibilidade

Embora a especificação do POS apresente a interface PO como o único caminho necessário (e suficiente) para que o cliente de um Objeto acesse o serviço, isso não pode ser assegurado (nem forçado). Já que todas as requisições devem eventualmente passar pelo POM, nada impediria, por sinal, que um cliente requisitasse diretamente a este alguma operação. Ou, como no caso anterior, nada impediria ao cliente, tendo conhecimento do PDS, de se conectar diretamente a este último para fazer alguma requisição específica. Contudo, isso afeta diretamente a proposta e filosofia da estrutura, qual seja a de esconder as funcionalidades e a de proporcionar uma independência com relação ao repositório. Decerto, é recomendável não se permitir esse tipo de comportamento, ou seja, nem possibilitar com que camadas (componentes) sejam desviadas (por exemplo, o cliente acessar diretamente o POM), tampouco assumir uma ligação explícita entre cliente e repositório.

Exceções

Durante toda a especificação, as interfaces são projetadas para um "uso direto". O tratamento a ser dado para situações de caráter excepcional (exceções), por exemplo, não é abordado [Klei96a]. Os casos mais importantes incluídos nesse contexto são os que envolvem as operações de *connect()*, com um único Objeto se conectando mais de uma vez, dentro de uma mesma sessão, com o mesmo repositório, *disconnect()*, quando se tenta desconectar um objeto de um repositório sem se ter realizado primeiro uma conexão lógica, e *restore()*, quando se tenta recuperar um estado que não é compatível com o Objeto-alvo. Mesmo que implementações venham a resolver ou a abordar esses problemas (considerando essas invocações como erro), não se pode, por exemplo, garantir (por falta de suporte e pela estrutura do serviço) que uma notificação de exceção chegue até o cliente por intermédio das interfaces padronizadas. Como decorrência, a implementação irá introduzir uma solução proprietária para o tratamento dessas condições anômalas, fugindo do escopo da especificação.

Funcionalidade do POM pouco especificada

Sem justificativa, o padrão não entra em detalhes sobre a funcionalidade do POM (como se verá, uma figura importante para a caracterização do SGPOM). Ao contrário dos requisitos colocados nos primeiros RFPs (*Requests For Proposal*), o critério usado pelo POM para despachar as invocações para os PDSs é delineado, muito brevemente, na especificação do POS. Mais ainda, não são explanados os meios pelos quais os Objetos e os PDSs acessam ou se registram junto ao POM — uma solução para o acesso seria a de se fazer uso do Serviço de Nomes, para se recuperar a referência do POM a partir de um nome lógico. Novamente, o que se espera é que os projetistas e desenvolvedores introduzam mecanismos proprietários que possibilitem essa interação. Exemplificando: uma solução para o registro dinâmico dos PDSs disponíveis poderia ser realizada a partir de uma interface (ou operação) adicional introduzida no POM.

Um dos argumentos colocados é o de que o POM, na verdade, não seria mais do que um PDS modificado [Klei96a], haja vista que ambos os componentes apresentam operações sintaticamente idênticas — à exceção de *connect()*. Uma idéia sugerida seria a de se permitir uma estrutura hierárquica de PDSs, passando o POM a atuar como seu nó raiz.

Carência de um Serviço de Persistência Composto

Um dos pontos-chave relacionados com a persistência de quaisquer objetos — principalmente a de OMs — é a manipulação das associações (conjunto de dependências) existentes entre eles. A especificação do POS, por sua vez, nada trata a respeito, haja vista que delega essa tarefa ao Serviço de Relações [OMG94c]. Mais ainda, a arquitetura proposta apenas associa um Objeto ao seu PID de forma muito tênue, tornando as referências entre Objetos mais difíceis de serem implementadas. Da mesma forma que foi concebido um adendo sob o nome de Serviço de Ciclo de Vida Composto [OMG94d] — o qual promove a idéia de um serviço de criação de Objetos compostos —, poderia ser analogamente especificado um anexo ao POS que possibilitasse o tratamento adequado da persistência desses agregados de dados.

Objeto atrelado diretamente à tarefa da persistência de seu estado

A filosofia embutida no POS é a de transferir ao PO (ou a seus clientes) a total responsabilidade pela manipulação de seus dados. Ou seja, o Objeto (ou seus clientes) deve lidar com as tarefas de ativação e desativação da persistência do seu estado. Isso, porém, não é adequado para aqueles OMs que não possuem estados definidos; apenas encapsulam dados e *streams* de uma ou mais mídias. Tais objetos, como visto no capítulo anterior, só apresentam algum significado quando inseridos em contextos lógicos definidos pelas aplicações que os criam, manipulam ou apresentam. Como o POS visa atender apenas a objetos que seguem o Modelo de Objetos da CORBA, suas funcionalidades não atingem aqueles que não possuem interfaces IDL definidas, estando, por sua vez, contidos em outras aplicações. A proposta do SGPOM está francamente vinculada a essa consideração.

4.4 Considerações Finais

A persistência está diretamente relacionada ao período de tempo em que um objeto existe e é útil à aplicação a que pertence [Silv97]. O interesse nessa propriedade pode ser melhor descrito tendo em mente o escopo dos dados a serem manipulados [Atki83] [Brow88]:

1. dados usados apenas na avaliação de uma expressão;
2. dados locais à ativação de uma rotina;
3. dados globais a todo o programa ou que sobrevivem à rotina (método ou função) que os criou;
4. dados que existem entre execuções de um programa;
5. dados que existem entre versões de um programa;
6. dados que continuam a existir além do programa.

Em sistemas tradicionais, as categorias (1) a (3) são usualmente suportadas por uma linguagem de programação, enquanto as demais, por um sistema de arquivos ou um sistema de gerência de bases de dados. Essa separação de suporte aos dados introduz uma distinção explícita entre aqueles de vida breve e os de longa duração. Para estes últimos, códigos adicionais são necessários a fim de que se mantenha a consistência entre as cópias dinâmica (em memória) e persistente (salva em disco).

Um serviço dedicado à persistência de dados deve garantir às aplicações que os manipulam uma independência transparente para com as fontes de armazenagem, atuando como uma camada intermediária de integração dos dois mundos. No caso de objetos distribuídos, alguns esforços vêm sendo conduzidos na busca de um paradigma adequado à concorrência e à manutenção consistente dos dados de componentes. A especificação do Serviço de Objetos Persistentes do OMG pretende atender, ainda que em parte, a essa demanda. Um trabalho de peso realizado sobre esse serviço foi conduzido, até recentemente, por Tüma [Tüma97].

Todavia, essa abordagem está passando por ciclos de revisão, tendo em vista que o seu caráter genérico parece não ter atendido às expectativas mais comuns de implementação. O próprio OMG reconhece a inabilidade do POS (versão 1.0) de cumprir o seu papel enquanto serviço básico, lançando as bases para a especificação de um nova arquitetura. O PSS (*Persistent State Service 2.0*) [OMG97b], ainda em fase de

submissão e avaliação de *drafts*, tende a cobrir essa lacuna, incorporando os mesmos objetivos do POS (ver Subseção 4.3.1). A meta é a de suprir as deficiências encontradas, eliminando dubiedades de interpretação da mecânica de interação dos componentes. A idéia que está por trás dessa nova empreitada é a de se imaginar uma especificação mais coerente e pragmática, suprimindo funcionalidades secundárias e proporcionando mecanismos para uma ativação "automática" da persistência. Não é exigido, contudo, que a nova arquitetura seja compatível com aquela proposta para o POS, o que, de certo modo, salienta a mudança de escopo.

O que se espera é que o PSS atenda a requisitos claros de escalabilidade, flexibilidade, eficiência, portabilidade e otimização, impostos por certas classes de aplicação. Desse modo, a sua integração a outros CORBAservices, mormente àqueles especificados posteriormente ao POS, e o uso de novos recursos a serem introduzidos com a especificação CORBA 3.0 (POA, por exemplo) devem ser levados em consideração. O intuito é o de se criar um *framework* mínimo de componentes, onde interfaces adicionais poderão ser futuramente incluídas, causando o menor número de modificações.

No âmbito dos repositórios, a especificação do PSS deve proporcionar um maior suporte à criação de *Schemas* neutros a modelos ou produtos. A possibilidade da definição de *views* (visões) em IDL, por exemplo, passaria a proporcionar um alicerce útil para as tarefas de realocação dos estados dos objetos (individualmente ou coletivamente) de um repositório para outro. Por outro lado, as novas interfaces para com os repositórios deverão permitir que implementações incorporem *tradeoffs* mais adequados às suas necessidades.

Capítulo 5

Caracterização do Serviço

Em face das considerações feitas nos segundo, terceiro e quarto capítulos desta dissertação, passa-se a descrever em detalhe a especificação de um serviço de gerência distribuído — responsável por tarefas de armazenagem e recuperação de objetos multimídia persistentes — localizado sobre repositórios multimídia (SGPOM). Para esse fim, ressaltam-se as principais deliberações de projeto e de implementação tomadas durante a sua idealização, explicitando as suas características segundo duas perspectivas: *funcional* (que delineia o contexto lógico do serviço dentro da proposta Banco de Dados Multiware) e *estrutural* (a qual apresenta os componentes, o conjunto de interfaces e a dinâmica de interações).

5.1 Visão Geral

Grosso modo, o SGPOM assume o papel de responsável pela manipulação e tratamento dos estados persistentes de objetos que compõem aplicações ou documentos multimídia. É localizado, dentro de um ambiente distribuído, por intermédio de um conjunto de interfaces públicas, permitindo a uma ou mais aplicações recuperarem, segundo restrições temporais, os estados de seus componentes multimídia. Sua arquitetura é pautada em uma coleção de objetos gerentes, integrando uma estrutura hierárquica e extensível, relacionada a um certo domínio ou classe de aplicações.

Formalmente, o SGPOM está montado sobre um conjunto de repositórios multimídia configurados para o tratamento adequado de objetos de mídia (contínua ou discreta) codificados sob formatos particulares. O SGPOM é uma camada de gerência composta por um *framework* de objetos distribuídos que torna transparente às aplicações-cliente a localização de e o acesso a esses repositórios. O seu compromisso é prover um instrumento de suporte à persistência de dados multimídia que passa a

atender a invocações de aplicações (ou outros serviços) que tenham acesso a um ORB. Desse modo, a arquitetura normatizada para o POS serve como modelo para o SGPOM, o qual a estende e enriquece a fim de atender às restrições temporais e de armazenagem inerentes aos OMs.

5.2 Perspectiva Funcional

Descrever o contexto lógico onde está inserido o SGPOM é a meta desta seção. Por conseguinte, serão apresentadas, num primeiro momento, as possíveis interações entre este e os outros componentes do ambiente Multiware. Posteriormente, serão estabelecidas as funcionalidades (e obrigações) a serem satisfeitas, tanto pelo SGPOM como por seus clientes, para que o contrato de serviço seja viável. Isso é feito na forma de ponderações de projeto e de implementação.

5.2.1 Contexto

Quando do processo de composição de apresentações multimídia, é comum se utilizar de partes de um ou mais objetos de mídia simples, já existentes ou não. Pode-se estar interessado, por exemplo, em apenas poucos segundos de um filme de várias horas ou em um simples detalhe de uma imagem ou fotografia. Sendo esse o caso, a disponibilidade de esquemas (*schemas*) para a representação desses fragmentos de mídia e de suas interdependências espaciais e temporais é de substancial importância para serviços de autoria e de apresentação de documentos multimídia. Com esses esquemas, capturam-se as estruturas de OMs complexos, codificando-os sob um formato que guarda as informações relevantes à sua apresentação. Esse formato, o qual Boll e Wäsch [Boll96a] denominam de *plano de apresentação*, contém referências aos reais objetos de mídia, além de informações adicionais acerca das restrições específicas (por exemplo, taxa de recuperação) das mídias envolvidas.

Desse modo, é sensato que se armazene, separadamente, os objetos de mídia e os dados de estruturação das composições multimídia. A cada objeto de mídia que compõe um documento, o plano de apresentação associa um único identificador e uma referência para a sua real localização (num repositório adequado). O identificador estabelece uma relação lógica entre o objeto e o documento, tornando-os fisicamente independentes. Essa separação evita redundâncias e permite a reusabilidade dos objetos de mídia, os quais podem estar logicamente atrelados a diferentes composições, sob características e qualidades distintas de apresentação (por exemplo, a resolução — número de *pixels* ou cores — de uma imagem pode variar de acordo com o aparato de exibição a ser usado quando da fase de apresentação).

Contudo, essa separação lógica entre os documentos e os objetos de mídia pode introduzir problemas concernentes à consistência das informações guardadas no plano de apresentação. Se um objeto de mídia for removido do repositório, todas as referências a ele não serão mais válidas. Isso acarreta na necessidade de se construir um sistema de informações multimídia *integrado*, composto por camadas de serviços

especializados, responsáveis pelas fases de autoria/apresentação, estruturação das composições e armazenagem/recuperação dos componentes de mídia envolvidos.

Esta é a filosofia que está por trás da proposta do Banco de Dados Multiware. O cenário reproduzido na Figura 5.1 mostra o ambiente dessa abordagem, onde se encaixa o SGPOM. Como indicado, a responsabilidade funcional desse serviço é atender, prontamente, a requisições de registro, armazenagem, recuperação e remoção de objetos de mídia, via uma interface pública (API) acessível a partir de uma implementação de ORB convencional. Nesse sentido, o SGPOM alivia os outros componentes Multiware da tarefa operacional de manipulação de grandes volumes de dados, liberando-os das sobrecargas de interatividade para com os repositórios. Um fluxo típico mostrando as interações entre os componentes é dado a seguir:

1. Num primeiro momento, um serviço de autoria [Tamb97] assiste os usuários na criação de apresentações completas, as quais capturam o conteúdo e a informação estrutural de documentos multimídia, interagindo com um Banco de Dados Orientado a Objetos apropriado [Janz]. Este, por sua vez, fica responsável pela geração dos esquemas de representação.
2. Os esquemas provêem o mapeamento lógico entre o documento multimídia gerado e os verdadeiros objetos de mídia simples, armazenados nos repositórios gerenciados pelo SGPOM.
3. Numa fase posterior, os objetos de mídia compondo o plano de apresentação podem ser, simultaneamente, recuperados, sendo preservadas as taxas apropriadas de sincronização intra e intermídia.
4. O serviço de apresentação pode, outrossim, se servir do SGPOM para guardar informações acerca de uma apresentação em curso, principalmente quando oferece suporte à interatividade para com o usuário ou quando os objetos de mídia precisam ser alterados.

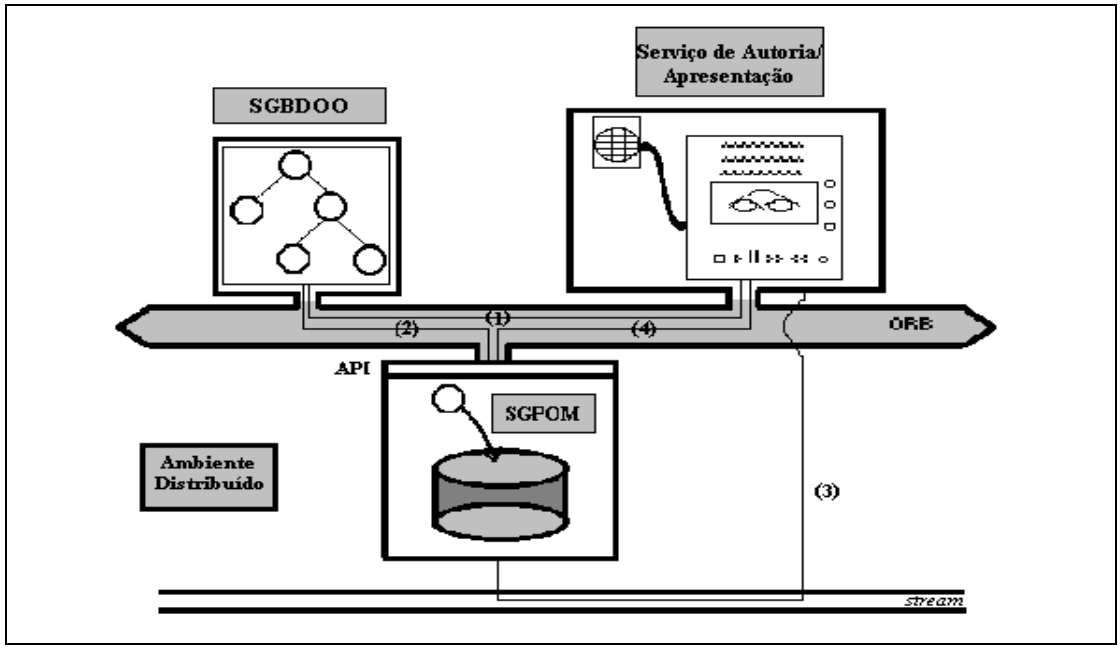

***Figura 5.1**: O ambiente do Banco de Dados Multiware e as interações entre seus componentes.*

5.2.2 Decisões de Projeto

Quando da concepção formal do SGPOM, foram assumidas como basais as seguintes regras de projeto e de utilização do serviço:

- Uso da arquitetura CORBA como alicerce de comunicação e de distribuição dos objetos que compõem a estrutura de gerência no acesso aos repositórios, com enfoque sobre o conjunto de transparências contemplado pelo RM-ODP.
- Pautar-se na especificação do POS como pano de fundo estrutural, abordando (estendendo/enriquecendo e modificando) as funcionalidades dos seus componentes, no que tange às restrições multimídia impostas.
- Os OMs a serem manipulados pelo serviço se restringem a simples objetos de mídia, os quais encapsulam dados (estruturados ou na forma de *streams*) codificados sob um dos diferentes formatos padrão existentes. Desse modo, não "apetece" ao SGPOM conhecer a origem desses objetos, nem como se relacionam temporal ou espacialmente dentro de uma apresentação. Tais objetos não precisam seguir o Modelo de Objetos Básico da OMA — isto é, não necessitam anunciar ou herdar interfaces IDL —, tampouco suportarem campos (métodos ou atributos) específicos, para a sua adequada condução até o repositório.
- Fica a cargo da aplicação-cliente a responsabilidade pelo fluxo adequado de invocações ao serviço e pela manutenção consistente dos estados dos seus objetos multimídia. O contrato de serviço oferecido compreende um conjunto de cinco operações de E/S (*Register()*, *Comm_way_Store()*, *Store()*, *Restore()* e *Delete()*) dedicadas às tarefas de gerência e acesso aos dados.
- Os repositórios multimídia têm autonomia para funcionarem independentemente do serviço. Assim, não se pode conjeturar que o acesso aos objetos se dê necessariamente via o SGPOM; isso é uma decisão de projeto circunscrita aos servidores de dados.

5.2.3 Decisões de Implementação

A partir do exposto na seção anterior, o modelo de objetos OMT esboçado na Figura 5.2 mostra o SGPOM sob o ponto de vista da aplicação-cliente (de autoria, de apresentação ou qualquer outro serviço). Tal aplicação, consoante essa visão, manipula um ou mais objetos multimídia, os quais têm associados a si identificadores exclusivos que os localizam nos repositórios gerenciados pelo SGPOM. Esses identificadores (SGPOM_PIDs) representam uma versão modificada do componente PID do POS.

Para os propósitos do SGPOM, um SGPOM_PID é definido como uma estrutura (tipo *struct*) descrita em IDL, contendo quatro campos de informação: um nome lógico do objeto, um atributo que identifica a aplicação, um caminho formado por alguns dos componentes de gerência da hierarquia do SGPOM e um *template* que localiza o estado do objeto no âmbito do repositório. O par formado pelos dois primeiros campos identifica unicamente um objeto e é usado na geração do *template*, cuja estrutura (mapeada para um tipo *Any* em IDL) é dependente do tipo de repositório (relacional, orientado a objetos, proprietário...). O caminho lógico, por sua vez, identifica um dentre os vários repositórios administrados pelo SGPOM.

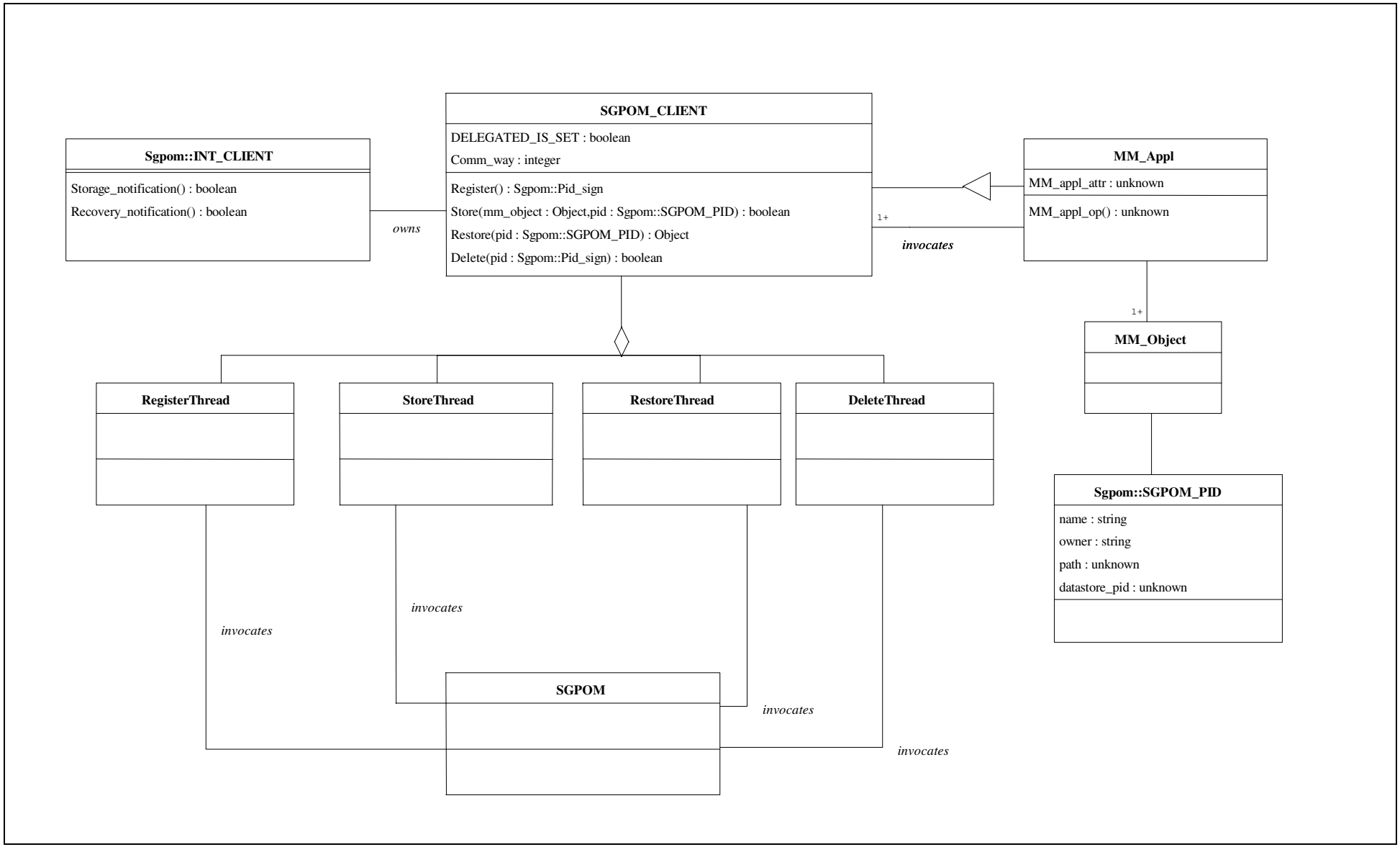

***Figura 5.2**: Modelo de Objetos (OMT) ressaltando a visão do cliente sobre o serviço (SGPOM).*

Visando tornar mais transparente o acesso às funcionalidades do SGPOM — principalmente para aplicações que não têm acesso direto a um ORB —, foram especificadas, em Java, classes adicionais contendo a infra-estrutura necessária e básica para a realização das invocações de E/S e para o conseqüente tratamento dos valores de retorno ou de exceções. No topo dessa camada, a qual compreende uma API opcional de mais alto nível, se encontra a classe SGPOM_CLIENT. Essa classe pode representar, para a aplicação multimídia, a única fronteira visível para se lançar mão dos serviços do SGPOM. O seu encargo é o de atuar como ponte de ligação entre o ambiente local e o distribuído. O desígnio é o de que a aplicação crie uma ou mais instâncias dessa classe, ou mesmo herde as suas funcionalidades, quando da necessidade de interação com um ou mais repositórios para a manipulação de um ou mais objetos de mídia. Esses OMs, por sua vez, devem ser modelados e implementados como objetos Java (representados, no diagrama, pela classe raiz da hierarquia — *Object*), contendo a mídia encapsulada sob um dos díspares formatos de codificação disponíveis (AVI, GIF...).

Um objeto SGPOM_CLIENT é modelado como uma agregação de quatro *threads*, as quais garantem a possibilidade de um *pool* de invocações concorrentes às operações de E/S providas pelo componente de mais alto nível do SGPOM, o SGPOM_PO, não mostrado no diagrama. Essas *threads* ficam bloqueadas a esperar pelos resultados provenientes de suas invocações.

Finalmente, a única interface em IDL passível de ser usada pela aplicação-cliente é INT_CLIENT, a qual, juntamente com o SGPOM_PID e outras construções básicas para o serviço, está definida num módulo à parte designado de *Sgpom*. Essa interface de *callback*[1], cuja implementação também compõe a classe SGPOM_CLIENT, autoriza a aplicação multimídia cliente a servir-se de um recurso incremental — baseado no uso de *operações delegadas* (discutidas em seção adjacente a esta) —, oferecido pelo SGPOM_PO. Pode-se recorrer a esse subterfúgio quando da impossibilidade de conclusão normal (geração de exceção) das requisições de armazenagem e recuperação do OM propagadas pelos níveis de gerência (componentes) do SGPOM. Mediante invocações às operações dessa interface (INT_CLIENT), o SGPOM_PO pode assumir o papel de representante da aplicação-cliente, notificando-a acerca do êxito ou insucesso das tentativas posteriores de realização dessas tarefas, as quais serão iniciadas por sua própria conta.

5.3 Perspectiva Estrutural

A Figura 5.3 traz a proposta da arquitetura do SGPOM. Ao longo desta seção, serão descritos, por meio do conjunto de interfaces e suas inter-relações, os componentes desse *framework* de objetos distribuídos, comparando-o, quando relevante, àquele proposto pela especificação do POS.

---

[1] Uma interface de *callback* possibilita a um objeto cliente implementar algumas das funcionalidades comumente associadas a um objeto servidor. Isso garante um mecanismo assíncrono pelo qual um cliente pode ser notificado acerca da ocorrência de algum evento importante no lado do servidor.

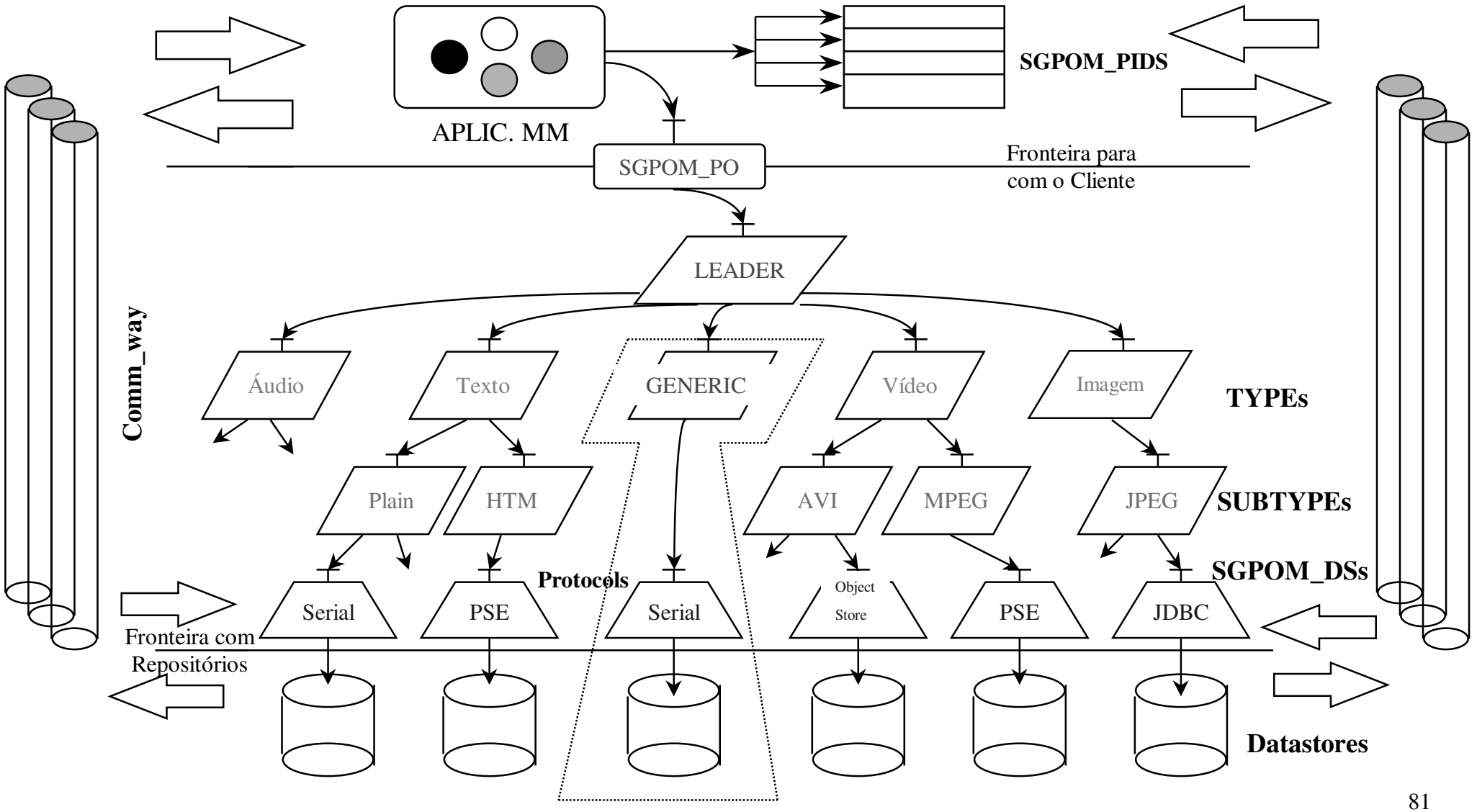

***Figura 5.3**: Arquitetura do SGPOM.*

5.3.1 Relato dos Componentes

A arquitetura do SGPOM é calcada sobre os seguintes componentes:

- ***Persistent Identifier* (SGPOM_PID)**: o qual descreve a localização dos dados de um objeto multimídia persistente em alguma fonte de armazenagem (*Datastore*), gerando uma estrutura de identificação para esses dados.
- ***Persistent Object* (SGPOM_PO)**: apresenta-se, para o SGPOM, como o elemento de mais alto nível da estrutura. Ao contrário do PO (POS), não é uma interface a ser herdada por um objeto CORBA. Comporta-se como a fronteira entre o serviço e as aplicações-cliente, preservando, para os outros componentes da hierarquia do SGPOM, a semântica (papel) de um objeto persistente. Para cada instância do serviço, apenas um SGPOM_PO está disponível, o qual municia um mecanismo adicional (operações delegadas) de suporte à armazenagem e à recuperação do OM.
- ***Persistent Object Managers* (SGPOM_OMs)**: um conjunto expansível (em forma de árvore) de objetos distribuídos, responsáveis pelo encaminhamento (roteamento) das requisições de E/S do SGPOM_PO ao SGPOM_DS — o qual mantém uma interface para um repositório multimídia — adequado. Esse grupo estende o conceito e as funcionalidades do POM do POS, de tal maneira a incluir (incorporar) os diversos tipos e formatos padronizados de mídia.
- ***Persistent Data Services* (SGPOM_DSs)**: oferece uma interface uniforme para qualquer combinação de *Datastore*, *Protocol* e *Comm_way*, coordenando as operações de persistência de mais baixo nível para um dado OM. Parte do SGPOM_PID descreve o subconjunto de componentes responsável pelo caminho lógico de identificação do repositório (ramo da árvore formado pelos SGPOM_OMs e pelo SGPOM_DS quando da operação de registro).
- ***Protocol***: em consonância com a idéia de que um OM não apresenta um estado definido (é um simples invólucro de dados multimídia), este componente abstrato apresenta uma conotação distinta daquela do seu correspondente no POS. Para o escopo do SGPOM, ele somente identifica qual é o tipo (modelo relacional, OO, sistema de arquivos...) ou produto (PSE, ObjectStore, JDBC...) usado na construção do repositório multimídia.
- ***Comm_way***: descreve qual mecanismo (protocolo ou serviço) de comunicação será usado para a transmissão (transporte) conveniente do OM.
- ***Datastore***: analogamente ao POS, este componente fornece um dos diversos modos de se armazenar os dados de um objeto, isso feito independentemente do espaço de endereçamento que o contém. Serve como abstração do repositório.

5.3.2 Funcionalidades

O SGPOM estende as funcionalidades do componente POM do POS, na medida em que cria um arranjo hierárquico (*top-down*) de gerentes especializados, relacionados aos tipos de mídia levantados na Seção 3.2. Como observado anteriormente (Subseção 4.3.6), o POM assume um papel central, mas limitado, de ligação entre o PO e os elementos de mais baixo nível (PDSs); porquanto tornou-se necessária uma nova configuração de gerência (a ser facilmente incrementada),

formada por camadas de elementos cooperantes, que pudesse atender a múltiplas requisições concorrentes para a gerência e o acesso aos OMs.

O SGPOM_PO é o único componente criado para ser visível aos clientes do SGPOM. Pode receber invocações provenientes de aplicações ou outros serviços que tenham acesso a um ORB, ou, indiretamente, por meio da classe SGPOM_CLIENT (Subseção 5.2.3). Por outro lado, o conjunto de SGPOM_OMs tem como raiz a figura do LEADER, que se comporta como o único servidor CORBA para o qual são repassadas as operações de E/S reestruturadas pelo SGPOM_PO.

Como todo gerente da hierarquia, o LEADER mantém informações acerca dos seus sucessores, no caso, os TYPEs. Estes últimos, por sua vez, compõem um nível relativo à miríade de tipos de mídia existentes (texto, áudio, imagem, vídeo...), encaminhando as requisições para os SUBTYPEs — um TYPE pode estar associado a um ou mais SUBTYPEs, os quais são introduzidos na estrutura para representarem formatos diferentes de codificação para cada mídia.

Na última camada, aparecem os SGPOM_DSs, responsáveis que são pela interface para com os repositórios multimídia. Um SUBTYPE pode manter um *link* simbólico de ligação com um ou mais SGPOM_DSs. Porque converte as operações de E/S para a API (ou linguagem de consulta e recuperação) definida pelo *Datastore*, as funcionalidades de um SGPOM_DS se assemelham a de um *driver*, o qual deve ser, pelo menos em parte, implementado pelo projetista do repositório. Dentre os componentes sob essa classificação, se encontra o GENERIC, o qual é empregado para o tratamento de dados (objetos) que fogem à classificação em pares de tipos e subtipos (*raw data*), servindo, alternativamente, como um adido para a manipulação dos estados dos outros componentes da arquitetura. O GENERIC — juntamente com o SGPOM_PO e o LEADER — integra a configuração padrão (mínima) para uma instância do SGPOM. Por essa razão, é classificado também como um SGPOM_OM.

A Figura 5.4 traz um novo cenário (modelo estático OMT) para o SGPOM, focando, agora, sobre os seus componentes estruturais. Esse diagrama é o resultado direto do processo de especificação computacional do serviço (segundo o ponto de vista de computação do RM-ODP), definindo a decomposição do sistema em estruturas adequadas para a distribuição de funções. As entidades *Coordinator* e *Manager_client* serão abordadas na subseção que se segue.

5.3.3 Decisões de Projeto

Os tópicos que seguem estão destinados à discussão da lista de ponderações de projeto mais relevantes tomadas quando do planejamento da estrutura do SGPOM.

Suporte a cinco operações básicas de E/S

São divididas em dois grupos. O primeiro encerra as *operações de gerência*, *Register()* e *Delete()*, as quais não implicam na manipulação direta do OM. Sob a segunda categoria, estão as *operações de acesso*, *Comm_way_Store(), Store()* e *Restore()*, que acarretam no uso de um meio de comunicação — afora o ORB — para o devido transporte do OM. De forma análoga ao POS, essas operações são

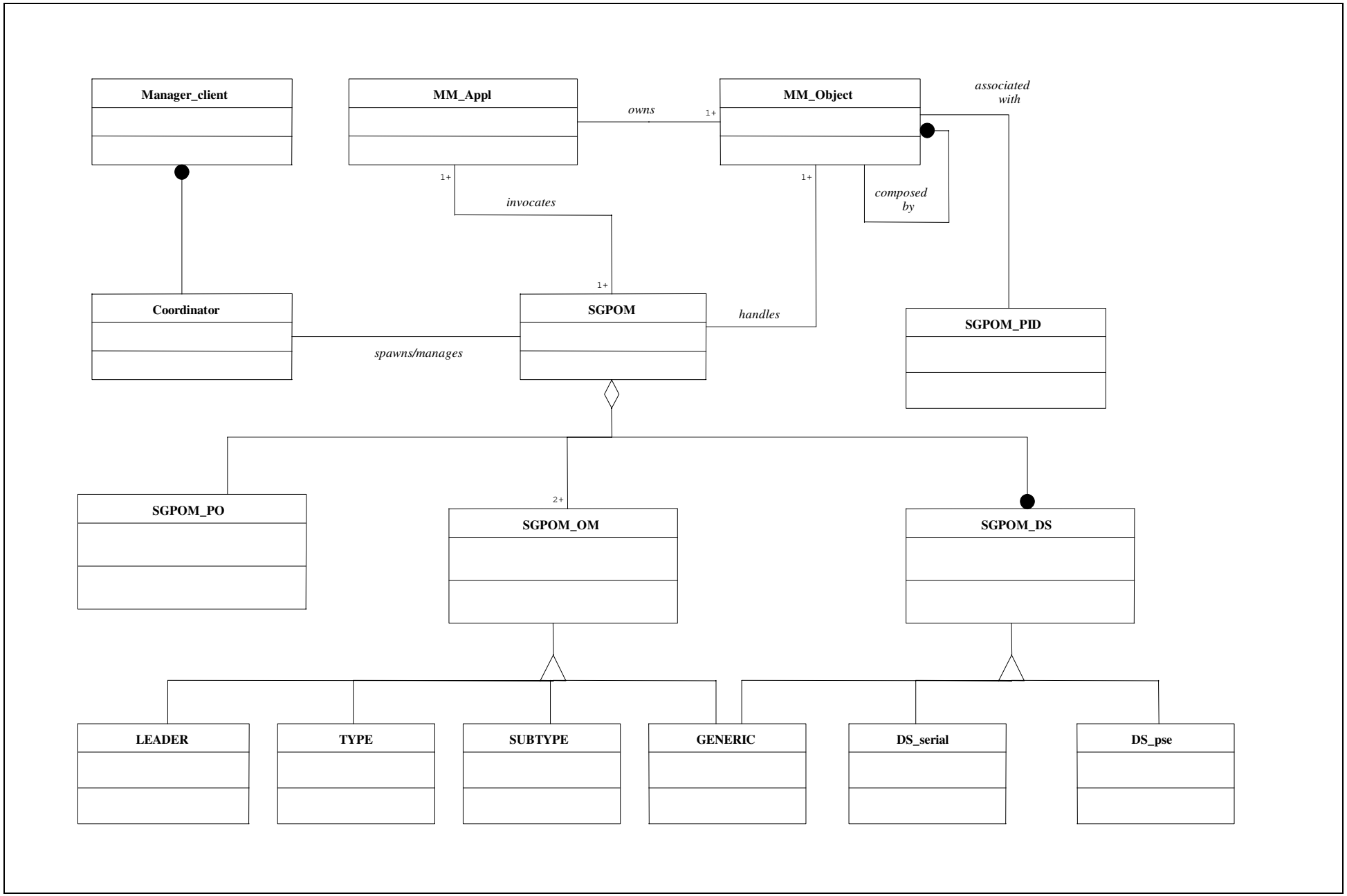

***Figura 5.4**: Modelo de Objetos (OMT) mostrando os componentes estruturais do SGPOM.*

anunciadas em todas as camadas da hierarquia (do SGPOM_PO ao SGPOM_DS, passando pelos três tipos de SGPOM_OMs), com assinaturas semelhantes.

Concordando com o exposto nas Subseções 4.3.3 e 4.3.6, isto é, que a semântica por trás das operações *connect()* e *disconnect()* do POS não está bem especificada, propôs-se uma outra operação a ser usada quando do registro do OM junto ao repositório que irá armazená-lo. Essa operação, *Register()*, gerará como resultado um SGPOM_PID[2], o qual descreverá, além do ramo lógico da estrutura (desde o LEADER até o SGPOM_DS) que determina o *Datastore*, uma chave (*template*) que aponta para a real localização do objeto dentro do repositório. Esse identificador (SGPOM_PID) é a única informação a ser guardada pela aplicação-cliente (ou propagada a outros serviços ou aplicações) para que possam ser realizadas as operações de acesso ao OM. Nessa fase de subscrição, a aplicação-cliente deve indicar o par de *strings* (relativas ao tipo/subtipo) que descreve a mídia e o formato de codificação do OM, além do protocolo a ser suportado pelo repositório. Tal protocolo define um contrato, entre a aplicação e o *Datastore*, acerca de quais funcionalidades o OM deve suportar (por exemplo, herdar de uma interface ou classe) para que este possa ser convenientemente manipulado.

A operação *store()* do POS foi mapeada em duas no SGPOM, *Comm_way_Store()* e *Store()*. A primeira retorna um objeto *Comm_way*, cuja interface IDL é exibida no módulo *Sgpom*. Essa interface não define atributos nem operações[3] e deverá ser estendida a fim de incorporar os diversos protocolos ou mecanismos de comunicação (*sockets*, RTP, Canal ODP...) suportados pelos SGPOM_DSs. Para o caso de *socket*, é proposta uma especialização de *Comm_way*, conhecida como *Socketport_host* (também publicada no módulo *Sgpom*), a qual estabelece dois campos de informação, quais sejam a porta e a máquina do servidor de dados (*streams*). Por outro lado, a operação *Store()* deve ser invocada após a aplicação-cliente ter repassado integralmente o objeto multimídia através do substrato de comunicação. Retorna, pois, um valor *booleano* indicando o resultado (sucesso ou não) acerca da efetiva armazenagem do OM por parte do *Datastore*. Este foi o meio encontrado — definir duas em vez de uma operação — para que se assegure à aplicação-cliente que o OM não seja corrompido no seu trajeto ao repositório.

A operação *Restore()* é empregada para o resgate do OM. Da mesma forma que *Comm_way_Store()*, retorna um objeto *Comm_way* usado para a transferência do fluxo de dados — agora no sentido inverso — que compõe o OM. A tarefa de recuperação, ao contrário da de armazenagem, apresenta restrições temporais mais significativas, uma vez que está diretamente relacionada à atividade de apresentação (ou consumo) dos dados dos objetos. Caso os dados sejam perdidos (corrompidos) durante a transmissão, cabe à aplicação-cliente a decisão de interromper a comunicação e reiniciar a operação — o que implica num novo ciclo de requisições de *Restore()* entre os componentes da hierarquia.

Almejando diferençar — resolver o conflito entre — o operador *delete* (palavra reservada de C++) e a operação de remoção do OM, a assinatura desta última passou a ser *Delete()*. A sua semântica, entretanto, continua a mesma daquela correspondente

---

[2] Encapsulado num *Pid_sign*.

[3] A esse tipo de interface costuma-se denominar de simbólica (*marker interface*).

do POS: extinguir quaisquer dados (o fluxo mais os metadados, caso haja algum) relativos ao OM, após o que, perderá o sentido lógico a realização das operações de acesso (para atualização/rearmazenagem ou recuperação do objeto). Para esse fim, deverão ser repassadas as etapas de registro e armazenagem.

Uma estrutura adicional — também definida no módulo *Sgpom* —, chamada de *Pid_sign*, é usada com o intuito de se prover um certo grau de proteção ao OM. Um *Pid_sign*, retornado quando da fase de registro, encapsula um SGPOM_PID e uma senha (assinatura), a qual deve ser mantida (e não propagada) pela aplicação-cliente. Quando da remoção dos dados, será verificada a consistência dessa senha, comprovando-se, via esse artifício, a identidade da aplicação à qual pertence o OM (autenticação).

Subcomponentes

A partir das ilustrações que sucedem (Figuras 5.5 a 5.7), constata-se que cada um dos três tipos de elementos da hierarquia do SGPOM (SGPOM_PO, SGPOM_OMs e SGPOM_DSs) abstrai, na verdade, uma camada composta por quatro subcomponentes com interfaces IDL bem delineadas, sendo eles:

1. **Objeto Fábrica (*Factory*)**: objeto CORBA em execução constante (ativação persistente) responsável pela criação/recriação, no mesmo processo, das instâncias dos outros integrantes da camada — à exceção dos objetos *Filhos*. Segue o conceito proposto pelo Serviço de Ciclo de Vida do OMG (Subseção 2.2.5).
2. **Objeto Controlador (*Control*)**: responsável pela tarefa de monitoração local de desempenho no atendimento às requisições de E/S endereçadas ao subcomponente *Central* da camada, com o qual está univocamente (relação de um-para-um) vinculado. A medida de controle é feita pela coleta intermitente de uma lista (extensível) de parâmetros especificados, dentre eles: *carga* (número de invocações de gerência ou de acesso recebidas desde a instanciação do objeto central), *taxa de invocações* (carga por intervalo de tempo) e *tempo médio de atendimento* (média dos intervalos em que os objetos Filhos ficam bloqueados à espera dos resultados das operações de acesso propagadas para a camada sucessiva). Outra função deste elemento é o de interagir com o objeto Coordenador — explicado no próximo tópico —, por meio de invocações assíncronas (tipo *oneway*), para identificação de falhas, sobrecarga ou inconsistências.
3. **Objeto Central (*Component*)**: pela acepção da definição, depreende-se que é o elemento mais importante da camada, assumindo o nome desta (SGPOM_XX, onde XX={PO, OM e DS}[4]). Quando da sua instanciação (via a Fábrica), é criado também um objeto Controlador associado. Cada objeto sob esta classificação está relacionado unicamente a uma instância de SGPOM. De um outro modo: a Fábrica pode ter, sob o mesmo processo, um ou mais objetos deste tipo pertencentes a versões (instâncias) distintas do serviço (SGPOM).
4. **Objetos Filhos**: para que um SGPOM_XX possa atender a múltiplas requisições (concorrentes) de E/S —procedentes, provavelmente, de mais de uma aplicação-cliente —, optou-se pela introdução, na arquitetura de cada camada, de um *pool* de objetos Filhos (*Childs*), a serem implementados como *threads* (*lightweight processes*). Esse conjunto deve ser criado pelo objeto Central (o qual repassa, a cada um da prole, uma referência — ou ponteiro — para si próprio) e não pela

[4] Deste modo, quando se passa a mencionar "Referência ao LEADER", por exemplo, está-se aludindo a *Object Reference* ao objeto Central da camada LEADER.

Fábrica. Os Filhos são os responsáveis diretos pelo tratamento e encaminhamento (a seus correlatos da camada subseqüente) das requisições bloqueantes de gerência e acesso, incrementando os valores de monitoração, mantidos como parte do estado do objeto Central.

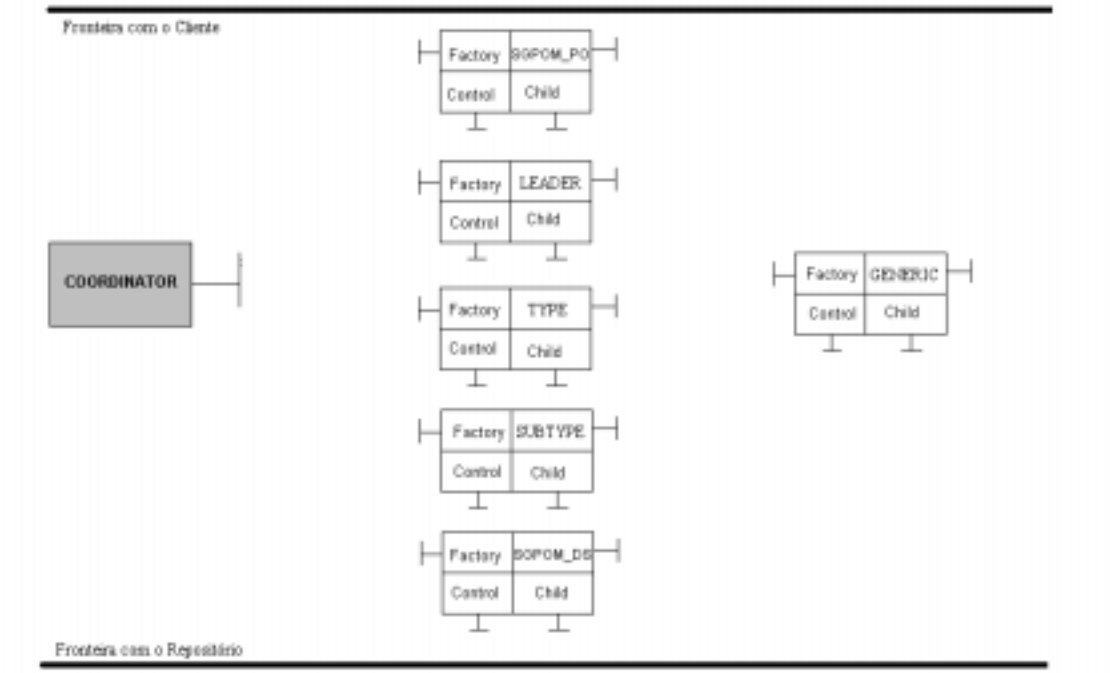

***Figura 5.5**: Hierarquia em camadas (componentes) do SGPOM.*

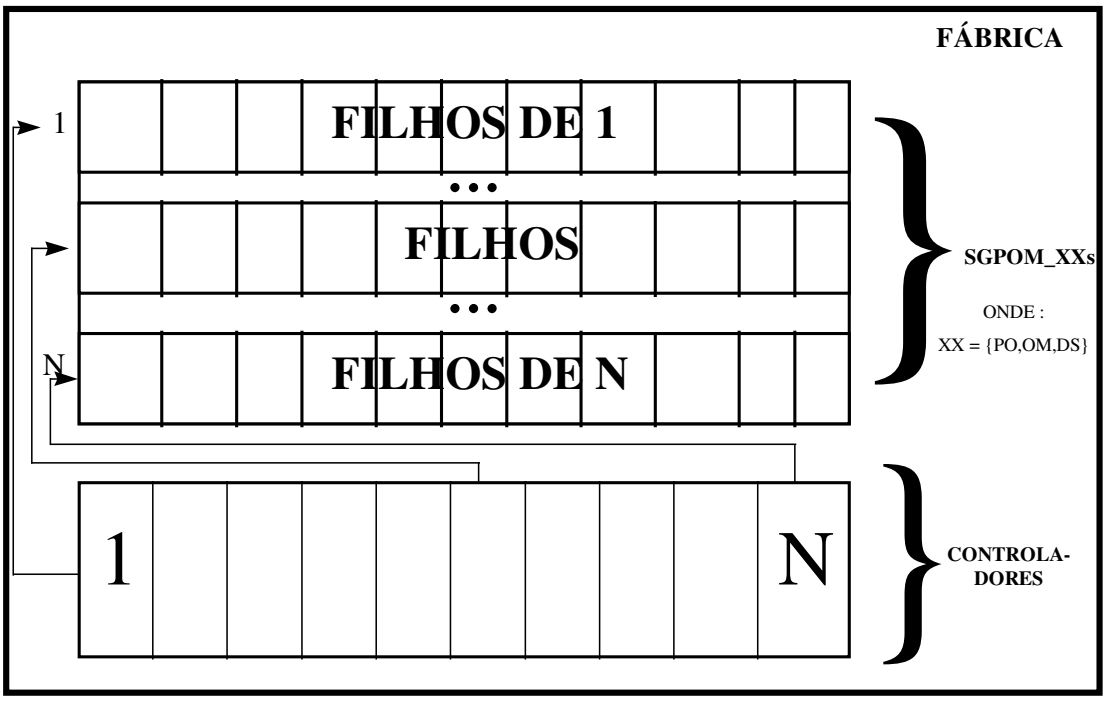

***Figura 5.6**: Subcomponentes de cada camada do SGPOM.*

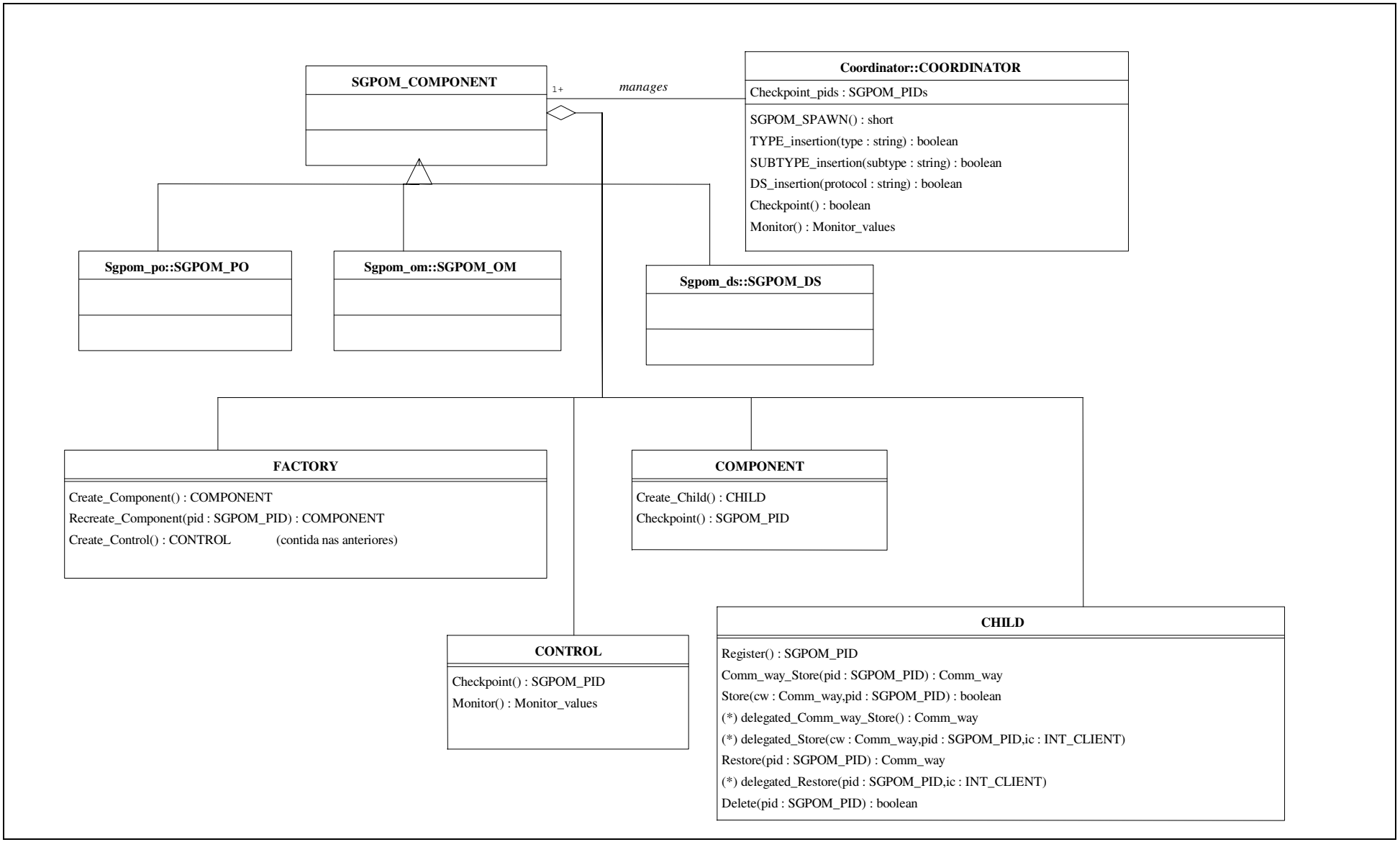

Figura 5.7: Modelo de Objetos (OMT) para o SGPOM enaltecendo os subcomponentes e o elemento de coordenação da hierarquia.

Coordenação dos componentes

Por ser uma arquitetura de especificação aberta, o POS não explicita (ou impõe) como os objetos devam ser coordenados; isso fica a cargo dos projetistas das implementações do serviço. Para os anseios do SGPOM, um objeto CORBA à parte (fora da hierarquia e com seu servidor próprio — vide Figuras 5.5 e 5.7), denominado de *Coordenador*, foi designado para a função de *orquestração* dos componentes (na verdade, de seus subcomponentes), o que se traduz em três conjuntos de operações:

1. **Instanciação**: para a criação/recriação dos objetos Central e Controlador de cada camada.
2. **Monitoração/Notificação**: para coleta geral das medidas de desempenho e para recebimento de informações excepcionais acerca dos elementos SGPOM_XXs.
3. ***Checkpoint***: para forçar os subcomponentes Centrais a salvarem os seus próprios estados de execução no componente GENERIC.

Uma ou mais versões do SGPOM podem ser instanciadas e incrementadas, atendendo, pois, às demandas de diferentes domínios de aplicações multimídia. Para possibilitar uma forma integrada e consistente de criação dos subcomponentes de cada camada, o Coordenador exporta, em sua interface IDL, quatro operações de instanciação, quais sejam: *SGPOM_SPAWN()*, *SGPOM_OM_TYPE_insertion()*, *SGPOM_OM_SUBTYPE_insertion()* e *SGPOM_DS_insertion()*. A primeira é usada por uma aplicação (ou *applet*) de administração — representada na Figura 5.4 como *Manager_client* — para "colocar no ar" uma instância padrão (configuração básica) de SGPOM. As demais serão usadas pelo usuário administrador do serviço para inserir (incrementar) na estrutura os gerentes de tipo (de mídia) e subtipo (formato) e os *drivers* de mais baixo nível, de acordo com os repositórios multimídia a serem anexados à hierarquia.

A *configuração básica* (SGPOM_PO, LEADER e GENERIC) provê as funcionalidades mínimas de uso do SGPOM. À proporção que novas mídias devam ser suportadas, novos gerentes de tipo (TYPE) serão incluídos pelo administrador. Quando da sua instanciação, um TYPE recebe o nome (*string*) da mídia da qual será representante, um número identificando a instância de SGPOM à qual pertence e duas Referências a Objetos. A primeira para o subcomponente Central da camada superior (no caso, o LEADER) e a segunda para o GENERIC. Com a primeira *Object Reference*, o TYPE subscreve ao LEADER (via uma operação apropriada deste), noticiando-o sobre o seu tipo associado e repassando-lhe a sua própria Referência. Com isso, o LEADER manterá uma lista de Referências a todos os TYPEs existentes, roteando as invocações de E/S apropriadamente. É interessante observar que, desse modo, um ou mais TYPEs podem estar associados a um mesmo tipo de mídia — isso fica a critério do administrador —, formando braços (ramos) distintos da hierarquia e levando a um balanceamento de carga entre os repositórios.

A consideração exposta no parágrafo anterior se aplica, dentro das proporções, aos SUBTYPEs e aos SGPOM_DSs, o que leva, possivelmente, a diversas *configurações estendidas*[5] do mesmo serviço (SGPOM). A regra é: um SUBTYPE está associado unicamente a um TYPE e um SGPOM_DS, a um SUBTYPE. Essas associações lógicas ficarão a cargo do administrador quando da instanciação de cada

[5] Qualquer configuração (arranjo das camadas) diferente da básica é considerada estendida.

novo componente e definem a topologia da hierarquia. O Coordenador, assumindo o papel de intermediário e "regente" de orquestração, mantém como parte do seu próprio estado listas (*arrays*) contendo todas as Referências a objetos Controladores instanciados. Cada lista está associada a uma versão do serviço. Como cada Controlador tem um atributo que identifica o seu correspondente objeto Central, o acesso a este último fica sendo viável, indiretamente, para o Coordenador.

A Figura 5.8 mostra, simultaneamente, dois cenários. O primeiro diz respeito à instanciação de um novo SGPOM_PO (e, por conseguinte, do Controlador e dos Filhos), enquanto o segundo se refere à criação de um TYPE (podendo, todavia, ser aplicado a qualquer um dos outros componentes da hierarquia). Para os dois cenários, temos os seguintes passos em comum:

1. O Coordenador, ao receber uma invocação (por parte de um *Manager_client*) a uma de suas operações de instanciação, envia uma requisição de criação — no caso, *Create_PO()* e *Create_OM_Type()* — à Fábrica apropriada (1a). Esta, por sua vez, instancia o objeto Central — SGPOM_PO e TYPE — e o Controlador (1b), retornando a Referência deste último (1c).
2. O Coordenador, de posse da *Object Reference* do Controlador, invoca uma operação sua pela qual passa alguns parâmetros importantes (Referências a outros objetos da hierarquia). No caso do SGPOM_PO_CONTROL, passam-se as Referências para o LEADER e para o GENERIC. Para o TYPE_CONTROL, passam-se as Referências para o LEADER (seu antecessor) e para o GENERIC.
3. O Controlador, detendo as Referências, repassa-as ao objeto Central (3a) — cada Referência está relacionada a um atributo IDL deste último —, o qual, em seguida, pode se registrar junto ao GENERIC (3b), recebendo um SGPOM_PID (3c) para futuras invocações de *checkpoint* (explicação a seguir).

Para o SGPOM_PO, em particular, o terceiro passo apresenta uma subdivisão a mais (3d), feita logo após aquela assinalada como 3a: o Controlador notifica a Fábrica (SGPOM_PO_FACTORY) que o objeto Central está pronto para receber invocações de E/S por parte das aplicações-cliente. Estas últimas, por outro lado, terão acesso, a partir de uma operação apropriada (*GetPo_reference()*) da Fábrica, às Referências para os SGPOM_POs existentes, podendo iniciar os ciclos de operações de gerência e de acesso a seus OMs.

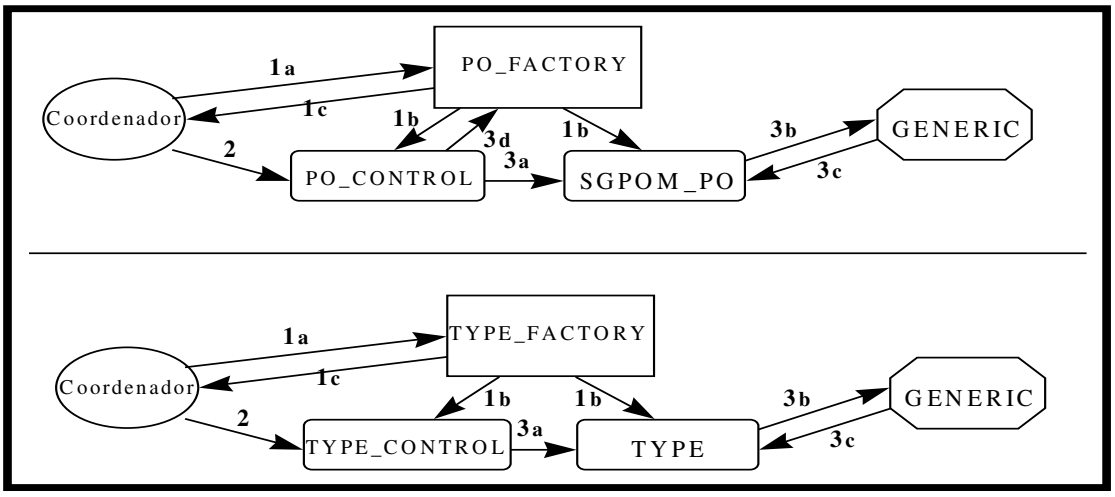

***Figura 5.8**: Fluxo de invocações para instanciação do SGPOM_PO e do TYPE.*

O Coordenador oferece ao(s) administrador(es) do serviço um meio rápido de se analisar o comportamento das camadas de cada uma das instâncias existentes. Para tanto, é definido, em sua interface IDL, um conjunto de estruturas (*structs*) e seqüências (*sequences*) contendo informações relativas aos objetos Centrais de cada nível. Tais informações podem ser divididas em duas classes, as qualificativas e as de monitoração. Informações qualificativas indicam, tipicamente, o índice relativo à instância do serviço a que pertence o componente, a data de sua criação, o seu índice dentro da arquitetura e um rótulo — os dois últimos dados se aplicam somente aos TYPEs, SUBTYPEs e SGPOM_DSs. O rótulo é mapeado de acordo com o componente: para o TYPE, é a *string* relacionada ao tipo da mídia; para o SUBTYPE, é a *string* correspondente ao formato de codificação; e para o SGPOM_DS, é a *string* relativa ao protocolo associado.

Informações de monitoração são coletadas, freqüentemente, sob a forma de uma estrutura especial (definida, no módulo *Sgpom*, como *Monitor_values*), por *threads* específicas pertencentes ao Coordenador. Cada *thread*, após um certo período, fica responsável por estabelecer um novo diálogo com um objeto Controlador, recuperando daí as medidas de desempenho. Para os TYPEs, SUBTYPEs e SGPOM_DSs, esses valores são posteriormente agrupados em conjuntos, facilitando o seu resgate. Fica a critério da aplicação de administração (ou de seu usuário, caso seja interativa), recuperar, isolada ou coletivamente, as informações dos componentes de cada instância. O Coordenador oferece, para esse fim, um mosaico apropriado de operações de monitoração.

Reiterando o que foi anunciado anteriormente, o SGPOM oferece, mediante um mecanismo de notificações assíncronas, um certo suporte à tolerância a falhas de seus componentes. Isso é conseguido por intermédio dos objetos Controladores, os quais reportam ao Coordenador condições extraordinárias (falhas, sobrecarga, falta de responsividade...) acerca dos subcomponentes Centrais e do GENERIC. Cabe ao Coordenador, de acordo com a sua política de implementação, tomar as devidas providências: emitir mensagens de alerta ao terminal gráfico, criar novos subcomponentes ou substituí-los. Para fins de substituição, as fábricas exportam uma segunda operação de instanciação (*Recreate()*), a qual inclui um argumento adicional àqueles da operação convencional (*Create()*): o SGPOM_PID relativo à localização da última cópia do estado do objeto Central junto ao GENERIC .

Almejando manter a consistência entre o estado dinâmico atual — Referências a outros objetos da hierarquia e seus valores de monitoração — do objeto Central e sua cópia persistente mantida no GENERIC, o seu correspondente Controlador fica responsável, também, pela invocação periódica à sua operação de *Checkpoint()*. Esse período pode ser definido por uma constante (imutável) ou, em tempo de execução, por um parâmetro configurável do sistema. Tal operação (*Checkpoint()*) se assemelha àquela de armazenagem de um OM, na medida em que corresponde a um ciclo de interações composto pela invocação de duas operações para o lado do repositório (no caso, o GENERIC): *Comm_way_Store_Component()* e *Store_Component()*.

O Coordenador, por outro lado, expõe ao *Manager_client* uma operação de *checkpoint* geral, pela qual o administrador do serviço pode, a qualquer momento, requisitar a ativação das operações locais de persistência para os objetos Centrais. Isso

é feito, indiretamente, via os Controladores, os quais exportam ao Coordenador uma operação de mesmo nome (*Checkpoint()*) — ver Figura 5.9. O conjunto decorrente de SGPOM_PIDs fica disponível, no elemento de orquestração, a partir de um atributo IDL específico (do tipo *readonly*).

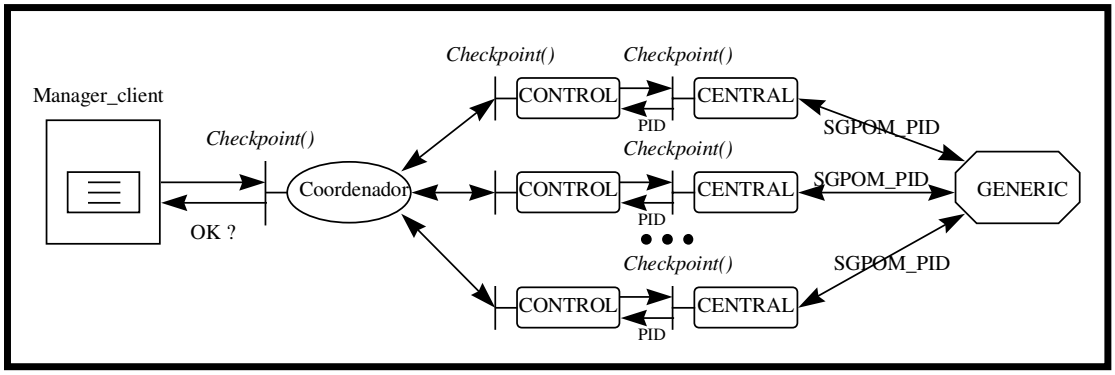

***Figura 5.9**: Fluxo de propagação das operações de checkpoint entre as entidades do SGPOM.*

A Figura 5.10 ilustra uma nova visão do SGPOM sob essa última perspectiva. O novo modelo estático OMT mostra, ao mesmo tempo, as relações entre as entidades e os papéis (*roles*) que elas assumem (cliente e servidor). Constata-se que todos os objetos da hierarquia do serviço mantêm uma relação com o GENERIC (*save state in*), simbolizando as atividades de *checkpoint*. O GENERIC é o único dos componentes que deve prover um suporte robusto (embutido) no tratamento a possíveis falhas internas. Desse modo, pode-se supor como contínua a sua existência (disponibilidade), o que elimina a necessidade de manutenção persistente do seu estado e da operação de reinstanciação na interface de sua Fábrica associada.

Hierarquia firmada no MIME

Quando da especificação do SGPOM, passou-se a avaliar alternativas para o suporte adequado à *concorrência* no acesso aos dados multimídia. A meta é a de prover a uma ou mais aplicações-cliente uma política equânime (sem benefício ou prejuízo) no tratamento a suas requisições de E/S. A saída encontrada foi a de expandir as funcionalidades do componente POM do POS, introduzindo novas camadas na estrutura. O resultado foi a inclusão dos conjuntos de TYPEs e de SUBTYPEs, para representarem, respectivamente, os tipos e os formatos de codificação, seguindo o modelo de classificação das mídias proposto pelo padrão MIME (Seção 3.3). Esses dois conjuntos são extensíveis, ou seja, podem ser incrementados dinamicamente, formando caminhos lógicos aos diversos repositórios a serem incorporados.

Um TYPE está associado a um ou mais SUBTYPEs; da mesma forma, a um SUBTYPE podem estar atrelados um ou mais SGPOM_DSs. Possivelmente, quando do registro de um novo OM, não exista à disposição um ramo TYPE–SUBTYPE–SGPOM_DS (que leva a um único repositório) que reflita a tríade <tipo, formato, protocolo> associada ao objeto. Sendo o caso, é encargo do LEADER repassar a invocação para o GENERIC, considerado como um repositório *default*, o qual ficará responsável pelas futuras operações de acesso. Entretanto, para se evitar problemas de desempenho no atendimento às requisições concorrentes de E/S, o número de objetos registrados junto ao GENERIC deve ser limitado.

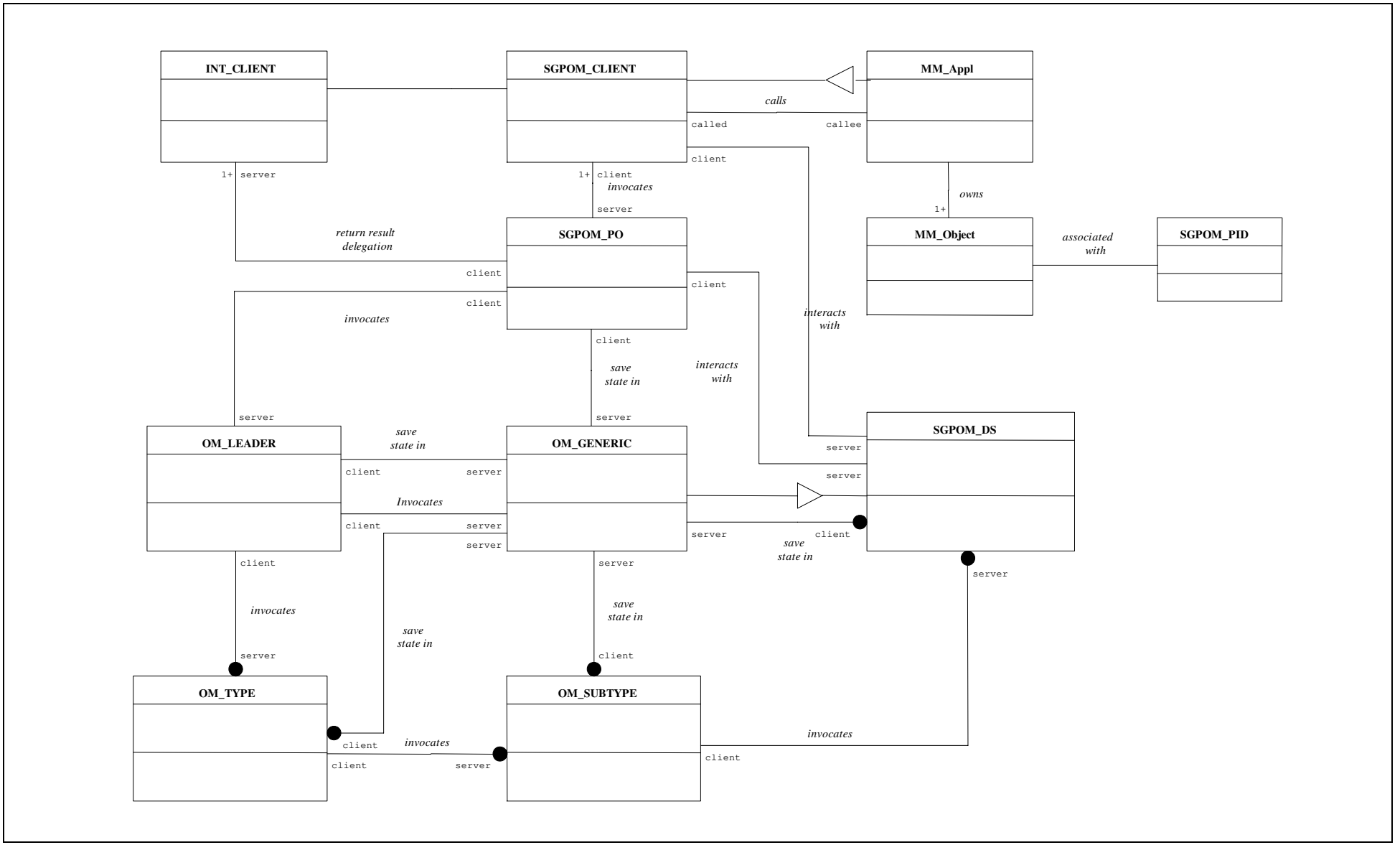

***Figura 5.10**: Modelo de Objetos (OMT) para o SGPOM ressaltando os papéis dos elementos em suas relações.*

Operações delegadas

Durante a trajetória (fluxo de propagações pelas camadas) de uma requisição de E/S, condições anormais podem ocorrer, ocasionando a geração de *exceções* (ver próximo item). No caso das operações de acesso (armazenagem e recuperação), é muitas vezes imprescindível — quando há flexibilidade nas restrições temporais — a existência de um mecanismo opcional que alivie as aplicações-cliente das tarefas de tratamento dessas exceções e no reinício do processo. Para esse fim, o SGPOM_PO provê, na interface IDL de seus Filhos, três operações alternativas às convencionais — *delegated_Comm_way_Store()*, *delegated_Store()* e *delegated_Restore()* — por meio das quais uma aplicação-cliente pode repassar-lhe (delegar) esse encargo.

Para lançar mão desse artifício, é exigido que o cliente provenha uma implementação da interface IDL de *callback* INT_CLIENT, por intermédio da qual será notificado, pelo SGPOM_PO, acerca do sucesso de suas próprias tentativas. Sob esse cenário, para a armazenagem do OM, é necessário, primeiramente, que este seja repassado ao SGPOM_PO, o qual o armazena num *buffer* interno. Posteriormente, o SGPOM_PO irá realizar o ciclo normal de operações de armazenagem (*Comm_way_Store()* e *Store()*), entregando o OM ao repositório. Num sentido inverso, para fins de recuperação, esse *buffer* é usado para guardar, temporariamente, a cópia resgatada do objeto junto ao repositório. O cliente, com a notificação de sucesso, receberá um objeto *Comm_way* a ser usado, nesse caso, para a transferência do OM do SGPOM_PO para si próprio.

Do enunciado até aqui, pode-se depreender o seguinte:

- É interessante, para fins de desempenho, que o SGPOM_PO e o cliente estejam num mesmo espaço de endereçamento ou num mesmo nó do sistema.
- O SGPOM_PO assume um papel híbrido: ora ele é cliente (do repositório e do SGPOM_DS associado), ora é servidor (para a aplicação-cliente).
- Só se deve recorrer a esse recurso de delegação quando de necessidades extraordinárias, haja vista não sobrecarregar o SGPOM_PO.

Uma ressalva: na Figura 5.7, a notação "(*)" destaca as operações delegadas, as quais devem ser suportadas, unicamente, pelos Filhos do SGPOM_PO. Embora seja um erro (ou abuso) de representação, optou-se, conscientemente, por esse subterfúgio, haja vista que preserva o principal enfoque daquele cenário, qual seja o de esboçar os subcomponentes de cada camada. A introdução de uma nova classe especializada de CHILD (SGPOM_PO_CHILD), com essas operações, constituiria o modelo adequado.

Conjunto de exceções

Na Subseção 4.3.6, observou-se a ausência no POS de uma coleção apropriada de exceções para o tratamento de situações anômalas que ocorressem em seus componentes. Quando do projeto do SGPOM, verificou-se que esse hiato poderia acarretar em inconsistências (efeitos colaterais) e em perda de desempenho, haja vista o aumento no número de camadas da hierarquia e a introdução do conjunto de subcomponentes em cada camada.

Todas as interfaces IDL definidas para o SGPOM — desde a INT_CLIENT até a do Coordenador, passando pelos subcomponentes — apresentam um tipo

exception (ver Subseção 2.2.1) em particular, associado a uma lista (tipo *enum*) de códigos indicando a razão (significado) da exceção gerada. O intento, com isso, é o de tornar factível tanto a localização exata da fonte geratriz como a ciência da causa da exceção, de tal modo a possibilitar o seu devido tratamento. As Tabelas 5.1 e 5.2 apresentam, como amostra, dois exemplos de exceções (e seus possíveis significados): PO_CHILD_EXCEPTION (para os Filhos do SGPOM_PO) e COORDINATOR_EXCEPTION (para o Coordenador).

Código	Interpretação
invalid_PO_reference	*indica que a ObjRef do PO, passado pelo próprio (PO), está "mal-formada"*
type_invalid	*indica que o tipo de mídia passado como parâmetro não é suportado*
subtype_invalid	*indica que o formato de mídia passado como parâmetro não é suportado*
protocol_not_defined	*indica que o protocolo não é suportado (não existe um SGPOM_DS apropriado)*
invalid_pid	*indica que o identificador (SGPOM_PID) p/ o OM está corrompido*
invalid_Comm_way	*o objeto que encapsula a comunicação do OM está corrompido ou é inválido*
operation_aborted	*a operação de E/S não foi completada (mais de um erro/exceção pode ter ocorrido)*
other_exception	*guarda outra exceção afora as listadas acima. Nesse caso, a operação de E/S foi completada*

***Tabela 5.1**: PO_CHILD_EXCEPTION e suas possíveis interpretações.*

Código	Interpretação
SGPOM_wrong_index	*parâmetro errado de entrada (índice de um SGPOM não existente)*
SGPOM_not_created	*indica que não se pode criar uma instância nova de um SGPOM*
type_invalid	*indica que o tipo de mídia passado como parâmetro está corrompido*
subtype_invalid	*indica que o formato de mídia passado como parâmetro está corrompido*
protocol_not_defined	*indica que o protocolo está corrompido (é inválido)*
operation_partial_completd	*usada para indicar que durante invocações em grupo (de checkpoint e de monitoração) um ou mais componentes não conseguiram completá-la*
operation_aborted	*a operação não foi completada (mais de um erro/exceção pode ter ocorrido)*
other_exception	*guarda outra exceção afora as listadas acima*

***Tabela 5.2**: COORDINATOR_EXCEPTION e suas possíveis interpretações.*

Subsistema de comunicação

Ferraz [Ferr96] ressalta a necessidade de se separar os substratos de transmissão e de controle dos fluxos de mídia contínua. Nesse sentido, ele propõe, como extensão a seu modelo, dois "ingredientes": um mecanismo de controle de taxas para sincronização entre mídias e um subsistema apropriado de comunicação. No caso do SGPOM, essa abordagem motivou a especificação do objeto *Comm_way*. Este atua como um invólucro que encapsula e abstrai a infra-estrutura — exterior ao ORB — a ser utilizada para o transporte do OM desde o cliente (aplicação multimídia ou SGPOM_PO) até o SGPOM_DS (responsável por repassá-lo ao repositório), ou vice-versa. Este foi o meio encontrado para abranger os diversos protocolos ou serviços de comunicação existentes e encarar as limitações das implementações de ORB convencionais.

Em tempo, uma observação: cada um dos subcomponentes SGPOM_DSs deve manter um *array* de *buffers* interno a ser compartilhado pelos objetos Filhos quando do cumprimento das tarefas de armazenagem e recuperação. Cada *buffer* é alocado para manter, temporariamente, o fluxo de dados compondo o estado de um OM oriundo de um cliente ou do repositório.

5.3.4 Decisões de Implementação

Sob a influência do ponto de vista tecnológico do RM-ODP (vide Seção 2.1), as resoluções mais proeminentes tomadas quando da implementação da estrutura do SGPOM compõem o rol a seguir:

1. Emprego de **Java** como linguagem e plataforma de implementação.
2. Construção de servidores ***multithreaded***.
3. Uso de ***sockets*** como mecanismo padrão de comunicação (transmissão) de OMs.
4. Uso do ***Object Serialization*** (Java) como o protocolo usual de *serialização* e de armazenagem dos estados dos OMs e dos subcomponentes Centrais das camadas.
5. Emprego de estruturas de dados cujos tamanhos podem ser incrementados "indefinidamente".
6. Cada uma das entidades do serviço (subcomponentes e Coordenador) possui uma interface IDL definida num módulo apropriado.

Endossando o que foi discutido nas Seções 3.6 e 4.2 e na Subseção 2.2.6, Java, por decisões de projeto, apresenta um suporte robusto, de alto nível, às tarefas de distribuição, persistência e comunicação de dados. Parte do seu conjunto de classes (em uma coleção de pacotes) provê um alicerce útil para a abstração de *sockets*, *threads* e estruturas de dados de uso comum, como tabelas de *hashing* (*hashtables*) e vetores (*vectors*). Por conseguinte, o ambiente SGPOM (composto tanto pelas aplicações-cliente como pelo próprio serviço) foi construído inteiramente em Java.

Ainda que tenham sido especificados os Serviços de Relações e de *Externalização* [OMG94b] [OMG94c] como instrumentos padronizados para o tratamento de composições e de *streams* de objetos CORBA, a sua coexistência, ajustada ao POS, foi desaconselhada por Kleindienst et al. [Klei96b]. O "desacoplamento" e as interdependências cíclicas entre as especificações são, decerto, as maiores pendências a serem ainda resolvidas. Como os OMs (consoante a visão do SGPOM) não precisam definir interfaces IDL, optou-se pelo emprego do protocolo de serialização [Sun97a] proposto para Java em sua versão 1.1.

Cada camada do SGPOM é composta por quatro tipos de subcomponentes, os quais, em um dado momento, poderão estar atendendo a requisições (de E/S, de criação, de monitoração...) provenientes de clientes distintos. Como todos esses subcomponentes estão sob um mesmo processo (modo de ativação compartilhado — ver Subseção 2.2.3), é imprescindível a construção de servidores CORBA com múltiplas *threads* de atendimento às invocações. Sob essa prerrogativa, lançou-se mão de um recurso adicional, baseado no uso de *filtros*, o qual é oferecido pelo produto OrbixWeb (implementação CORBA). Um filtro é uma entidade que realiza alguma operação (ou transformação) fixa sobre as requisições que chegam ou saem de um dado processo. Com ele, tornou-se factível o devido encaminhamento (em paralelo) das requisições a partir da identificação dos seus objetos-alvo. Outros dois tipos de recursos utilizados foram o *locator* — solução proprietária paliativa e alternativa ao Serviço de Nomes do OMG — e o *loader* — espécie de filtro especial que provê transparências de acesso e falha ao cliente do objeto CORBA, na medida em que este é resgatado de uma fonte persistente [IONA97a].

A Figura 5.11 traz a hierarquia dos módulos IDL que compõem e descrevem o SGPOM. Da mesma forma que para o POS, optou-se por manter cada subcomponente da estrutura associado a uma interface, mesmo que isso incorra em uma especificação longa — trazida no Apêndice A deste documento — e em algumas redundâncias evitáveis (por exemplo, as definições dos códigos das exceções e dos *typedefs*).

5.4 Considerações Finais

O quadro a seguir (Tabela 5.3) apresenta um palco comparativo entre o SGPOM e o POS, elucidando as funcionalidades suportadas pelas duas propostas. A ilação decorrente dessa comparação leva à constatação de que o SGPOM exibe uma abordagem modificada para persistência de dados, a qual lida, prioritariamente, com as restrições trazidas com a classe de objetos multimídia. Em suma, o projeto do SGPOM teve como objetivo básico dirimir as deficiências apresentadas pelo POS (Subseção 4.3.6), introduzindo um arranjo mais adequado às necessidades de gerência e acesso a objetos multimídia, dentro do ambiente Multiware.

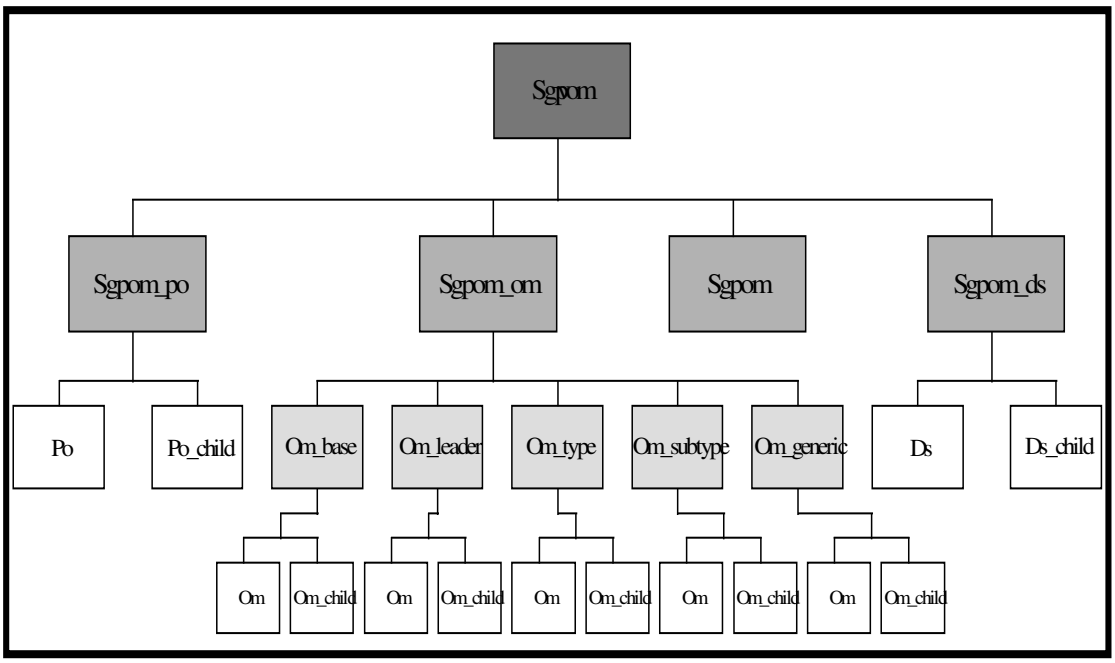

***Figura 5.11**: Arranjo dos módulos IDL que compõem o SGPOM.*

No capítulo que se segue (Capítulo 6), é apresentado o ambiente de execução de um arquétipo do serviço anunciado nesta parte, trazendo, para efeito de análise, um conjunto de medidas de desempenho relativas a configurações distintas de seus componentes. O Apêndice B, por sua vez, expõe as séries de possíveis interações entre os elementos do serviço, segundo a notação MSC (*Message Sequence Chart*). Essas interações compreendem as tarefas de orquestração (instanciação, monitoração e *checkpoint*) dos subcomponentes, bem como as operações de E/S entre as camadas. Além disso, pode-se depreender, mesmo que de maneira parcial, os fluxos de informação entre os componentes do serviço, o que auxilia no processo de especificação do sistema segundo o ponto de vista de informação do RM-ODP.

Funcionalidade	POS	SGPOM
Objeto atrelado à sua persistência	*SIM*	*NÃO*
Objeto CORBA	*SIM*	*NÃO (tanto faz)*
Operações de E/S	*connect(), disconnect(), store(), restore() e delete()*	*Register(), Comm_way_Store(), Store(), Restore() e Delete()*
Operações delegadas	*NÃO*	*SIM*
Componentes (camadas)	*PO, PID, POM, PDSs, Protocol, Datastore*	*SGPOM_PO, SGPOM_PID, SGPOM_OMs (LEADER, TYPE, SUBTYPE, GENERIC), SGPOM_DSs, Protocol, Comm_way e Datastore*
Suporte à Multimídia	*NÃO (pois é geral)*	*SIM*
Suporte à Concorrência ao estado do Objeto	*NÃO (delega para o Serviço de Concorrência)*	*SIM*
Tratamento de exceções	*NÃO*	*SIM*
Monitoração do desempenho	*NÃO*	*SIM*
Protocolo *default* de serialização	*Externalization (OMG)*	*Serialization (Java)*
Transmissão do Objeto	*ORB*[6]	*Subsistema de Comunicação*
Número de interfaces da especificação	*25*[7]	*31*

***Tabela 5.3**: POS X SGPOM.*

[6] Para o Serviço de Externalização.

[7] A especificação do POS traz, ao longo do seu bojo, um conjunto de interfaces opcionais (não-padronizadas), num total de 18, as quais estão relacionadas a exemplos de fábricas, protocolos e repositórios (*Datastores*) passíveis de serem usados na implementação do serviço.

Capítulo 6

Avaliação do Serviço

Tendo em mente a validação da proposta de serviço, um protótipo do SGPOM foi construído, seguindo as ponderações de projeto e de implementação anunciadas no Capítulo 5 desta dissertação. Nesta parte, o interesse se concentra em analisar esse protótipo sob dois pontos de vista: o primeiro visa a breve apresentação do seu ambiente de execução; o segundo, a discussão de suas características de desempenho.

O ambiente de execução do serviço traz um retrato momentâneo das atividades em curso, exibindo as interfaces gráficas de supervisão dos componentes do *framework* e de apresentação das mídias. Por outro lado, a avaliação qualitativa das medidas de desempenho serve como meio de consolidação da filosofia que está por trás do SGPOM, apontando a sua adequação às necessidades levantadas no primeiro capítulo.

6.1 Ambiente de Execução

A construção do arquétipo do SGPOM assegurou a sua *factibilidade*, isto é, tornou possível a comprovação das funcionalidades por ele oferecidas. A observação de seu espaço de execução traz consigo uma "radiografia" dos procedimentos de uso do serviço e esboça como este pode ser integrado àqueles já existentes dentro dos limites de um ambiente distribuído — em particular, da Plataforma Multiware.

A Figura 6.1 ilustra o espaço de execução (*workspace*) do SGPOM. Constata-se que o mesmo é composto por um conjunto de entidades gráficas distintas, compreendendo as interfaces das aplicações-cliente (de gerência e de apresentação) e a hierarquia dos componentes — estes associados às janelas (*consoles*) de execução. Cada uma das interfaces gráficas é descrita, sucintamente, em uma subseção própria.

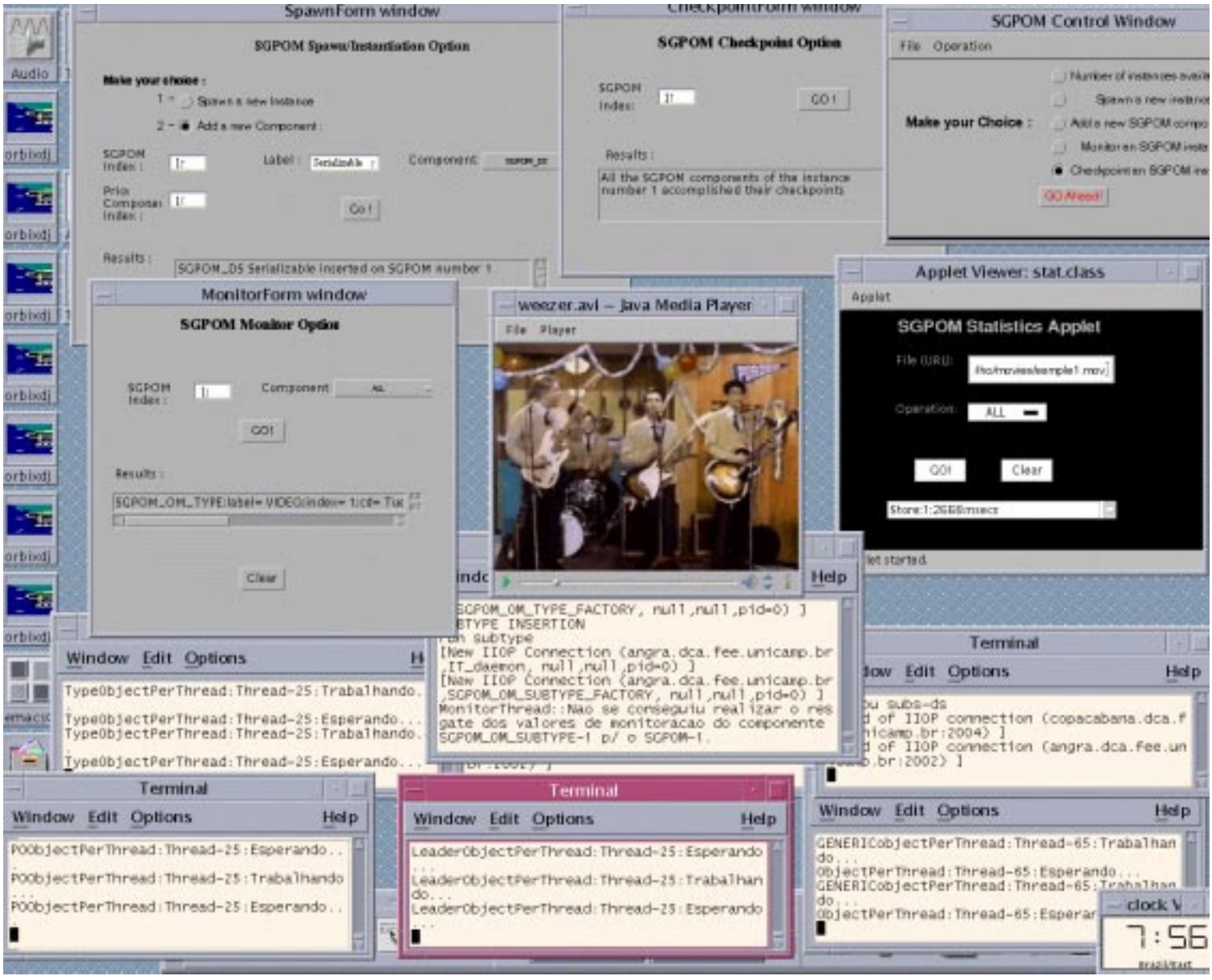

***Figura 6.1**: Ambiente de execução do SGPOM.*

6.1.1 Menu de Operações de Gerência

Mediante esta primeira interface (chamada de *SGPOM Control Window*), o administrador das instâncias do SGPOM pode encetar as tarefas de gerência das possíveis hierarquias de (sub)componentes. Quando entra em atividade (a partir de uma *applet* inicial), esse componente gráfico estabelece uma conexão via ORB com o elemento Coordenador, o qual se encontra em um servidor ativado persistentemente (Subseção 2.2.3), passando a atuar como cliente deste. Por intermédio desse menu de opções (vide Figura 6.2), pode-se conhecer o número de instâncias (cópias) de SGPOM em execução — o que se traduz em uma consulta remota ao atributo *SGPOM_number_instances* da interface do Coordenador — ou se ter acesso às outras janelas de gerência, responsáveis pelas atividades de instanciação, monitoração e *checkpoint*.

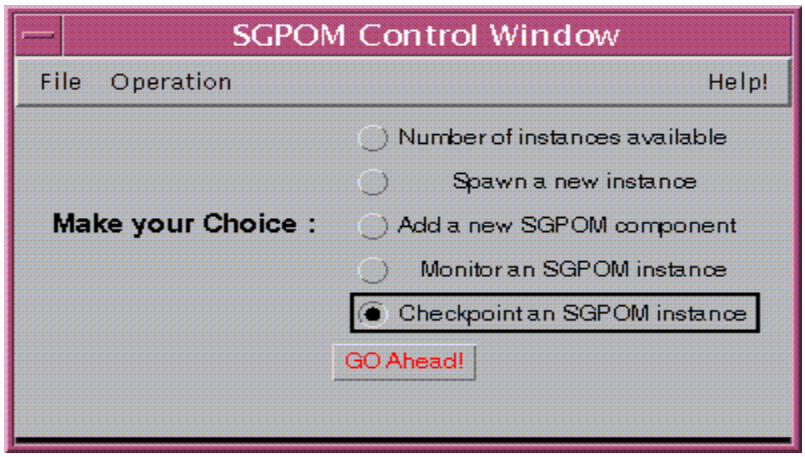

***Figura 6.2**: Interface principal de gerência do SGPOM.*

6.1.2 Janela de Instanciação

A Figura 6.3 traz a interface de criação (*spawning*) de novas instâncias de SGPOM — sob a configuração básica (SGPOM_PO, LEADER e GENERIC) — e de adição dinâmica de novos componentes (gerentes de tipo e subtipo e os *drivers* para os repositórios) às suas hierarquias. Essa janela é ativada ao se premir o botão correspondente no menu principal e se apresenta, em parte, como um formulário a ser preenchido com os dados relacionados aos novos componentes. Por seu intermédio, o administrador pode instanciar diversas cópias do serviço e incrementá-las (estendê-las) de acordo com a necessidade de se introduzir novos repositórios dedicados a tipos e formatos de mídia particulares. Os resultados (ou exceções) gerados pelas operações remotas (IDL) ao Coordenador são também exibidos.

O campo *SGPOM index* se refere ao índice da instância do serviço a que pertence o novo componente, ao passo que *Prior Component Index* deve apenas ser preenchido quando da adição de novos SUBTYPEs e SGPOM_DSs e se refere ao índice do componente localizado, hierarquicamente, na camada acima à sua (para o qual deve manter uma ligação lógica "tipo-subtipo" ou "subtipo-protocolo"). *Label*, por sua vez, é o rótulo indicando o tipo, formato ou protocolo associado. Valores inválidos para esses campos levam a notificações de alerta para o seu preenchimento.

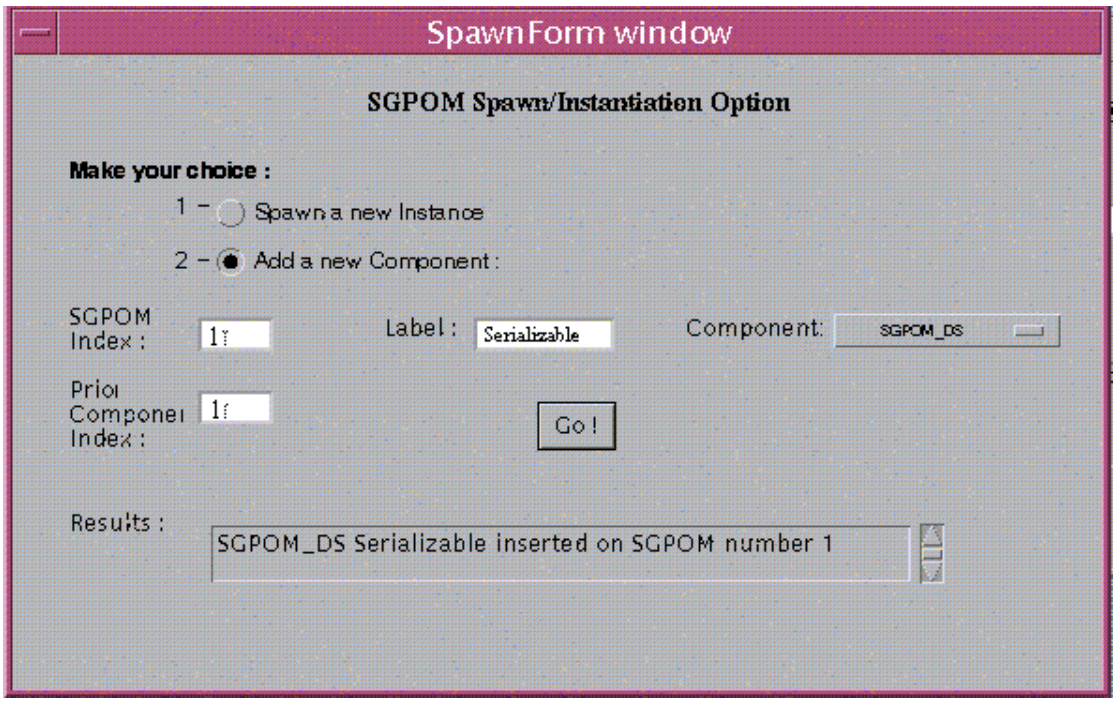

***Figura 6.3**: Janela de instanciação de componentes ou de cópias do SGPOM.*

6.1.3 Janela de Monitoração

Para fins de inspeção (coleta das medidas de desempenho) dos subcomponentes Centrais das camadas, recorre-se a uma interface específica, mostrada na Figura 6.4. Com ela, pode-se optar pela monitoração completa de uma cópia do SGPOM (isto é, de todos os componentes de suas camadas) ou pela monitoração individual. Os resultados são apresentados numa lista de *scrolling*, onde, para cada tipo de componente (do SGPOM_PO aos vários SGPOM_DSs), é reservada uma linha.

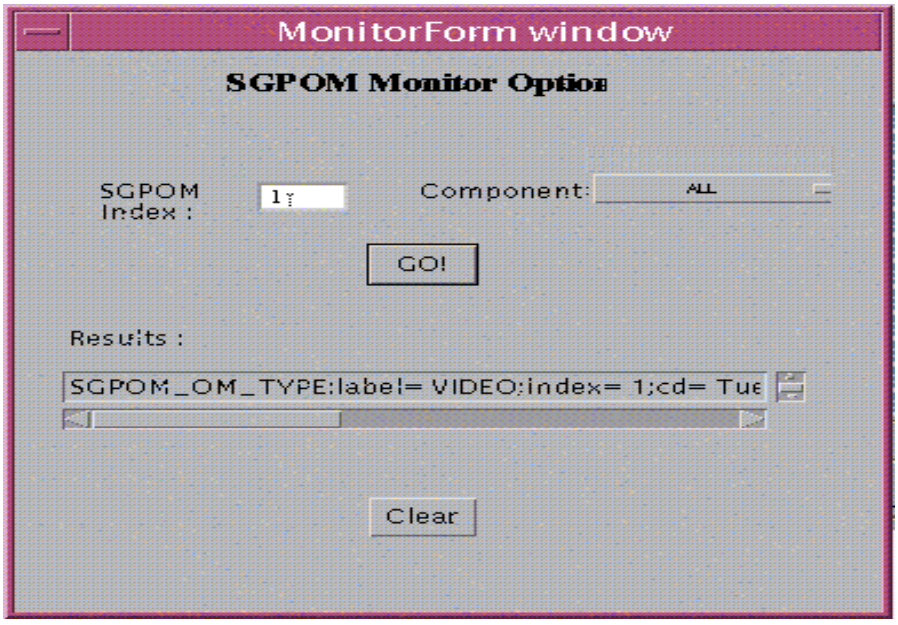

***Figura 6.4**: Janela de monitoração de componentes ou de cópias do SGPOM.*

6.1.4 Janela de *Checkpoint*

A última das interfaces de supervisão é aquela destinada ao *checkpoint* assíncrono (“forçado”) dos componentes de uma instância. Com ela, o administrador pode requisitar, a qualquer momento, a ativação, via o Coordenador, das atividades de armazenagem dos estados dos objetos Centrais de cada camada da hierarquia junto ao GENERIC (repositório padrão). O resultado da invocação à operação *Checkpoint()* do Coordenador reflete o sucesso ou não dessa intervenção de controle (Figura 6.5)

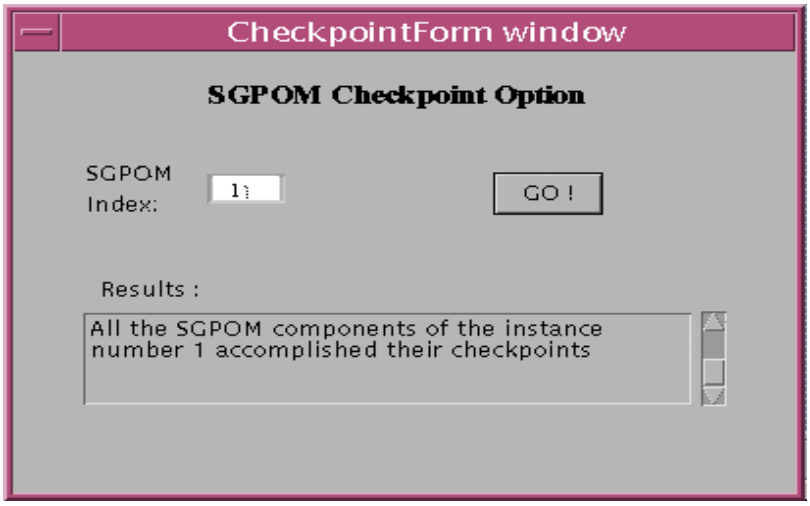

***Figura 6.5**: Janela de checkpoint dos componentes de uma instância do SGPOM.*

6.1.5 Interface de Apresentação Multimídia

A Figura 6.6 traz a interface de exibição de um trecho de vídeo codificado sob o formato AVI. Esse trecho foi encapsulado numa classe VIDEO-AVI, implementada

***Figura 6.6**: Exibição de um OM codificado sob o formato AVI — Interface de um cliente do SGPOM.*

em Java, e representa um dos possíveis OMs manipulados pelo SGPOM. A interface de apresentação foi construída utilizando-se parte do conjunto de classes e interfaces oferecidas com o JMF (retratado na Seção 3.6). Após o resgate do OM, a aplicação multimídia (cliente dos serviços do SGPOM) lança mão dessa janela para fins de exibição e sincronização das mídias de seus objetos. Operações VCR (*play, pause, rewind*, *fast-forward* e *stop*) podem ser realizadas à proporção do consumo da mídia.

6.1.6 Interface de Estatísticas

Esta foi a interface utilizada para a bateria de testes de desempenho sobre o protótipo do serviço, simulando o papel de uma possível aplicação-cliente. A partir de uma URL para um arquivo de mídia, um OM é criado, encapsulando o fluxo de dados multimídia como um atributo (*array* de *bytes*) próprio. Esse objeto será armazenado e recuperado via a API oferecida pela classe SGPOM_CLIENT (vide Subseção 5.2.3).

A partir dessa interface, pode-se optar, em específico, por uma das operações de E/S (de acesso ou gerência). De outro modo, pode-se optar por uma série contínua dessas operações, seguindo o ciclo de vida natural de um OM: registro, armazenagem, recuperação e remoção. A Figura 6.7 ilustra a faceta da *applet* de estatísticas.

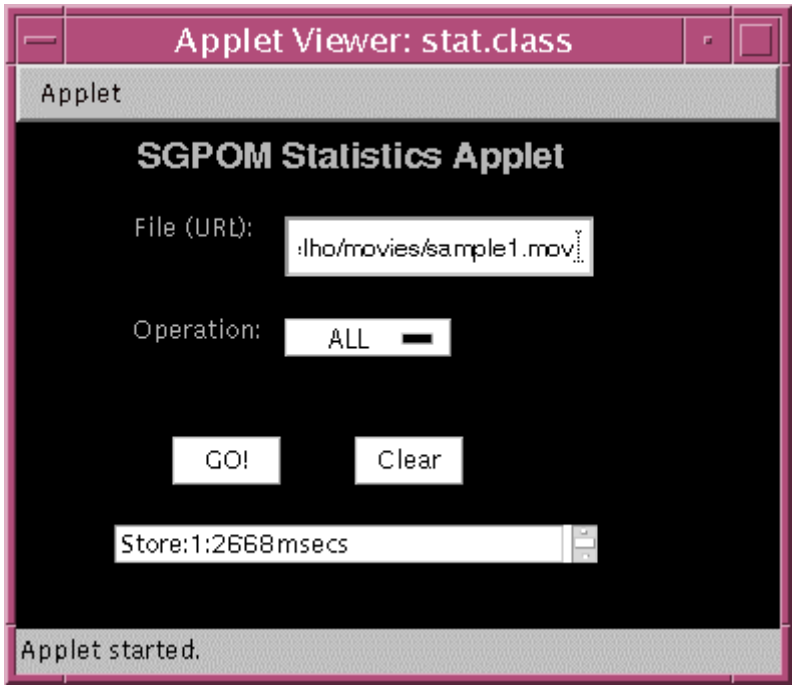

***Figura 6.7**: Interface usada para o levantamento estatístico de desempenho do SGPOM.*

6.2 Análise de desempenho (*Benchmarking*)

Nesta seção, serão discutidos, qualitativamente, os resultados obtidos quando do ciclo de testes de desempenho realizado sobre o protótipo do SGPOM. O propósito é o de se comparar o comportamento ativo (temporal) encontrado para o serviço com aquele conceitualmente esperado, localizando, inclusive, possíveis gargalos.

Para a realização da seqüência de *benchmarking*, dois cenários de aplicações (vide Figura 6.8) passaram a ser avaliados, um de *playback* e outro de *videolog*. O segundo é composto por duas fases: a de captura, em tempo-real, dos fluxos de áudio e vídeo e a de sua exibição. A fase de exibição, por si só, constitui o cenário de *playback*. Como a versão corrente do JMF (1.0.5) não provê suporte às tarefas de captura e geração de mídias, preteriu-se o segundo cenário em favor do primeiro. A *applet* exibida na Figura 6.7 foi construída segundo esse preceito: ao mesmo tempo que serve para a coleta das medidas de desempenho (tempos de invocação das operações de E/S[1]), funciona como aplicação de *playback* — na medida em que repassa o OM para a interface de apresentação (Figura 6.6) instanciada por ela própria.

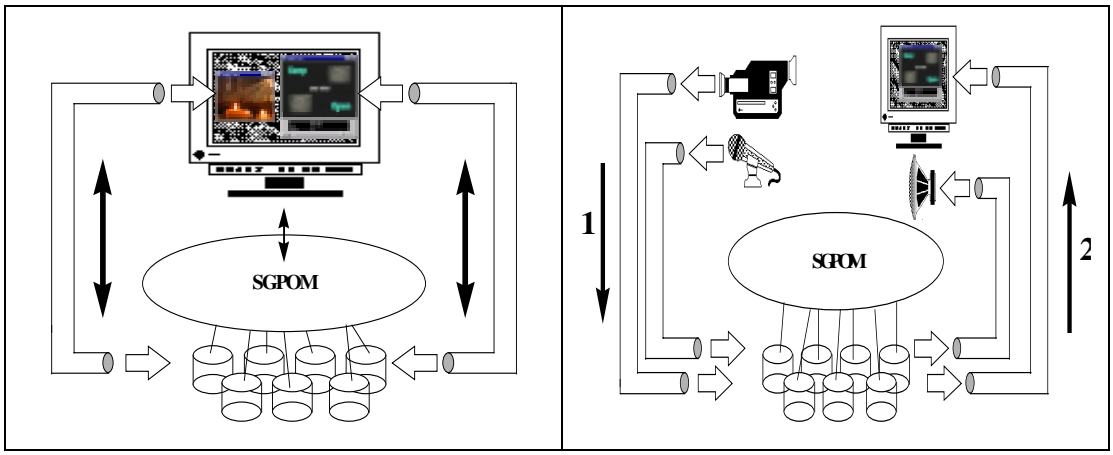

***Figura 6.8**: Cenários de aplicação do SGPOM: playback e videolog.*

6.2.1 Bancada de Testes

A Figura 6.9 apresenta o ambiente (conjunto de máquinas e a topologia da rede de comunicação) onde foram realizadas as medidas de desempenho. Observam-se dois padrões de enlace lógico, ambos baseados no conceito de barramento compartilhado (LAN): um com taxa de transmissão de dados a 10 Mbps (*Ethernet*) e outro operando a 100 Mbps (*Fast-Ethernet*). Para a interconexão das subredes, dois tipos de dispositivos se fazem presentes: um par de concentradores (*hubs*) e um comutador (*switch*). O primeiro é usado para a ligação física entre as estações de trabalho (*End Systems*), mantendo cada uma num segmento lógico próprio; o segundo garante uma taxa constante de entrega de pacotes de dados entre as suas portas, evitando a subutilização ou sobrecarga dos recursos (meio) de comunicação.

Para manter uma análise consistente e mais próxima da realidade, optou-se por trabalhar em um ambiente (*testbed*) heterogêneo, composto por máquinas com recursos diferenciados. Os dados foram obtidos em mais de uma sessão de trabalho (dias diferentes) e sob condições uniformes (menor número possível de usuários se servindo da rede). A Tabela 6.1 traz a configuração (memória, CPU, interface de rede e SO) dos *hosts* usados para as coletas de desempenho dos componentes do serviço.

[1] No conjunto de testes em discussão, não foram realizadas medidas sobre as operações delegadas.

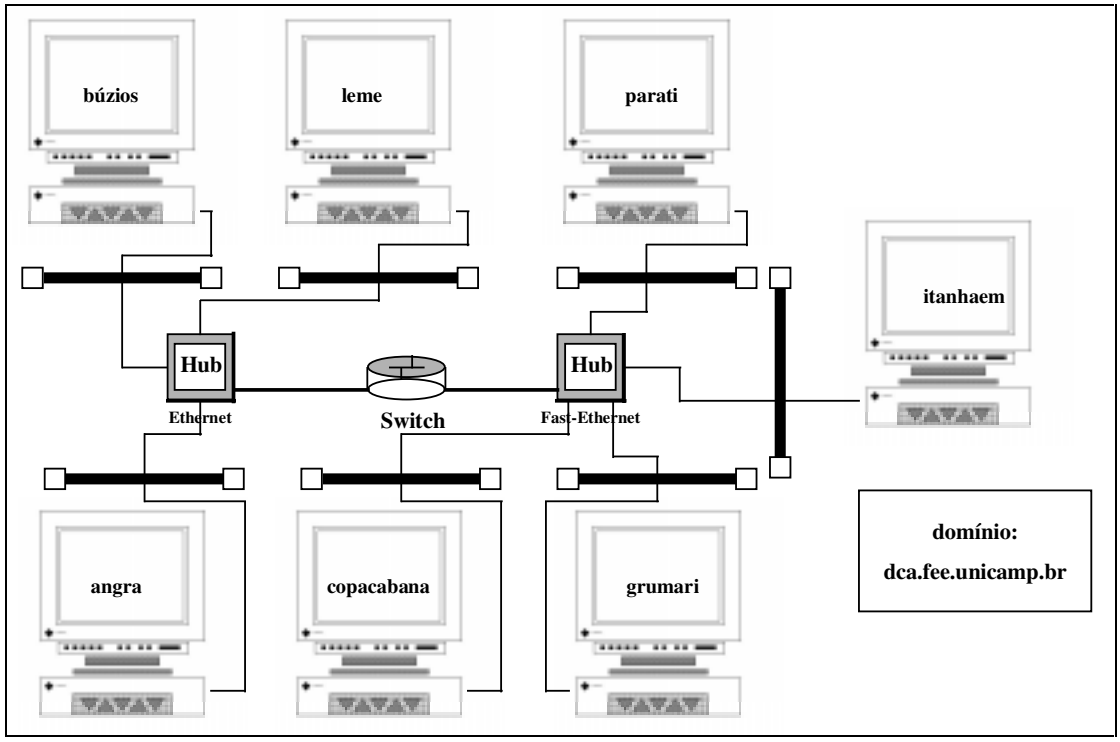

***Figura 6.9**: Ambiente de testes para o SGPOM.*

Host	CPU	Memória	SO[2]	Rede[3]
buzios	UltraSPARC 167 MHz	256 MB	SunOS Release 5.6	10Base-T
grumari	UltraSPARC 167 MHz	256 MB	SunOS Release 5.6	100Base-T
copacabana	UltraSPARC 167 MHz	256 MB	SunOS Release 5.6	100Base-T
leme	UltraSPARC 143 MHz	64 MB	SunOS Release 5.5.1	10Base-T
parati	UltraSPARC 143 MHz	64 MB	SunOS Release 5.5.1	100Base-T
itanhaem	UltraSPARC 167 MHz	64 MB	SunOS Release 5.5.1	100Base-T
angra	SPARCstation-5 110 MHz	64 MB	SunOS Release 5.6	10Base-T

***Tabela 6.1**: Configuração das máquinas que compõem a bancada de testes para o SGPOM.*

6.2.2 Métricas

Para a avaliação do serviço, foi definido o seguinte conjunto de *medidas*, tendo em vista os aspectos temporais de atendimento às requisições de E/S:

- **Tempo de Invocação**: para uma simples requisição, isto significa o intervalo de tempo compreendido entre duas ações. A primeira é a de tratamento do evento gerado ao se comprimir o botão "GO!" da interface de estatísticas (Figura 6.7). A segunda é a dessa mesma interface receber, a partir de SGPOM_CLIENT, a resposta da requisição — para as tarefas de gerência e para a de armazenagem, essa

[2] Todos os *hosts* estão configurados com S.O. UNIX System V™ Release 4.0.
[3] A interface de rede 10Base-T (*Ethernet*) é comumente chamada de par trançado (*twisted pair*), enquanto a 100Base-T é a sua versão para *Fast-Ethernet*.

resposta se traduz em uma indicação de êxito ou insucesso, enquanto para a de recuperação, no recebimento efetivo do OM. No caso de se realizar um *ciclo completo de requisições* (registro–armazenagem–recuperação–remoção do OM), o tempo inicial, à exceção da operação de registro, será dado a partir do tempo final da operação anterior. Para efeito de comparação, estar-se-á interessado na média aritmética desses intervalos (*Tempo médio de Invocação*[4]).

- **Tempo de gerência**: refere-se ao Tempo de Invocação para uma das operações de gerência (registro e remoção). Com este parâmetro, pode-se estimar a parcela de tempo gasta entre as camadas do SGPOM (ou seja, o custo temporal devido exclusivamente ao serviço), haja vista não haver manipulação direta do OM.
- **Tempo de acesso**: refere-se ao Tempo de Invocação para um dos dois tipos de operações de acesso ao OM (armazenagem e recuperação). Pode-se, com este parâmetro, verificar o tempo gasto na comunicação do objeto e na sua manipulação junto ao repositório. Para a armazenagem, este é o tempo total envolvido no atendimento às operações *Comm_way_Store()* e *Store()*.
- **Eficiência**: é a diferença (percentual) entre a unidade e a razão dada entre os tempos médio de gerência (registro + remoção) e de acesso (armazenagem + recuperação) a um mesmo OM.

Os seguintes *parâmetros* (variáveis e constante) tiveram influência direta sobre o conjunto de dados obtido na seqüência de testes:

- **Volume (tamanho) do OM**: como o OM é, simplesmente, um fluxo de mídia encapsulado em memória, este parâmetro (variável) está diretamente associado ao número de *bytes* de um arquivo de mídia. Foram escolhidos cinco tipos de arquivos (sob os formatos MPEG, AVI e Quicktime) com tamanhos diferenciados.
- **Número de objetos Filhos**: como se viu no Capítulo 5, cada um dos SGPOM_XXs tem associado a si um ou mais objetos Filhos, responsáveis pelo real atendimento às requisições de E/S. No caso do protótipo construído, esses objetos são criados concomitantemente à instanciação de um novo SGPOM_XX e ficam disponíveis para o atendimento às requisições. Quanto maior este parâmetro, maior será o número possível de requisições de E/S atendidas simultaneamente pelo mesmo componente, ao mesmo tempo que maior será o tamanho da estrutura de dados (*hashtable* ou vetor) do objeto Central necessária para guardar as *Object References* relativas aos Filhos. Mais ainda, maior será o tempo gasto na busca de um Filho (*thread*) livre para o atendimento à nova requisição. Para a realização das medidas aqui apresentadas, optou-se por trabalhar com um número constante desses objetos (20) para todos os componentes do serviço.
- **Número de clientes**: o número de aplicações-cliente realizando requisições simultâneas (isso se traduz no número de cópias da *applet* de estatística).

Relativa ao primeiro parâmetro (*Volume do OM*), a seguinte consideração deve ser feita: optou-se por trabalhar com arquivos de mídia na faixa de 1 MB a 25 MB, haja vista que o manuseio de volumes maiores a estes apresentou problemas relativos aos recursos de memória alocados dinamicamente (espaço no *heap*). Java oferece como única solução para liberação de memória o mecanismo de *garbage collection* — ao contrário de C e C++, por exemplo, não existe nenhum método geral ou operador para essa função, como *free()* ou *delete*. Esse mecanismo, por ser executado na JVM em uma *thread* de menor prioridade, não é preditivo, ou seja, não garante quando ou

[4] Ao longo desta seção, $\overline{T_x}$ representará esta medida, sendo T_x a notação para o Tempo de Invocação.

em que ordem os objetos (e suas dependências) serão realmente desalocados da memória. O uso contínuo de OMs com grandes volumes pode vir a acarretar em custos adicionais de desempenho por parte dos componentes do serviço e, por esse motivo, descaracterizaria ou invalidaria a proposta de *benchmarking*.

Salvo para o SGPOM_DS, o qual tem que manipular diretamente o OM (via *buffers* e *sockets*), recuperando-o do, ou repassando-o para, o repositório, o tempo gasto em cada camada pode ser inferido, tipicamente, a partir da seqüência de ações apresentada na Figura 6.10, descrita a seguir:

1. Filho da camada anterior envia uma requisição de *Create()* ao SGPOM_XX.
2. O SGPOM_XX busca na tabela o primeiro Filho livre.
3. O SGPOM_XX retorna a Referência ao Filho escolhido.
4. O Filho da camada anterior envia uma requisição de E/S ao Filho escolhido.
5. O Filho escolhido reinicia o ciclo descrito pelos passos anteriores, ficando bloqueado à espera do resultado.
6. O resultado da requisição, finalmente, é propagado e o Filho é liberado.

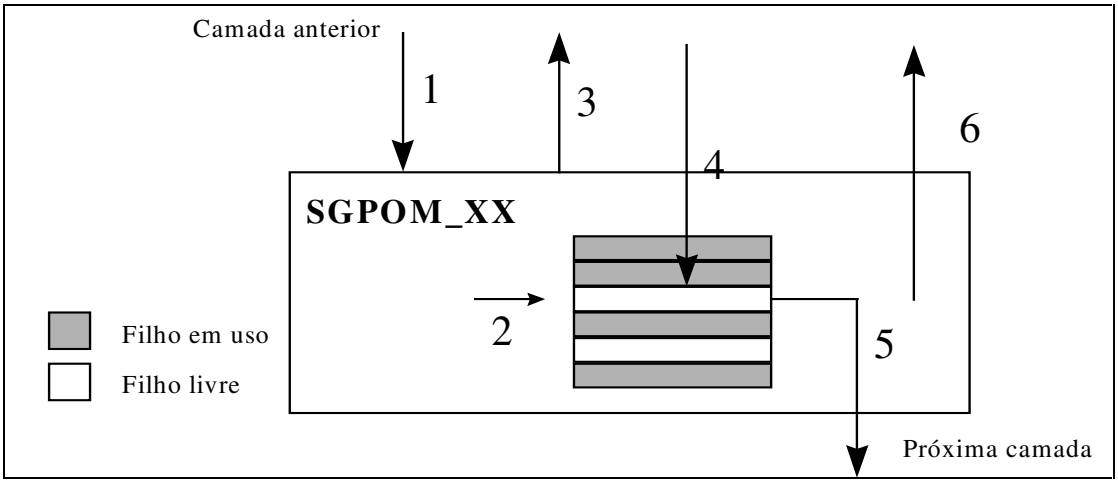

***Figura 6.10**: Seqüência de ações que determina o tempo gasto em cada camada do SGPOM.*

6.2.3 Medidas

Passo a passo, serão apresentados, nesta subseção, os gráficos e tabelas obtidos quando da coleta das medidas de desempenho. Comentários pertinentes descreverão o contexto onde cada *layout* se insere, bem como introduzirão as deduções mais relevantes que se pode, qualitativamente, depreender. Antes, porém, uma ressalva: visando uma análise consistente, manteve-se, em todas as medidas levantadas, a aplicação-cliente executando no mesmo *host* do elemento SGPOM_PO, evitando, com isso, a inclusão de um atraso adicional irrelevante ao estudo.

Configuração básica: Tempo médio de Invocação para as quatro requisições de E/S

A Figura 6.11 (*Layout* #1) traz o conjunto de valores para a medida *Tempo de Invocação*, considerando os quatro tipos de requisições de E/S (*Register*,

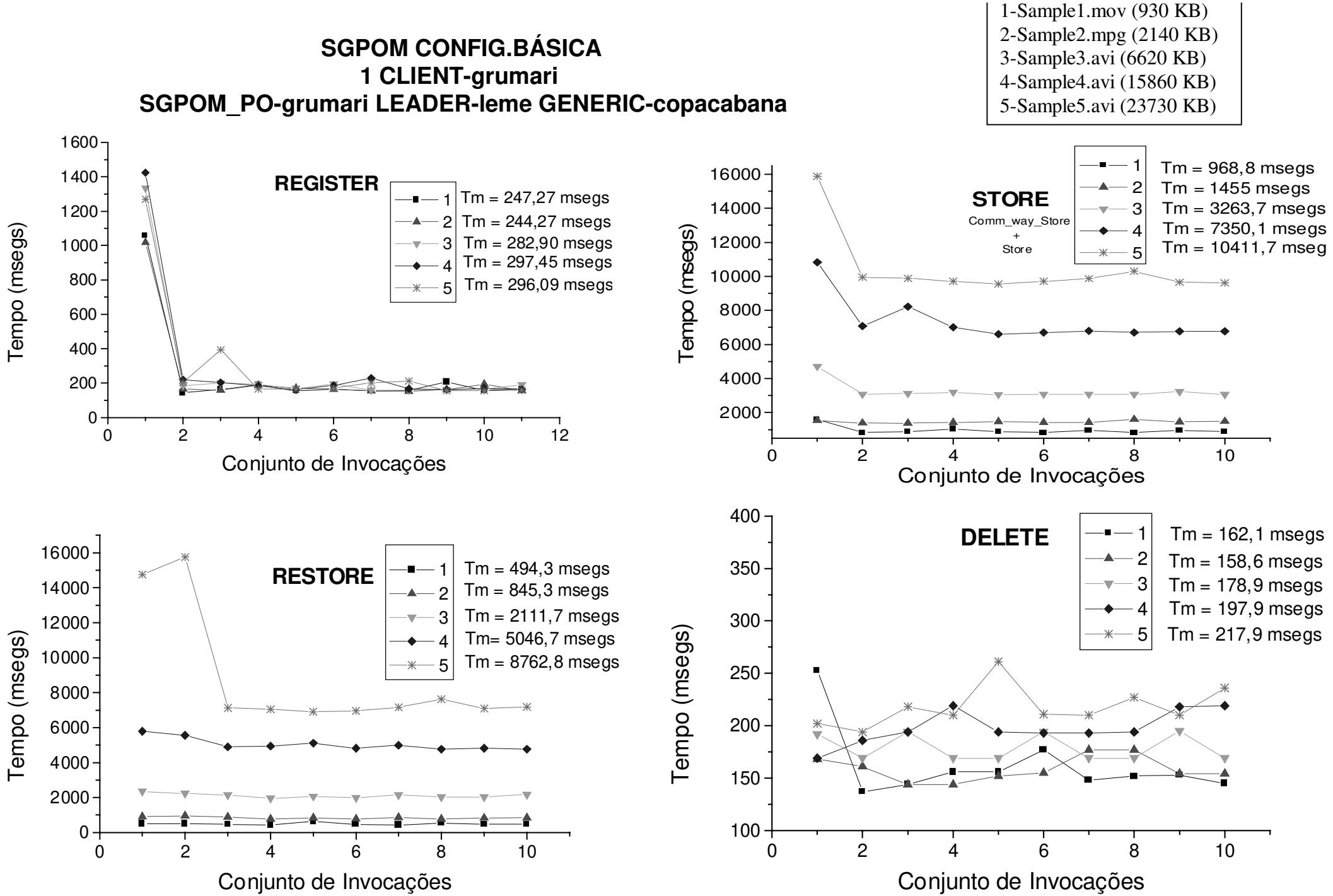

***Figura 6.11**: Tempos de Invocação para os quatro tipos de operações de E/S (configuração básica).*

Comm_way_Store+Store, Restore e *Delete*) e a configuração básica do serviço (SGPOM_PO, LEADER e GENERIC, cada um numa máquina diferente). Isso é feito para os cinco tipos de arquivos (parâmetro *Volume do OM*), seguindo ciclos completos de requisições (Subseção 6.2.2) para um só cliente. Observa-se, à exceção da requisição de registro, um conjunto de dez pontos de medidas, cuja média temporal é dada em milissegundos (msegs), sendo cada ponto obtido em uma série.

A Tabela 6.2 traz, por sua vez, o cálculo da medida *Eficiência* para cada um dos cinco tipos de OMs. Esse cálculo é feito a partir da seguinte fórmula:

$$Eficiência = 1 - Tm_{gerência} / Tm_{acesso} \quad (1),$$

onde $Tm_{gerência}$ é a média dos tempos médios de registro e de remoção, enquanto Tm_{acesso} é a média dos tempos médios de armazenagem e de recuperação do OM.

Arquivo	$Tm_{gerência}$ (msegs)	Tm_{acesso} (msegs)	*Eficiência* (%)
Sample1.mov	204.68	731.55	72.02
Sample2.mpg	201.43	1150.15	82.49
Sample3.avi	230.9	2687.7	91.41
Sample4.avi	247.67	6198.4	96.00
Sample5.avi	256.99	9587.25	97.32
Média	228.33	4071.01	94.39

***Tabela 6.2**: Medida Eficiência para os dados obtidos no Layout #1.*

Da Tabela 6.2 e do *Layout* #1, pode-se depreender que:

- No caso da atividade de registro, o primeiro ponto de cada curva merece uma atenção especial: ele se destaca dos outros pelo fato de acumular a latência embutida no estabelecimento inicial das conexões entre os objetos das camadas do serviço. A implementação de ORB utilizada (OrbixWeb3.0) permite a configuração manual (estática ou dinâmica) de uma série de parâmetros relativos a funcionalidades adicionais oferecidas aos objetos CORBA, dentre os quais se destaca o de *timeout* da conexão entre um cliente e um servidor (*IT_CONNECTION_TIMEOUT* [IONA97b]). Com esse parâmetro, pode-se especificar o tempo em que uma conexão (*socket* de comunicação das requisições) entre dois objetos ficará disponível (*alive*) para o fluxo de próximas invocações entre eles. Desse modo, evita-se que para cada uma das outras requisições de E/S dos ciclos se tenha um *overhead* adicional de criação de uma nova conexão.
- A transmissão do OM (para as operações de armazenagem e recuperação) é feita, via *socket*, entre máquinas semelhantes (e de melhor configuração — *grumari* e *copacabana*), utilizando-se, desse modo, apenas o enlace *Fast-Ethernet*.
- As requisições de gerência, a despeito de pequenas diferenças, apresentam um comportamento temporal uniforme (constante ou "determinístico"), na ordem de 230 msegs (ver média do $Tm_{gerência}$). Por outro lado, as requisições de acesso demonstram, como esperado, um crescimento contínuo dos tempos de invocação, de acordo com o tamanho do OM manipulado.
- Quanto maior o parâmetro *Volume do OM*, maior é a medida *Eficiência*. Ou seja, menor é a parcela proporcional do tempo total devida efetivamente ao SGPOM. Por conseguinte, pode-se depreender que quanto maior a *Eficiência*, maior é a relação custo/benefício de se lançar mão do serviço.

Configuração básica X Configurações estendidas: Tempo médio de Invocação para a requisição de registro do OM

A Figura 6.12 (*Layout* #2), que tem como suporte as Tabelas 6.3 e 6.4, traz a medida *Tempo médio de Invocação* para a operação de registro de um único OM, considerando-se, agora, o serviço sob configurações diferenciadas (estendidas). Da análise dos valores, pode-se constatar que:

- Com o aumento do número de camadas do serviço (ordens 2, 3 e 4 da Tabela 6.3), o tempo de invocação passa a ser, como esperado, proporcionalmente maior (dentro de um intervalo previsível), não produzindo efeitos colaterais no comportamento dos componentes.
- Com o aumento do número de componentes — TYPE, SUBTYPE e SGPOM_DS — nas camadas inferiores (ordens 5, 6 e 7 da Tabela 6.3), houve um incremento na medida de desempenho devido ao tempo introduzido quando da escolha, em cada nível, de para qual dos elementos logicamente seguintes deveria ser repassada a nova invocação de E/S (no caso, de registro). Conclui-se que quanto maior o número de ramos lógicos, maior será o custo temporal incorrido pela decisão[5] acerca do melhor componente sucessor da hierarquia.

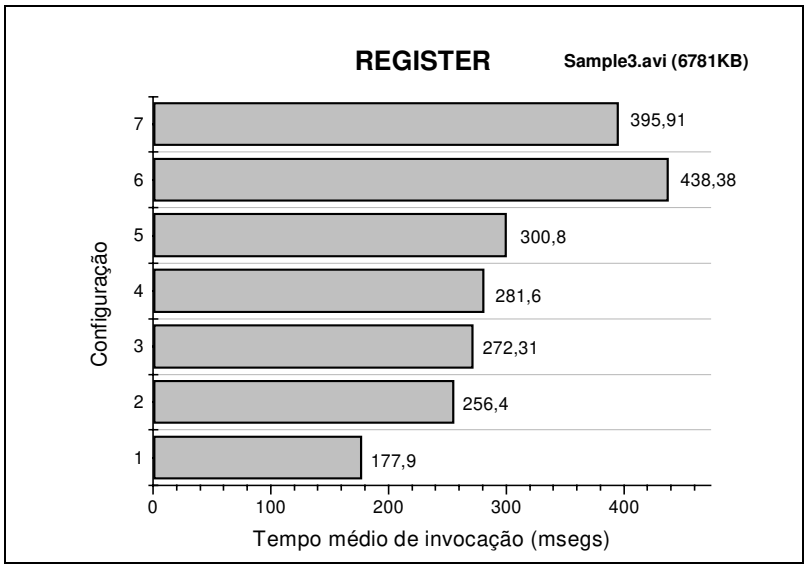

***Figura 6.12**: Tempo médio de Invocação para as requisições de registro de um OM (Sample3.avi), segundo as configurações do serviço dadas na Tabela 6.3.*

Ordem	Configuração
1	1 SGPOM_PO – 1 LEADER – 1 GENERIC (configuração básica)
2	configuração básica + 1 TYPE
3	configuração básica + 1 TYPE + 1 SUBTYPE
4	configuração básica + 1 TYPE + 1 SUBTYPE + 1 SGPOM_DS (config. típica)
5	configuração básica + 2 TYPEs + 1 SUBTYPE + 1 SGPOM_DS
6	configuração básica + 2 TYPEs + 2 SUBTYPEs + 2 SGPOM_DSs, na forma: 2 ramos lógicos na seqüência (1 TYPE – 1 SUBTYPE – 1 SGPOM_DS)
7	configuração básica + 2 TYPEs + 2 SUBTYPEs + 10 SGPOM_DSs, na forma: 2 ramos lógicos na seqüência (1 TYPE – 1 SUBTYPE – 5 SGPOM_DSs)

***Tabela 6.3**: Configurações do SGPOM (arranjos com diferentes componentes) usadas no Layout #2.*

[5] Como exemplo para o LEADER, essa decisão se reflete em duas atividades: busca dos TYPEs associados ao tipo do OM e verificação daquele com nível de sobrecarga mais baixo.

Componente	*Host*
SGPOM_PO	grumari
LEADER	leme
GENERIC	copacabana
TYPEs	parati
SUBTYPEs	angra
SGPOM_DSs	itanhaem

***Tabela 6.4**: Máquinas alocadas aos componentes do SGPOM.*

Configuração básica X Configuração típica: Tempo médio de Invocação para as requisições de acesso ao OM

Denomina-se *configuração típica* do SGPOM aquela em que para cada um dos componentes da estrutura do serviço (SGPOM_XX) se tem uma única instância disponível. Ela estende a básica na medida em que envolve a existência de um ramo lógico TYPE–SUBTYPE–SGPOM_DS (ou seja, a existência de um repositório além do GENERIC). A Figura 6.13 (*Layout* #3) oferece a comparação entre as configurações básica e típica, tendo em vista a medida *Tempo médio de Invocação* realizada para cada um dos dois tipos de requisições de acesso (armazenagem e recuperação) e para cada um dos cinco tipos de arquivos de mídia. Nessa comparação, cada componente se encontra associado a um domínio (máquina) distinto, de acordo com os pares colocados na Tabela 6.4.

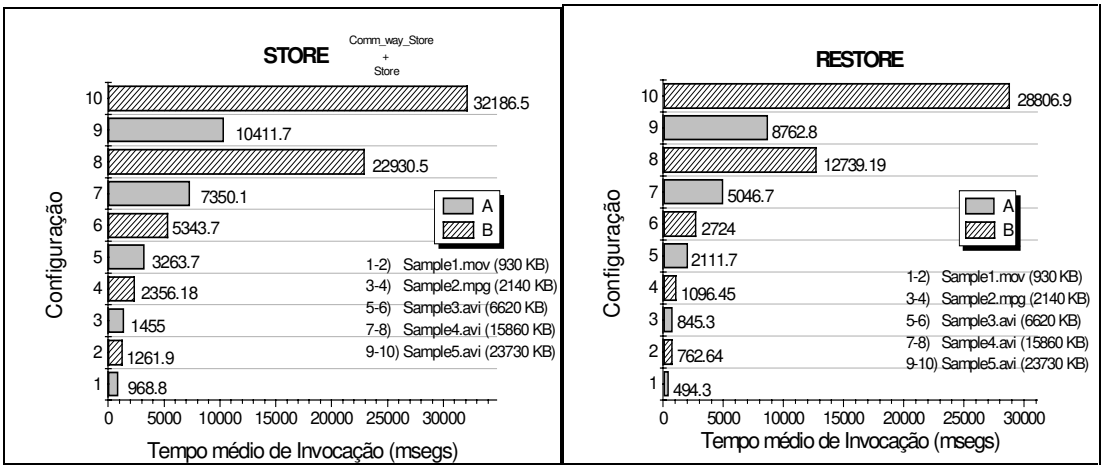

***Figura 6.13**: Comparação da medida Tempo médio de Invocação para as requisições de acesso ao conjunto de OMs, obtida para as configurações básica (A) e típica (B) do SGPOM.*

Verifica-se que o *Tempo médio de Invocação* para a configuração típica é substancialmente maior que o correspondente da configuração básica, quando da manipulação de OMs volumosos (*Sample4.avi* e *Sample5.avi*). Isso se deve a dois fatos:

- o aumento no número de camadas (similar ao discutido no cenário anterior).
- a diferença de configuração de memória das máquinas alocadas ao GENERIC e ao SGPOM_DS (*copacabana* e *itanhaem*, respectivamente). Quanto maior o OM, maior será a reserva dos recursos de rede (para ambos os casos, *Fast-Ethernet*) e de memória (256 MB *versus* 64 MB).

Configuração básica: SGPOM_PO e GENERIC na mesma máquina

Neste cenário, passa-se a investigar o *Tempo médio de Invocação* para as requisições de acesso, sob a configuração básica do serviço, considerando-se a

localização da aplicação-cliente e a do GENERIC — para caracterização da fonte e destino quando da transmissão dos OMs. O *Layout* #4 (Figura 6.14), suportada pela Tabela 6.5, traz a comparação entre os valores encontrados para essa medida, considerando o SGPOM_PO (e, logo, o cliente) e o GENERIC sob a mesma máquina e em máquinas distintas.

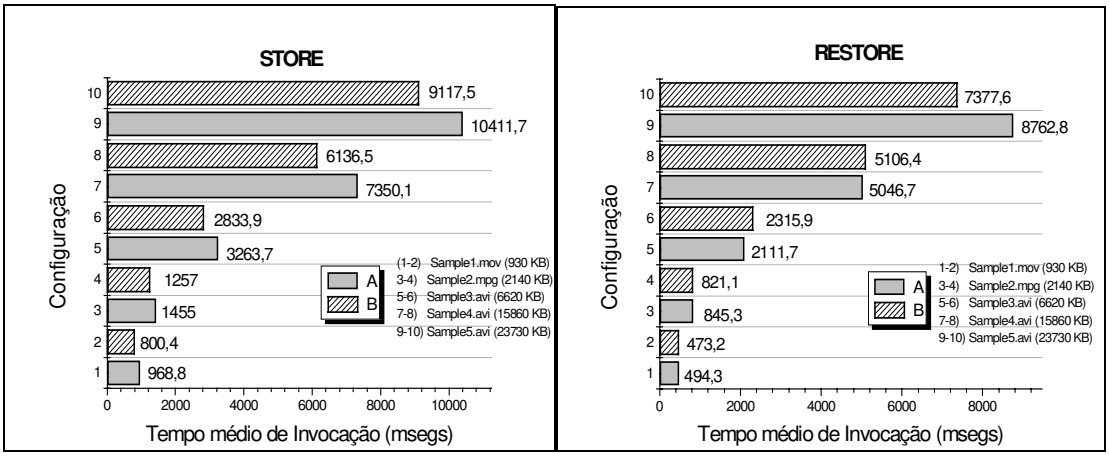

***Figura 6.14**: Comparação da medida Tempo médio de Invocação para as requisições de acesso ao conjunto de OMs, obtida para a configuração básica, com o SGPOM_PO (cliente) e o GENERIC na mesma máquina (A) e em máquinas distintas (B) — ver Tabela 6.5.*

Configuração	Máquinas
A	SGPOM_PO = GENERIC (buzios)
B	SGPOM_PO (grumari) e GENERIC (copacabana)

***Tabela 6.5**: Máquinas alocadas aos componentes SGPOM_PO e GENERIC para o Layout #4.*

A partir da análise dos valores, verifica-se um comportamento temporal similar para os dois tipos de configuração, independentemente do *Volume do OM*. A justificativa mais plausível para isso é a de que a transmissão de dados volumosos em uma rede com vazão (*throughput*) alta — *Fast-Ethernet* — apresenta um desempenho comparável (no caso, até melhor) àquela realizada em uma mesma máquina (*buzios* — operando em *Ethernet*), sob algum possível mecanismo de otimização (memória compartilhada) no estabelecimento do *socket* (substrato de comunicação do OM).

Configuração básica: Detalhamento do Tempo médio de Invocação para os quatro tipos de requisições de E/S

O interesse neste item é o de se revelar as parcelas de tempo devidas às atividades de transmissão (tempo médio gasto no *socket*, $\overline{T}_{socket}$), gerência e manipulação junto ao repositório ($\overline{T}_{Datastore}$) e encapsulamento em memória ($\overline{T}_{memória}$) do conjunto de OMs, deduzindo daí o custo temporal restrito ao SGPOM, em sua configuração básica ($\overline{T}_{SGPOM}$), para as operações de acesso e gerência.

Sob essa expectativa, o *Layout* #5 (Figura 6.15) traz, em detalhe, a proporção (percentual) relativa a cada uma das atividades para a medida *Tempo médio de Invocação*, considerando-se as requisições de gerência dos OMs. O $\overline{T}_{SGPOM}$ pode ser deduzido, nesse caso, a partir da seguinte fórmula:

$$\overline{T}_{SGPOM} = \overline{T}_{TOTAL} - \overline{T}_{Datastore} \quad \textbf{(2)},$$

onde $\overline{T}_{TOTAL}$ representa os valores mostrados, em detalhe, no *Layout* #1, para o *Tempo médio de Invocação*.

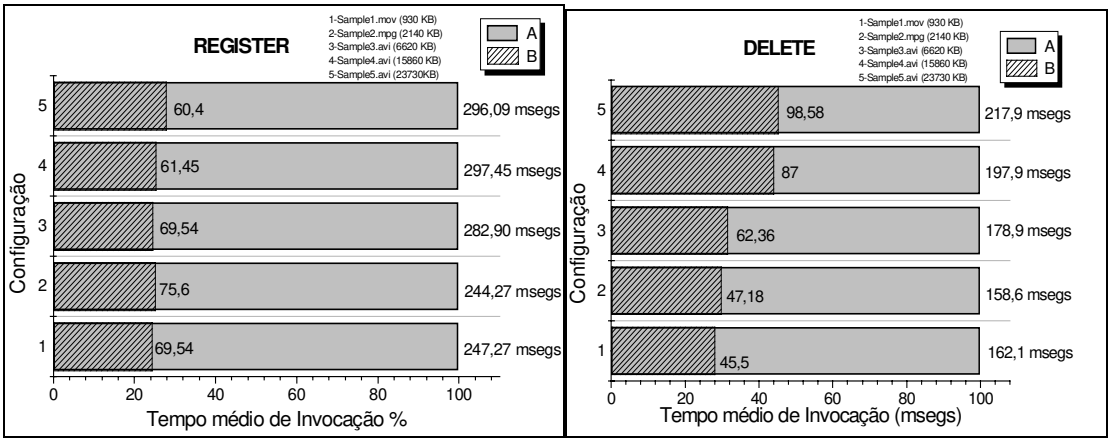

***Figura 6.15**: Detalhamento da medida Tempo médio de Invocação para as requisições de gerência ao conjunto de OMs, obtida para a configuração básica do SGPOM, onde (A) indica o $\overline{T}_{TOTAL}$ e (B), o $\overline{T}_{Datastore}$.*

O *Layout* #6 (Figura 6.16) segue a mesma orientação colocada no parágrafo anterior. Contudo, aqui, o $\overline{T}_{SGPOM}$ pode ser inferido, para as requisições de armazenagem e de recuperação dos OMs, a partir das respectivas expressões:

$$\overline{T}_{SGPOM} = \overline{T}_{TOTAL} - \overline{T}_{Datastore} - \overline{T}_{socket} - \overline{T}_{memória} \quad \textbf{(3)},$$

$$\overline{T}_{SGPOM} = \overline{T}_{TOTAL} - \overline{T}_{datastore} - \overline{T}_{socket} \quad \textbf{(4)}.$$

A parcela $\overline{T}_{memória}$ diz respeito ao tempo gasto na leitura (para a memória) do fluxo da mídia que compõe o OM, a partir da URL do arquivo de mídia. Por esse motivo, só é contabilizada para a tarefa de armazenagem.

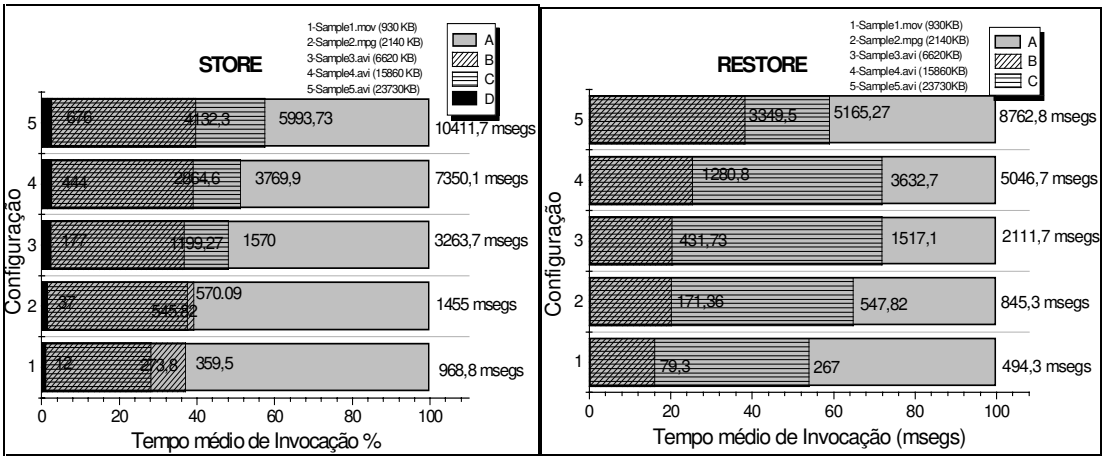

***Figura 6.16**: Detalhamento da medida Tempo médio de Invocação para as requisições de acesso ao conjunto de OMs, obtida para a configuração básica do SGPOM, onde (A) indica o $\overline{T}_{TOTAL}$, (B), o $\overline{T}_{Datastore}$, (C), o $\overline{T}_{socket}$ e (D), o $\overline{T}_{memória}$.*

Por ainda não se dispor de uma implementação completa de um repositório dedicado a mídia (com otimizações e restrições temporais de acesso ao disco asseguradas — ver Capítulo 3), utilizou-se, para efeito de *benchmarking*, o sistema de arquivos da rede (com o *Object Serialization* da Java como protocolo de serialização dos dados) como *Datastore* padrão. Por esse motivo, $\overline{T}_{Datastore}$ é relativo, em grande parte, ao tempo gasto no acesso ao arquivo (que contém o OM) dentro do diretório adequado (geralmente remoto), o que é feito via NFS (*Network File Services*). A partir dessa consideração e da observação da Tabela 6.6 — a qual fornece os valores de $\overline{T}_{SGPOM}$ para os quatro tipos de requisições de E/S, de acordo com as expressões (2), (3) e (4) — e dos *Layouts* #5 e #6, pode-se deduzir que:

- Para a atividade de registro, a contribuição temporal de $\overline{T}_{Datastore}$ se mantém, praticamente, constante. Isso porque ela se restringe à criação de um arquivo de conteúdo nulo e da *string* de identificação (SGPOM_PID) associada ao OM. No caso da tarefa de remoção, à medida que cresce o volume do OM manipulado, aumenta-se a quota devida a $\overline{T}_{Datastore}$, o que está de acordo com o esperado: maior será o tempo gasto na remoção física dos *bytes* do objeto no disco.
- Tanto $\overline{T}_{Datastore}$ como $\overline{T}_{socket}$ são responsáveis diretos pela grande parte do custo temporal para as atividades de armazenagem e recuperação, as quais envolvem a manipulação direta dos OMs. Para esses casos (colunas 2 e 3 da Tabela 6.6), a parcela de tempo gasta, estritamente, com os componentes do SGPOM manteve-se, como esperado, dentro de uma faixa uniforme.
- O $\overline{T}_{memória}$ tem contribuição ínfima para o $\overline{T}_{TOTAL}$ da tarefa de armazenagem, podendo ser desprezado sem prejuízo de consistência da análise.
- A atividade de armazenagem compõe-se de um ciclo de propagações pelas camadas de duas operações: *Comm_way_Store()* e *Store()*. Por esse motivo, o $\overline{T}_{SGPOM}$ apresenta valores superiores (na ordem de duas vezes) àqueles comumente esperados.

  Nos processos de armazenagem e recuperação, grande parte do $\overline{T}_{SGPOM}$ é devida às tarefas de alocação e retenção do OM em *buffers* intermediários nos SGPOM_DSs (no caso, no GENERIC). Para o protótipo construído, nenhuma técnica especial de implementação de *buffers* (vide Subseção 3.5.3) foi empregada, o que garantiria um acesso otimizado ao objeto e, por conseguinte, incorreria em melhor desempenho por parte do serviço. Extrapolando-se para grandes OMs, esse pode vir a se tornar em um ponto de gargalo acentuado e, por isso, merece atenção à parte. Uma outra fonte de medida (dada, por exemplo, por $\overline{T}_{buffer}$) poderia indicar a quota temporal relativa às atividades de *bufferização*, algo interessante para os projetistas dos repositórios multimídia.

	Registro	**Armazenagem**	**Recuperação**	**Remoção**
Sample1.mov	177.73 (71.88%)	323.5 (33.39%)	148 (29.94%)	116.6 (71.93%)
Sample2.mpg	168.67 (69.05%)	302.29 (20.77%)	126.12 (14.92%)	111.42 (70.25%)
Sample3.avi	213.36 (75.42%)	317.43 (9.73%)	162.87 (7.71%)	116.54 (65.14%)
Sample4.avi	236 (79.34%)	271.6 (3.69%)	133.2 (2.64%)	110.9 (56.04%)
Sample5.avi	236.69 (79.60%)	387.1 (3.72%)	248.03 (2.83%)	119.32 (54.75%)

Tabela 6.6: $\overline{T}_{SGPOM}$ *(em msegs) para os quatro tipos de requisições de E/S e cinco tipos de OMs, configuração básica, conforme as expressões (2), (3) e (4).*

Configuração básica: Variação do Número de clientes

Finalmente, considera-se o último dos parâmetros introduzidos na lista apresentada na Subseção 6.2.2: número de aplicações-cliente se servindo, simultaneamente, dos serviços do SGPOM (*Número de clientes*). A interpretação desse parâmetro reflete o comportamento do serviço no cumprimento de um de seus objetivos de projeto: o de prover suporte à concorrência no acesso e gerência dos OMs. Porém, antes da análise dos dados, uma nota passa a ser cabível (mais do que isso, imprescindível!) dentro desse contexto: a ferramenta utilizada na construção do protótipo (OrbixWeb3.0), por se encontrar numa versão de demonstração (de domínio público), apresentou problemas de restrição ao uso de todas as suas possíveis funcionalidades, acarretando, como efeito colateral, a privação de certos mecanismos suplementares àqueles considerados básicos. Nesse produto, parte das funções do ORB e do BOA (ver Subseção 2.2.3) é implementada na forma de um *daemon* (*orbixdj* [IONA97a]), o qual fica responsável pela comunicação e ativação dos objetos CORBA. Esse *daemon*, sob essa versão, impôs limites no número de conexões estabelecidas, simultaneamente, a um mesmo servidor e no número de servidores sob a sua "tutela" em uma mesma máquina, algo que afetou sobremaneira o desempenho dos componentes do SGPOM, quando da realização de suas tarefas de atendimento às requisições concorrentes de E/S.

Mesmo com as limitações impostas pela implementação de ORB utilizada, foram levantados alguns valores para a medida *Tempo médio de Invocação*, considerando, unicamente, o arquivo *Sample1.mov*, para os quatro tipos de requisições de E/S, sob a configuração básica do serviço. Esses valores são exibidos no *Layout* #7 (Figura 6.17), onde *Número de clientes* é o parâmetro variável (um, dois ou cinco — este último, o máximo valor factível) e corresponde ao número de *applets* de estatísticas se servindo, concomitantemente (ciclo completo de invocações iniciados ao mesmo tempo), de uma mesma instância do SGPOM.

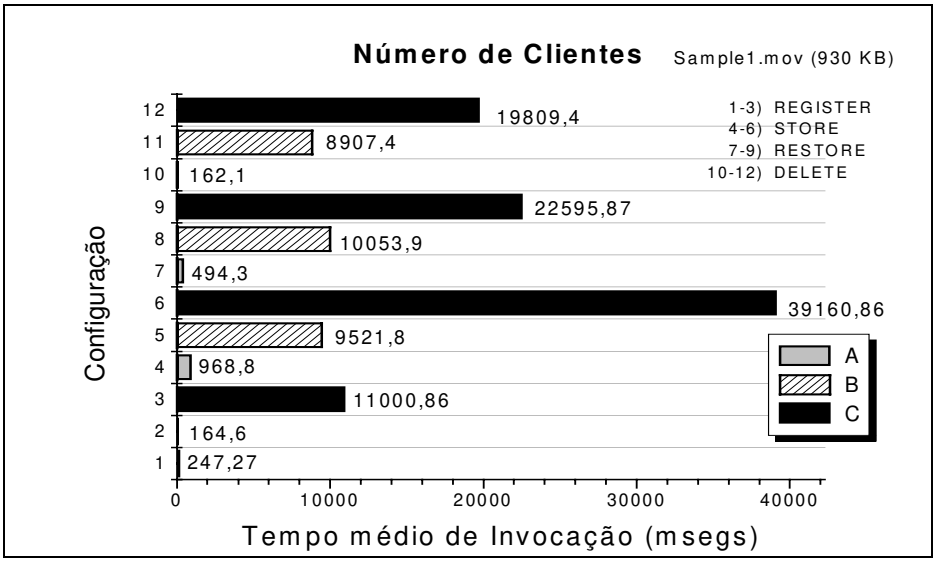

***Figura 6.17**: Tempo médio de Invocação para as requisições de E/S relativas a um OM (Sample1.mov), obtido para a configuração básica, onde o Número de clientes é um (A), dois (B) ou cinco (C).*

A única observação permissível é a de que há uma *degradação* da medida *Tempo médio de Invocação* à proporção do aumento do número de clientes,

independentemente do tipo de requisição realizada (de gerência ou acesso). O que se pode cogitar é que a liberação dos recursos de comunicação do ORB fica comprometida consoante alguma política interna demarcada pelo produto, a qual favorece as requisições que primeiro atingem os seus objetos-alvo (o que se constitui num sério gargalo de desempenho). Esse aspecto obsta uma maior compreensão do comportamento dos componentes do serviço, haja vista se tornar inviável qualquer conclusão determinística (predição) acerca do atendimento real às requisições de E/S.

6.3 Considerações Finais

No decorrer deste capítulo, uma implementação piloto do SGPOM serviu como alvo de avaliação; o intuito foi o de apresentar o conjunto de componentes gráficos que a compõe, bem como comparar o comportamento temporal de sua estrutura sob situações e arranjos distintos.

Todavia, por se tratar de uma primeira versão, algumas modificações/extensões são passíveis de consideração. No caso das interfaces gráficas, um bom exemplo seria o da criação de uma *applet* adicional para a "calibragem", em tempo de execução do sistema, do conjunto de parâmetros de configuração dos componentes, tais como o número de objetos Filhos para cada SGPOM_XX, a freqüência do ciclo temporal de monitoração realizado pelos objetos Controladores e o período de tempo entre duas atividades seqüenciais de *checkpoint* iniciadas pelo Coordenador. Esses valores são, no esquema atual do serviço, definidos ainda em fase de compilação.

Uma segunda janela poderia ser construída para possibilitar a visualização gráfica dos correntes arranjos estruturais (topologia) das instâncias do serviço, auxiliando o administrador do SGPOM nas suas tarefas de supervisão.

Em se tratando das atividades de *benchmarking* de desempenho do protótipo do serviço, é necessário que sejam destacados os seguintes pontos:

1. **Dependência para com a implementação CORBA escolhida**: o estudo da proposta, realizado neste capítulo, se baseou em valores de medidas temporais obtidas para um protótipo construído sobre um único produto CORBA, o OrbixWeb 3.0. Como o OMG ainda não especificou parâmetros padrão (neutros) de avaliação da qualidade das ferramentas existentes, torna-se dificultoso inferir como seria o comportamento da mesma implementação em outros ambientes. Plášil e outros [Pláš98] ressaltam a dificuldade de se avaliar quaisquer medidas de *benchmarking* feitas sobre ações complexas em um ORB convencional (tais como seqüências de requisições), haja vista que é custosa a identificação da influência implícita de certos fatores. O caminho proposto pelos autores é o de se definir, explicitamente, ações elementares (operações de *marshalling* e de *dispatching*, por exemplo) relevantes ao cenário em questão (avaliação de *throughput*, escalabilidade e latência do ORB ou de seu grau de robustez). Para o SGPOM, uma biblioteca de *benchmarking* desses parâmetros básicos viria a ser interessante, auxiliando na estimativa da parcela temporal gasta, unicamente, com a transmissão das requisições de E/S pelas camadas do *framework*.
2. **Carência de resultados para fins de comparação**: devido às particularidades do serviço proposto, não se dispõe de outros conjuntos de medidas de desempenho para fins de cotejo. Em outros ramos de atuação da Multimídia, como o do levantamento da percepção sensorial humana à defasagem de sincronização entre

fluxos de mídia — fruto da pesquisa de Steinmetz e outros [Blak96] —, é mais simples de se estabelecer palcos de comparação e de avaliação, haja vista a promulgação confirmada de valores padrão para certas medidas básicas. Dentro desse espírito, pode-se estipular que os levantamentos introduzidos aqui sirvam como primeira referência à realização de outros trabalhos na área de gerência e acesso a mídias alocadas a repositórios de dados distribuídos.

Na mesma linha de raciocínio, contata-se que outras seqüências de testes de medição poderiam ser encaminhadas, avaliando a integração do SGPOM com outros substratos (subsistemas) de comunicação — tais como o serviço de *streaming* de mídias, conhecido como *A/V Streams*, este proposto, recentemente, para CORBA [Mung99] — e outros tipos de repositórios — uma segunda implementação de *Datastore* foi concretizada, tendo o ObjectStore PSE (Seção 4.2) como produto de suporte (e, por conseguinte, como protocolo de armazenagem do OM). Parâmetros e medidas suplementares de desempenho (p. ex., $\overline{T}_{buffer}$) poderiam ser propostos, aumentando o horizonte de julgamento da filosofia da abordagem. Uma idéia sugerida foi a de se avaliar o Tempo de Invocação quando da recuperação de apenas uma primeira parte do conteúdo dos OMs, haja vista que, em atividades interativas, os fluxos de dados contidos nesses objetos são resgatados na forma de blocos ou fragmentos, preenchendo *buffers* de dados a serem consumidos na apresentação.

Capítulo 7

Conclusão

Findando a contribuição deste trabalho, passa-se às últimas impressões acerca da proposta do Serviço de Gerência da Persistência de Objetos Multimídia. Este capítulo é destinado a esse remate, trazendo considerações sucintas a respeito da abordagem apresentada, seguindo duas frentes distintas:

1. Fazer uma análise *pragmática* da adequação do SGPOM às metas visadas.
2. Ampliar as vertentes atuais do SGPOM, avaliando a sua aplicação em outras áreas de estudo e propondo possíveis extensões.

7.1 Análise

Nesta dissertação, as descrições funcional e arquitetural de um novo serviço são reportadas. O SGPOM, fruto de um conjunto de ponderações de projeto e de implementação, é um instrumento voltado para as tarefas de gerência e acesso a objetos multimídia persistentes e tem seu escopo definido no âmbito da Plataforma Multiware. A motivação que está por trás desse esforço é a de prover um serviço distribuído apropriado ao atendimento às restrições cominadas pela classe de objetos de mídia contínua. Sob essa perspectiva, o sistema é constituído por um arranjo hierárquico de componentes, modelado e inspirado naquele padronizado para o Serviço de Objetos Persistentes (CORBAservice POS) do OMG.

Para um julgamento da conformidade do SGPOM às expectativas levantadas inicialmente, passa-se a rever o rol de premissas que norteou a proposta consignada no Capítulo 1, qual seja:

- **Interagir com diferentes tipos de repositórios**: a arquitetura do serviço provê um mecanismo lógico de abstração (via os elementos *Datastore* e *Protocol*) dos vários modelos de implementação utilizados (OO, relacional, hierárquico, sistema de

arquivos, solução proprietária...) para a configuração dos repositórios multimídia. Isso oferece uma vantagem substancial, a de garantir a independência das aplicações-cliente para com os detalhes de projeto dos repositórios alocados para a manipulação dos dados (estados) de seus objetos.

- **Atender a necessidades de armazenagem e recuperação de diferentes aplicações multimídia**: o serviço, por se basear num padrão aberto (POS), não impõe restrições aos seus possíveis clientes. Estes, por sua vez, para se favorecerem das funcionalidades do SGPOM, devem ser projetados para cumprirem adequadamente o contrato de serviço firmado a partir das interfaces IDL anunciadas no Apêndice A.
- **Transparências de acesso aos dados e de localização dos repositórios**: um dos preceitos de maior prioridade dentro da proposta é a de se ter um serviço continuamente expansível e/ou mutável. Isso se traduz na possibilidade de introdução de ramos lógicos na estrutura, formando uma topologia incremental em árvore dos componentes, segundo a política de inserção de novos repositórios. Isso fica a cargo do administrador das instâncias do serviço, aliviando as aplicações-cliente (e seus projetistas) das tarefas e estratégias de alocação dos OMs aos repositórios multimídia mais adequados.
- **Suporte à concorrência**: a introdução dos objetos subcomponentes em cada camada do *framework* atende, entre outras, a essa demanda. Isso possibilitou a divisão das funcionalidades entre os subcomponentes, ficando os objetos Filhos responsáveis pelo atendimento às requisições concorrentes de E/S. Para esse fim, essas entidades devem ser implementadas segundo as técnicas de *multithreading*.
- **Robustez e flexibilidade**: a introdução em cada camada dos elementos Controladores e a provisão das atividades de orquestração (por intermédio do elemento Coordenador) dos vários SGPOM_XXs se constituem em iniciativas que almejam a especificação de um serviço robusto (ajustado/tolerante a falhas) e adaptável a novas restrições de desempenho.

É interessante expor neste espaço um modo alternativo de apreciação da proposta apresentada no decorrer deste texto, seguindo uma visão mais prática. Para tanto, passa-se a analisar a arquitetura do SGPOM sob o crivo de um conjunto de três medidas de qualidade: *autonomia*, *escalabilidade* e *cooperação* [Spen97].

O SGPOM pode ser visto como uma organização lógica bem estruturada de vários gerentes (componentes SGPOM_OMs), cada um relacionado a uma funcionalidade específica (atuar como raiz da estrutura ou representar um tipo ou formato de mídia). Esse arranjo garante o último dos critérios acima, o da cooperação, haja vista propiciar a integração desses gerentes no atendimento às várias requisições de E/S. Mais do que isso: cada gerente está inserido numa camada própria, a qual esconde da sua superior as funcionalidades que municia. Em cada um dos três últimos níveis (TYPEs, SUBTYPEs e SGPOM_DSs) da hierarquia, pode-se ter um número variado de SGPOM_XXs, o que leva a um balanceamento da carga entre eles e entre os diversos repositórios disseminados pelo ambiente distribuído.

O segundo dos parâmetros atendidos pelo SGPOM é o que diz respeito ao fator escalabilidade: como cada *driver* ou gerente tem associado a si, direta ou indiretamente, um ou mais repositórios, são factíveis a configuração da topologia da rede lógica de objetos e a sua contínua expansão, a fim de que se assegure um

tratamento adequado a grandes volumes de OMs, de acordo com as necessidades do domínio onde se insere a aplicação-cliente. Com o artifício auxiliar de monitoração do comportamento e da carga de trabalho dos componentes da hierarquia (e, por conseguinte, dos repositórios), pode-se identificar pontos de gargalo de desempenho do serviço ou novas demandas antes não previstas, implicando na adesão dinâmica de novos depósitos dedicados a mídia.

Finalmente, o critério da autonomia. Este é assegurado na medida em que cada um dos tipos de componentes (SGPOM_XXs) da estrutura realiza as suas funcionalidades em um servidor CORBA particular. Cada tipo tem associado a si uma interface IDL própria, a qual unicamente exporta o compromisso de serviço inerente ao ofício lógico de cada componente. Autonomia implica em eqüidade e independência operacional entre os repositórios multimídia incorporados pelo SGPOM; desse modo, é também mantida nos níveis superiores que compõem o serviço.

7.2 Trabalhos Futuros

O SGPOM, enquanto serviço aberto e genérico, pode servir de alvo para futuras modificações, adaptadas a novas necessidades ou, quiçá, a outras áreas de estudo. Algumas alternativas de projeto que porventura incorreriam em benefícios funcionais e de desempenho para o serviço são dadas a seguir:

- Integrar as funcionalidades dos componentes SGPOM_PO e LEADER em uma só entidade, eliminando uma camada da estrutura. A desvantagem trazida com essa decisão é a de se combalir as interpretações semânticas intrínsecas a cada componente, comprometendo a compreensão do papel lógico do novo objeto dentro da arquitetura.
- Inserir suporte à redundância no acesso aos repositórios. Isso se tornaria viável, por exemplo, se um mesmo *Datastore* (via o seu correspondente SGPOM_DS) pudesse logicamente se registrar a mais de um SUBTYPE e este, por sua vez, a mais de um TYPE. Isso incrementaria a confiabilidade relativa a cada repositório.
- Prover suporte à replicação dos dados contidos nos repositórios, assegurando a manutenção da sua disponibilidade (*availability*) — ver Capítulo 1.
- Introduzir mecanismos adicionais para a busca dos repositórios mais aliviados (com menor sobrecarga de trabalho).
- Acrescentar uma outra operação de orquestração: a de remoção dinâmica, via Coordenador, dos componentes cujas funcionalidades não se apresentem mais interessantes dentro da estrutura de uma instância. Essa tarefa implicaria na introdução de uma nova operação nas interfaces IDL das Fábricas de cada tipo de componente e aumentaria o grau de flexibilidade oferecido ao administrador do serviço.
- Adotar técnicas de *bufferização* otimizadas para armazenagem temporária dos OMs nos SGPOM_DSs (ver Capítulo 6) e no SGPOM_PO (para o caso das operações delegadas).
- Propor mecanismos outros que possibilitem maior segurança à manipulação dos OMs ao longo da hierarquia e nos repositórios.

Embora tenha sido projetado tendo em vista o cenário do Banco de Dados Multiware, o SGPOM pode ter suas funcionalidades inseridas em campos afins de estudo. No caso da Hipermídia [Verh98], por exemplo, o conjunto de nós e *links* de informação pode ser modelado como um feixe de objetos a serem armazenados e recuperados em uma ou mais instâncias do serviço, garantindo, daí, a natureza intrínseca de distribuição do conteúdo. Para tanto, repositórios adicionais para as ligações devem ser confeccionados e incorporados à estrutura, mediante algumas alterações no procedimento de inserção dos TYPEs e SUBTYPEs. Como há uma certa flexibilidade na atribuição dos rótulos de indicação dos novos tipos e formatos de mídia a serem introduzidos, pode-se fugir à regra comum de classificação oriunda do MIME, passando facilmente a estendê-la.

Na mesma linha de raciocínio, ferramentas voltadas a anotações de documentos multimídia e de suporte à construção de federações (comunidades) de agentes móveis [OMG98d] podem se beneficiar dos "insumos" trazidos com o SGPOM. Em se tratando de agente (objeto com certa autonomia) multimídia, a fim de que se assegure a sua propriedade de mobilidade, é inevitável, muitas vezes, o emprego de um mecanismo adequado à manutenção consistente do seu comportamento ativo (estado) ao longo de suas jornadas de trabalho [Silv97]. Como agravante a essa necessidade, encontra-se o fato da heterogeneidade dos nós do sistema por onde percorre, o que leva a possíveis divergências na representação dos seus dados. Essa tarefa pode ser delegada a um serviço à parte, visível dentro de um ambiente distribuído, que assegure as transparências de localização e acesso às várias versões dos seus estados dinâmicos — cada versão pode estar associada a uma plataforma de máquina distinta.

Apêndice A

Conjunto de Interfaces IDL do Serviço

A lista a seguir traz a definição dos módulos e interfaces que compõem o SGPOM, seguindo a hierarquia mostrada na Figura 5.10. Esse conjunto de definições deve servir de fonte a um compilador (*parser*) IDL[1] apropriado. Comentários pertinentes são introduzidos, trazendo uma maior compreensão das funcionalidades das entidades (componentes e subcomponentes) que integram o serviço.

1) Módulo *Sgpom*

SINOPSE: É o modulo central, no qual se inserem todos os outros, estes dedicados aos componentes do SGPOM e às definições comuns. *Sgpom*, por si só, não define nenhuma interface. É mapeado num *package* Java de mesmo nome.

```
module Sgpom
{ }
```

2) Módulo *Sgpom::Sgpom*

SINOPSE: Contém definições gerais usadas pelos componentes do SGPOM e pelas aplicações-cliente.

```
module Sgpom
{
struct SGPOM_PID{
 string name; //indica o nome do OM
 string owner; //indica o nome da aplic.(cliente do SGPOM) dona do OM
 short Type; //indica um índice de TYPE relativo ao LEADER
 short Subtype; //indica um índice de SUBTYPE relativo ao TYPE
 short Ds; //indica um índice de SGPOM_DS relativo ao SUBTYPE
```

[1] Para o compilador IDL do produto OrbixWeb3.0, isso de traduz na seguinte linha de comando (as definições, neste caso, estão contidas num só arquivo, denominado *SGPOM.idl*):

idl -F -C -jO Sgpom -jP br.unicamp.fee.dca -jQ SGPOM.idl

```
 any pid; //indica o template p/ recuperar o OM no Datastore
}; //SGPOM_PID

struct Pid_sign{
 SGPOM_PID pid; //guarda o SGPOM_PID do OM registrado
 string password; //senha criada pelo SGPOM_DS para remoção do OM
}; //Pid_sign

struct Monitor_values{
 long number_invocations; //núm. de invocações de E/S desde sua criação
 float invocations_per_minute; //a taxa de invocações de E/S por minuto
 string creation_date; //sua data de criação (no formato dd-mm-aaaa)
 float store_average_time; //tempo médio gasto numa camada para uma
      //invocação de armazenagem (Comm_way_store() ou Store()). Calculado
      //entre o recebimento da invocação e a entrega da resposta. Este
      //valor é marcado pelos CHILDs correspondentes.
 float restore_average_time; //idem ao anterior para Restore()
}; //Monitor_values➔guarda valores de monitoração de um objeto Central

struct Check_values{
 string date; //data
 string time; //hora
 SGPOM_PID pid; //identificador relativo ao GENERIC
}; //Check_values➔valores relativos ao último checkpoint de um SGPOM_XX

interface Comm_way{ };//indica o modo de comunicação. Deve ser herdada
      //para definir a forma (protocolo ou serviço) como o OM vai ser
      //transferido entre o SGPOM_DS e o cliente

interface Socketport_host:Comm_way{
 attribute string host; //indica a máquina p/ socket
 attribute string port; //indica a porta p/ socket
}; //Socketport_host➔especialização de Comm_way para sockets

interface INT_CLIENT{
//DEFINIÇÕES:
 typedef boolean NOTIFICATION; //alias
 enum INT_CLIENT_EXCEPTION_CODE{
  invalid_Comm_way, //o meio de comunicação está corrompido
  invalid_pid, //o SGPOM_PID passado está corrompido
  Comm_way_not_supported, //o meio de comunicação não é suportado
  operation_aborted, //a operação de leitura do objeto a partir do
      //buffer do PO não foi completada
  other_exception //exceção não catalogada (operação completada)
 }; //INT_CLIENT_EXCEPTION_CODE➔guarda o significado das exceções
 exception INT_CLIENT_EXCEPTION {INT_CLIENT_EXCEPTION_CODE error;};

//OPERAÇÕES:
 boolean Storage_notification(in SGPOM_PID pid, in NOTIFICATION
            notify)raises (INT_CLIENT_EXCEPTION);
 boolean Recovery_notification(in SGPOM_PID pid, in NOTIFICATION
            notify, in Comm_way cw) raises (INT_CLIENT_EXCEPTION);
}; //INT_CLIENT➔tipo herdável por uma aplic. cliente p/ se servir das
   //operações delegadas de armazenagem e recuperação dos OMs. Neste caso,
   //o SGPOM_PO assume o papel do cliente e recupera/armazena o objeto.
   //Esta interface é de callback, para notificação do cliente acerca
   //do sucesso ou não da operação delegada.

}; //module Sgpom
```

3) Módulo *Sgpom::Coordinator*

SINOPSE: Contém uma única interface, a que define o elemento Coordenador.

```
module Coordinator
{
interface COORDINATOR{
//DEFINIÇÕES:
 enum COORDINATOR_EXCEPTION_CODE{
  SGPOM_wrong_index, //índice de um SGPOM não existente
  SGPOM_not_created, //não se pode criar uma instância nova de um SGPOM
  type_invalid, //indica que o tipo passado como parâmetro não existe
  subtype_invalid, //indica que o subtipo passado não existe
  protocol_not_defined, //o protocolo não é suportado por nenhum SGPOM_DS
  operation_partial_completed, //durante invocações em grupo (de
      //checkpoint e de monitoração), um ou mais componentes não
      //conseguiram completá-la
  operation_aborted, //a operação não foi completada (mais de um erro/
      //exceção pode ter ocorrido por parte de um ou mais componentes)
  other_exception //guarda outra exceção afora as listadas acima
 }; //COORDINATOR_EXCEPTION_CODE→significado das exceções
 exception COORDINATOR_EXCEPTION {COORDINATOR_EXCEPTION_CODE error;};
 typedef Sgpom::Monitor_values Po;
 typedef Sgpom::Monitor_values OM_Leader;
 typedef Sgpom::Monitor_values OM_Generic; //aliases
 struct OM_Comp{
  string label; //indica o tipo, subtipo ou protocolo
  Sgpom::Monitor_values monitor_values;
  short COMP_index; //guarda um índice relativo ao Coordinator
 };//OM_Comp
 typedef OM_Comp OM_Type;
 typedef OM_Comp OM_Subtype;
 typedef OM_Comp Ds; //alias
 typedef sequence<OM_Type> OM_Types_roll; //cj. de valores para todos
      //os TYPEs de uma instância de SGPOM
 typedef sequence<OM_Subtype> OM_Subtypes_roll //idem para os SUBTYPEs
 typedef sequence<Ds> Ds_roll; //idem p/ os SGPOM_DSs
 struct Sgpom_instance{
  short index;
  string creation_date;
  Po my_PO;
  OM_Leader my_OM_Leader;
  OM_Generic my_OM_Generic;
  OM_Types_roll my_OM_Types_roll;
  OM_Subtypes_roll my_OM_Subtypes_roll;
  Ds_roll my_Ds_roll;
 }; //Sgpom_instance→guarda valores p/ uma dada instância de SGPOM
 typedef Sgpom::Check_values check_values; //alias
 typedef sequence<check_values> Checkpoint_values; //cj. de SGPOM_PIDs
      //relativos ao GENERIC, p/ todos os componentes de uma instância de
      //SGPOM, após realizarem a tarefa de checkpoint
 typedef sequence<Checkpoint_values> All_Checkpoint_values; //cj. de
      //todos os SGPOM_PIDS de todas as instâncias do serviço
 enum COORDINATOR_NOTIFICATION_CODE{
  component_crashed, //indica que um dos SGPOM_XXs (e Filhos) não existe
      //mais, ou está fora do ar
  component_overloaded, //indica que um dos SGPOM_XXs está
      //sobrecarregado. A solução a ser tomada fica a cargo do Coordenador
  component_not_responding, //o CONTROL indica que o SGPOM_XX não está
      //respondendo às suas invocações de controle
  Generic_not_responding, //qualquer Filho do SGPOM_XX pode estar
      //requisitando uma operação no GENERIC e este não está respondendo
  next_component_not_responding, //o CONTROL indica que o componente da
      //hierarquia abaixo dele não está respondendo às invocações dos
      // Filhos do SGPOM_XX associado a ele (somente p/ PO, Leader, Type
      //e Subtype)
```

```
  other_notification //outra notificação não classificada acima
 }; //COORDINATOR_NOTIFICATION_CODE➔guarda o significado das notificações
    //assíncronas vindas dos CONTROLs
//PARTE DE CONSULTA E MONITORAÇÃO:
 readonly attribute short SGPOM_number_instances; //número de
      //instâncias de SGPOM ativas

//_____________MONITORAÇÃO INDIVIDUAL_____________
 OM_Leader OM_Leader_monitor(in short SGPOM_index)
                              raises (COORDINATOR_EXCEPTION);
 OM_Generic OM_Generic_monitor(in short SGPOM_index)
                              raises (COORDINATOR_EXCEPTION);
 Po PO_monitor(in short SGPOM_index) raises (COORDINATOR_EXCEPTION);
 void OM_Types_monitor(in short SGPOM_index, out OM_Types_roll roll)
                              raises (COORDINATOR_EXCEPTION);
 void OM_Subtypes_monitor(in short SGPOM_index, out OM_Subtypes_roll
                              roll) raises (COORDINATOR_EXCEPTION);
 void Ds_monitor(in short SGPOM_index, out Ds_roll roll)
                              raises (COORDINATOR_EXCEPTION);

//_____________MONITORAÇÃO COMPLETA_____________
 void Sgpom_instance_monitor(in short SGPOM_index, out Sgpom_instance
                              sgpom) raises (COORDINATOR_EXCEPTION);

//PARTE DE INSTANCIAÇÃO/CRIAÇÃO
 short SGPOM_SPAWN() raises (COORDINATOR_EXCEPTION); //nova instância
 boolean SGPOM_OM_TYPE_insertion(in short SGPOM_index, in string
      type) raises (COORDINATOR_EXCEPTION);
      //insere em uma instância existente um novo gerente de tipo
 boolean SGPOM_OM_SUBTYPE_insertion(in short SGPOM_index, in short
type_index, in string subtype) raises (COORDINATOR_EXCEPTION);
      //insere em uma instância existente um novo gerente de subtipo
 boolean SGPOM_DS_insertion(in short SGPOM_index, in short
                              subtype_index, in string protocol)
                              raises (COORDINATOR_EXCEPTION);
      //insere em uma instância existente um novo gerente de repositório

//PARTE DE CHECKPOINT
 readonly attribute All_Checkpoint_values All_Components_values;
 boolean Checkpoint(in short SGPOM_index)
                  raises (COORDINATOR_EXCEPTION);

//PARTE DE NOTIFICAÇÃO
 oneway void notification_PO(in short SGPOM_index,
      in COORDINATOR_NOTIFICATION_CODE error);
 oneway void notification_OM_Leader(in short SGPOM_index,
      in short next_index, in COORDINATOR_NOTIFICATION_CODE error);
 oneway void notification_OM_Type(in short SGPOM_index, in short
      Comp_index, in short next_index,
      in COORDINATOR_NOTIFICATION_CODE error);
 oneway void notification_OM_Generic(in short SGPOM_index,
      in COORDINATOR_NOTIFICATION_CODE error);
 oneway void notification_OM_Subtype(in short SGPOM_index, in short
      Comp_index, in short next_index,
      in COORDINATOR_NOTIFICATION_CODE error);
 oneway void notification_DS(in short SGPOM_index, in short
      Comp_index, in COORDINATOR_NOTIFICATION_CODE error);
      //SGPOM_index➔índice da instância SGPOM
      //Comp_index➔índice que identifica o componente dentro da estrutura
      //next_index➔índice que identifica qual dos componentes sucessores
            //(do LEADER, do TYPE ou do SUBTYPE) tem problemas
      //error➔significado do erro/notificação

};//interface COORDINATOR

};//module Coordinator
```

4) Módulo *Sgpom::Sgpom_po*

SINOPSE: Aqui são definidas, em 2 submódulos, as interfaces de 4 tipos de subcomponentes: SGPOM_PO_FACTORY, SGPOM_PO_CONTROL, SGPOM_PO e SGPOM_PO_CHILD. Os submódulos são *Po* e *Po_child*.

```
module Sgpom_po{

module Po{

interface SGPOM_PO_FACTORY{ //Fábrica do SGPOM_PO e PO_CONTROL
//DEFINIÇÕES
 enum PO_FACTORY_EXCEPTION_CODE{
  SGPOM_wrong_index, //índice inválido p/ uma instância de SGPOM
  invalid_COORDINATOR_reference, //indica que a Referência ao
      //Coordenador está corrompida
  invalid_pid, //o SGPOM_PID para um SGPOM_PO está corrompido (para
      //fins de recriação)
  component_not_created, //não se pode criar uma instância de um SGPOM_PO
  operation_aborted //a operação em questão não foi completada
 }; //PO_FACTORY_EXCEPTION_CODE→significados das exceções
typedef Sgpom::SGPOM_PID Pid; //alias
exception PO_FACTORY_EXCEPTION {PO_FACTORY_EXCEPTION_CODE error;};
//OPERAÇÕES
void Create_PO(in Coordinator::COORDINATOR coord, in short
               SGPOM_index, out SGPOM_PO_CONTROL po_control)
               raises(PO_FACTORY_EXCEPTION);
void Recreate_PO(in Coordinator::COORDINATOR coord, in short
               SGPOM_index, in Pid pid, out SGPOM_PO_CONTROL
               po_control) raises(PO_FACTORY_EXCEPTION);
void GetPo_reference(out SGPOM_PO po) raises(PO_FACTORY_EXCEPTION);
      //op. chamada por uma instância de SGPOM_CLIENT
void Po_is_ready(in short SGPOM_index) raises(PO_FACTORY_EXCEPTION);
      //op. usada pelo PO_CONTROL: o seu PO está pronto p/ as reqs. de E/S
};//interface SGPOM_PO_FACTORY

interface SGPOM_PO_CONTROL{ //OBJETO CONTROLADOR
//DEFINIÇõES
 enum PO_CONTROL_EXCEPTION_CODE{
  invalid_LEADER_reference, //indica que a Referência para o LEADER,
      //passada pelo Coordenador, está "mal formada"
  invalid_GENERIC_reference, //idem para a Referência ao GENERIC
  operation_aborted, //operação não foi completada
  other_exception //guarda outra exceção afora as listadas acima
 };//guarda o significado das exceções
 exception PO_CONTROL_EXCEPTION {PO_CONTROL_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid;
 typedef Sgpom::Monitor_values monitor_values;
 typedef Sgpom::Check_values check_values; //alias

//ATRIBUTOS
 attribute Coordinator::COORDINATOR Coord;
 attribute SGPOM_PO Sgpom_po;
 attribute short SGPOM_index; //índice da instância de SGPOM
      //os valores destas variáveis são "fixados" pelo PO_FACTORY
//OPERAÇÕES
 boolean PO_attributes(in Sgpom_om::Om_leader::Om::SGPOM_OM_LEADER
      Sgpom_om_leader, in Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic) raises(PO_CONTROL_EXCEPTION);
      //chamada pelo Coordinator p/ passar as ObjRefs dos outros comps.
 boolean Startmonitor() raises(PO_CONTROL_EXCEPTION);
      //op. chamada pela Fábrica, acarretando na criação de uma thread de
      //monitoração periódica, a qual, de tempo em tempo, fará uma
```

```
      //invocação à operação monitored() do SGPOM_PO
 oneway void Setparameters(in monitor_values params);
      //essa operação fecha o ciclo iniciado pela anterior: o SGPOM_PO a
      //invoca depois de ter monitored() invocada
 void Getparameters(out monitor_values params)
                  raises(PO_CONTROL_EXCEPTION);
      //op. chamada pelo Coordenador na tarefa de monitoração
 oneway void Leader_not_responding();
      //SGPOM_PO_CHILD indica que não consegue interação com o LEADER
 void Checkpoint(out check_values values)
                  raises(PO_CONTROL_EXCEPTION);
      //op. chamada pelo Coordenador na tarefa de checkpoint
 oneway void Self_destroy();
      //quando ocorrer problema c/ o SGPOM_PO e a solução for a sua
      //substituição, o Coordenador irá notificar o CONTROL p/ que se
      //destrua (outro CONTROL irá ser criado)
};//interface SGPOM_PO_CONTROL

interface SGPOM_PO{ //Objeto Central

//DEFINIÇÕES
 enum PO_EXCEPTION_CODE{
  PO_Child_not_created, //resultado de uma invocação à oper. Create()
  operation_aborted //a operação não foi completada
 }; //PO_EXCEPTION_CODE→guarda o significado das exceções
 exception PO_EXCEPTION{ PO_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //alias
//ATRIBUTOS
 attribute Pid PO_pid;// SGPOM_PID do SGPOM_PO junto ao GENERIC usado
      //na sua reinstanciação
 attribute SGPOM_PO_CONTROL control;
 attribute ::Sgpom::Sgpom_om::Om_leader::Om::SGPOM_OM_LEADER
      Sgpom_om_leader;
 attribute ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic;
      //valores destas variáveis são "fixados" pelo CONTROL
//OPERAÇÕES
 oneway void monitored(); //para monitoração
 void Checkpoint(out Pid pid) raises (PO_EXCEPTION); //checkpoint
 void Create(out ::Sgpom::Sgpom_po::Po_child::SGPOM_PO_CHILD
      po_child) raises(PO_EXCEPTION);
      //essa op. simula a criação de um objeto Filho, o qual atenderá a
      //invocação de E/S
};//interface SGPOM_PO

};//module Po

module Po_child{

interface SGPOM_PO_CHILD{//Objeto Filho do SGPOM_PO
//DEFINIÇÕES
 enum PO_CHILD_EXCEPTION_CODE{
  invalid_PO_reference, //ObjRef do PO está corrompida
  type_invalid, //o tipo passado como parâm. a uma op. de E/S não existe
  subtype_invalid, //idem para o subtipo (formato)
  protocol_not_defined, //idem para o protocolo
  invalid_pid, //o identificador p/ o OM está corrompido
  invalid_Comm_way, //o objeto de comunicação do OM está corrompido
  operation_aborted, //operação não completada (mais de um erro/exceção)
  other_exception //outra exceção afora as listadas acima.
 }; //PO_CHILD_EXCEPTION_CODE→guarda o significado das exceções
 exception PO_CHILD_EXCEPTION {PO_CHILD_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //alias
 typedef Sgpom::Pid_sign pid_sign;
 typedef Sgpom::Comm_way comm_way;
//OPERAÇÕES
 void Create(in Po::SGPOM_PO sgpom_po) raises (PO_CHILD_EXCEPTION);
```

```
      //op. chamada por um SGPOM_PO
 void Register(in string name,in string owner, in string type,
              in string subtype, in string protocol, out pid_sign ps)
              raises (PO_CHILD_EXCEPTION); //operação de Registro do OM
void Comm_way_Store (in Pid pid, inout comm_way cw)
      raises(PO_CHILD_EXCEPTION);//primeira das ops. de armazenagem
boolean Store(in comm_way cw, in Pid pid) raises(PO_CHILD_EXCEPTION);
      //segunda das ops. de armazenagem
void delegated_Comm_way_Store(inout comm_way cw)
      raises(PO_CHILD_EXCEPTION);
      //primeira das ops. delegadas de armazenagem. O OM é passado ao
      //SGPOM_PO
void delegated_Store(in comm_way cw, in Pid pid, in Sgpom::INT_CLIENT
      ic) raises(PO_CHILD_EXCEPTION);
      //segunda das ops. delegadas de armazenagem. O OM já foi repassado ao
      //SGPOM_PO. Um dos parâms. é o SGPOM_PID do OM junto ao repositório
      //adequado e outro é a Referência à interface de callback do cliente.
void Restore(in Pid pid, inout comm_way cw)
      raises(PO_CHILD_EXCEPTION);
      //operação de recuperação do OM
void delegated_Restore(in Pid pid, in Sgpom::INT_CLIENT ic)
      raises(PO_CHILD_EXCEPTION);
      //operação delegada de recuperação do OM
boolean Delete(in pid_sign ps) raises(PO_CHILD_EXCEPTION);
      //operação de remoção do OM
};//interface SGPOM_PO_CHILD

};//module Po_child

}; //module Sgpom_po
```

5) Módulo *Sgpom::Sgpom_om*

SINOPSE: Aqui são definidos 4 submódulos, cada um deles relacionado a um SGPOM_OM. Dentro da arquitetura SGPOM, um SGPOM_OM pode ser um LEADER, um TYPE, um SUBTYPE, ou um GENERIC. Um outro submódulo (chamado de *Om_base*) contém as funcionalidades básicas pertencentes, em comum, a todos os SGPOM_OM_XXs. No final de cada ramo dessa hierarquia, se encontram os submódulos *Om* e *Om_child*, onde são definidas as interfaces dos subcomponentes Fábrica, Control, Central e Filho.

```
module Sgpom_om{

module Om_base{

module Om{

interface SGPOM_OM_CONTROL{ //CONTROLADOR (BÁSICO)
//DEFINIÇÕES
 enum OM_CONTROL_EXCEPTION_CODE{
  operation_aborted,
  other_exception
 };//guarda o significado das exceções
 exception OM_CONTROL_EXCEPTION {OM_CONTROL_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid;
 typedef Sgpom::Monitor_values monitor_values;
 typedef Sgpom::Check_values check_values;// //aliases
//ATRIBUTOS
 attribute Coordinator::COORDINATOR Coord;
 attribute SGPOM_OM Sgpom_om;
 attribute short SGPOM_index;
      //valores fixados pela Fábrica correspondente
```

```
 boolean Startmonitor() raises(OM_CONTROL_EXCEPTION);
      //ver Startmonitor() de SGPOM_PO_CONTROL
 oneway void Setparameters(in monitor_values params);
      //ver Setparameters() de SGPOM_PO_CONTROL
 void Getparameters(out monitor_values params)
      raises(OM_CONTROL_EXCEPTION); //op. chamada pelo Coordenador
 oneway void Next_Component_not_responding(in short next_index);
      //SGPOM_OM indica que não consegue comunicação com o próximo
      //componente da hierarquia passando o índice relativo deste
 void Checkpoint(out check_values values)
      raises(OM_CONTROL_EXCEPTION); //op. chamada pelo Coordenador
 oneway void Self_destroy(); //ver Self_destroy() de SGPOM_PO_CONTROL
}; //interface SGPOM_OM_CONTROL

interface SGPOM_OM{ //OBJETO CENTRAL(BÁSICO)
//DEFINIÇÕES
 enum OM_EXCEPTION_CODE{
  OM_Child_not_created, //resultado de uma invocação a oper. Create()
  operation_aborted
 }; //guarda o significado das exceções
 exception OM_EXCEPTION {OM_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //alias
//ATRIBUTO
 attribute Pid OM_pid; // SGPOM_PID do SGPOM_OM_XX junto ao GENERIC usado
      //na sua reinstanciação. Não sendo o caso, este atributo será nulo
 attribute SGPOM_OM_CONTROL control; //fixado pelo OM_FACTORY
                                     //correspondente
 attribute ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic;//fixado pelo Coordenador
 oneway void monitored();//op. chamada pelo OM_CONTROL p/ checkpoint
 void Checkpoint(out Pid pid) raises (OM_EXCEPTION);
      //op. chamada pelo OM_CONTROL
 void Create(out ::Sgpom::Sgpom_om::Om_base::Om_child::SGPOM_OM_CHILD
      om_child) raises(OM_EXCEPTION); //similar a Create() de SGPOM_PO
}; //interface SGPOM_OM

};//module Om

module Om_child{

interface SGPOM_OM_CHILD{
//DEFINIÇÕES
 enum OM_CHILD_EXCEPTION_CODE{ //FILHO (BÁSICO)
  invalid_OM_reference, //indica que a ObjRef do SGPOM_OM_XX passada pelo
      //próprio (OM) está corrompida
  type_invalid, //indica que o tipo passado como parâmetro não é suportado
  subtype_invalid, //idem para o formato de mídia
  protocol_not_defined, //idem para o protocolo do repositório
  invalid_pid, //o identificador p/ o OM está corrompido
  invalid_Comm_way, //objeto que encapsula o meio de comum. corrompido
  operation_aborted,
  other_exception
 };//significado das exceções
 exception OM_CHILD_EXCEPTION {OM_CHILD_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //aliases
 typedef Sgpom::Pid_sign pid_sign;
 typedef Sgpom::Comm_way comm_way;
//OPERAÇÕES
 void Create(in Om::SGPOM_OM sgpom_om) raises (OM_CHILD_EXCEPTION);
      //op. chamada por um SGPOM_OM_XX
 void Register(in string name, in string owner,in string type,
      in string subtype,in string protocol, out pid_sign ps)
      raises (OM_CHILD_EXCEPTION);//op. E/S de registro do OM
 void Comm_way_Store(in Pid pid, inout comm_way cw)
      raises (OM_CHILD_EXCEPTION);
      //primeira das ops. relacionadas à armazenagem de um OM
 boolean Store(in comm_way cw,in Pid pid) raises(OM_CHILD_EXCEPTION);
```

```
      //segunda das ops. relacionadas à armazenagem de um OM
 void Restore(in Pid pid, inout comm_way cw)
      raises(OM_CHILD_EXCEPTION); //op. de E/S p/ resgate do OM
 boolean Delete(in pid_sign ps) raises(OM_CHILD_EXCEPTION);
      //op. de E/S p/ remoção do OM
};//interface SGPOM_OM_CHILD

};//module Om_child

 };//module Om_base

module Om_leader{

module Om{

interface SGPOM_OM_LEADER_FACTORY{ //FÁBRICA DO LEADER E SEU CONTROL
//DEFINIÇÕES
 enum OM_LEADER_FACTORY_EXCEPTION_CODE{
  SGPOM_wrong_index, //índice inválido p/ uma instância de SGPOM
  invalid_COORDINATOR_reference, //ObjRef ao Coordenador corrompida
  invalid_pid, //identificador p/ o SGPOM_PO corrompido
  component_not_created, //não se pode criar uma instância de um LEADER
  operation_aborted
 }; //OM_LEADER_FACTORY_EXCEPTION_CODE
 typedef Sgpom::SGPOM_PID Pid; //alias
 exception OM_LEADER_FACTORY_EXCEPTION
                  {OM_LEADER_FACTORY_EXCEPTION_CODE error;};
//OPERAÇÕES
 void Create_OM_Leader(in Coordinator::COORDINATOR coord,
    in short SGPOM_index, out SGPOM_OM_LEADER_CONTROL leader_control)
    raises(OM_LEADER_FACTORY_EXCEPTION);//ops. chamadas pelo Coordenador
 void Recreate_OM_Leader(in Coordinator::COORDINATOR coord,
    in short SGPOM_index, in Pid pid, out SGPOM_OM_LEADER_CONTROL
    leader_control)raises(OM_LEADER_FACTORY_EXCEPTION);
};//interface SGPOM_OM_LEADER_FACTORY

interface SGPOM_OM_LEADER_CONTROL:Om_base::Om::SGPOM_OM_CONTROL{
//DEFINIÇÕES
 enum OM_LEADER_CONTROL_EXCEPTION_CODE{ //significado das exceções
  invalid_GENERIC_reference, //ObjRef ao GENERIC corrompida
  invalid_PO_reference, ObjRef ao SGPOM_PO corrompida
  operation_aborted,
  other_exception
 }; //OM_LEADER_CONTROL_EXCEPTION_CODE
 exception OM_LEADER_CONTROL_EXCEPTION
                  {OM_LEADER_CONTROL_EXCEPTION_CODE error;};
//OPERAÇÕES
 boolean OM_LEADER_attributes(in Sgpom_po::Po::SGPOM_PO Sgpom_po,
      in ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic) raises(OM_LEADER_CONTROL_EXCEPTION);
      //chamada pelo Coordinator p/ passar as ObjRefs dos outros comps.
};//interface SGPOM_OM_LEADER_CONTROL

interface SGPOM_OM_LEADER:Om_base::Om::SGPOM_OM{
//DEFINIÇÕES
 enum OM_LEADER_EXCEPTION_CODE{ //código das exceções
  type_invalid, //tipo não existente
  TYPE_reference_invalid, //ObjRef. ao OM_Type corrompida
  operation_aborted
};
 exception OM_LEADER_EXCEPTION {OM_LEADER_EXCEPTION_CODE error;};
//ATRIBUTO
 attribute Sgpom_po::Po::SGPOM_PO Sgpom_po; //fixado pelo Coordenador
//OPERAÇÕES
 void Subscription_Type(
      in ::Sgpom::Sgpom_om::Om_type::Om::SGPOM_OM_TYPE sgpom_om_type,
```

```
      in string type,in short Type_index)
      raises (OM_LEADER_EXCEPTION);
      //um novo TYPE deve se registrar com o LEADER a partir desta op.,
      //passando-lhe a sua ObjRef e o seu tipo associado
 oneway void Overload(in short Type_index);
      //op. chamada por um TYPE indicando sobrecarga de trabalho! Se houver
      //outro TYPE com o mesmo rótulo, o LEADER deve repassar a este as
      //próximas ops. de registro. Type_index é o índice deste TYPE dentro
      //da instância do SGPOM
}; //interface SGPOM_OM_LEADER

};//module Om

module Om_child{

interface SGPOM_OM_LEADER_CHILD:Om_base::Om_child::SGPOM_OM_CHILD
{ };//interface SGPOM_OM_LEADER_CHILD → não há nenhuma nova operação

};//module Om_child

 };//module Om_leader

module Om_type{

module Om{

interface SGPOM_OM_TYPE_FACTORY{
//DEFINIÇÕES
 enum OM_TYPE_FACTORY_EXCEPTION_CODE{ //significado das exceções
  SGPOM_wrong_index, //índice inválido p/ uma instância de SGPOM
  invalid_COMP_index, //índice inválido p/ um componente→ p. ex. fora de
                      //uma faixa aceitável – negativo)
  invalid_COORDINATOR_reference,
  invalid_pid, //identificador p/ o TYPE corrompido
  component_not_created, //não se pode criar uma instância de TYPE
  operation_aborted
 };//OM_TYPE_FACTORY_EXCEPTION_CODE
 typedef Sgpom::SGPOM_PID Pid; //alias
 exception OM_TYPE_FACTORY_EXCEPTION {OM_TYPE_FACTORY_EXCEPTION_CODE
                                      error;};
//OPERAÇÕES
 void Create_OM_Type(in Coordinator::COORDINATOR coord,
            in short SGPOM_index, in short COMP_index,
            out SGPOM_OM_TYPE_CONTROL type_control)
            raises(OM_TYPE_FACTORY_EXCEPTION);
 void Recreate_OM_Type(in Coordinator::COORDINATOR coord, in short
            SGPOM_index, in short COMP_index,in Pid pid,
            out SGPOM_OM_TYPE_CONTROL type_control)
            raises(OM_TYPE_FACTORY_EXCEPTION);
            //ops. chamada pelo Coordenador
};//interface SGPOM_OM_TYPE_FACTORY

interface SGPOM_OM_TYPE_CONTROL:Om_base::Om::SGPOM_OM_CONTROL{

 enum OM_TYPE_CONTROL_EXCEPTION_CODE{ //significado das exceções
  type_invalid, //indica que o tipo passado como parâmetro não é suportado
  invalid_GENERIC_reference, //ObjRef ao GENERIC corrompida
  invalid_LEADER_reference, //idem para a Referência ao LEADER
  operation_aborted,
  other_exception
 };
 exception OM_TYPE_CONTROL_EXCEPTION {OM_TYPE_CONTROL_EXCEPTION_CODE
                                      error;};
//ATRIBUTOS
 attribute short COMP_index; //índice relativo do TYPE na instância
//OPERAÇÕES
```

```
 boolean OM_TYPE_attributes(in string Type,
      in Om_leader::Om::SGPOM_OM_LEADER Sgpom_om_leader,
      in ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic) raises(OM_TYPE_CONTROL_EXCEPTION);
      //chamada pelo Coordenador p/ passar as ObjRefs dos outros comps.
};//interface SGPOM_OM_TYPE_CONTROL

interface SGPOM_OM_TYPE:Om_base::Om::SGPOM_OM{
//DEFINIÇÕES
 enum OM_TYPE_EXCEPTION_CODE{//significado das exceções
  subtype_invalid, //nenhum SUBTYPE para tratar o formato passado como
                   //parâmetro
  SUBTYPE_reference_invalid, //ObjRef. passada pelo SUBTYPE corrompida
  operation_aborted
 }; //OM_TYPE_EXCEPTION_CODE
 exception OM_TYPE_EXCEPTION {OM_TYPE_EXCEPTION_CODE error;};
//ATRIBUTOS
 attribute string Type; //o TIPO de mídia associado
 attribute Om_leader::Om::SGPOM_OM_LEADER Sgpom_om_leader;
      //atributos fixados via TYPE_CONTROL
 void Subscription_Subtype(
      in ::Sgpom::Sgpom_om::Om_subtype::Om::SGPOM_OM_SUBTYPE
      sgpom_om_subtype, in string subtype, in short SUBTYPE_index)
      raises (OM_TYPE_EXCEPTION);
      //um novo SUBTYPE deve se registrar com o TYPE a partir desta op.,
      //passando-lhe a sua ObjRef e o seu subtipo associado
 oneway void Overload(in short SUBTYPE_index);
      //op. chamada por um SUBTYPE indicando sobrecarga de trabalho! Se
      //houver outro SUBTYPE com o mesmo rótulo, o TYPE deve repassar a
      //este as próximas ops. de registro. SUBTYPE_index é o índice deste
      //SUBTYPE dentro da instância do SGPOM
}; //interface SGPOM_OM_TYPE

};//module Om

module Om_child{

interface SGPOM_OM_TYPE_CHILD:Om_base::Om_child::SGPOM_OM_CHILD
{ }; //interface SGPOM_OM_TYPE_CHILD

}; //module Om_child

 };//module Om_type

module Om_subtype{

module Om{

interface SGPOM_OM_SUBTYPE_FACTORY{
//DEFINIÇÕES
 enum OM_SUBTYPE_FACTORY_EXCEPTION_CODE{
                  //similar ao OM_TYPE_FACTORY_EXCEPTION_CODE
  SGPOM_wrong_index,
  invalid_COMP_index,
  invalid_COORDINATOR_reference,
  invalid_pid,
  component_not_created,
  operation_aborted
 }; //significado das exceções
 typedef Sgpom::SGPOM_PID Pid; //alias
 exception OM_SUBTYPE_FACTORY_EXCEPTION
            {OM_SUBTYPE_FACTORY_EXCEPTION_CODE error;};
//OPERAÇÕES
 void Create_OM_Subtype(in Coordinator::COORDINATOR coord,
      in short SGPOM_index, in short COMP_index,
      out SGPOM_OM_SUBTYPE_CONTROL subtype_control)
```

```
      raises(OM_SUBTYPE_FACTORY_EXCEPTION);
 void Recreate_OM_Subtype(in Coordinator::COORDINATOR coord,
      in short SGPOM_index, in short COMP_index, in Pid pid,
      out SGPOM_OM_SUBTYPE_CONTROL subtype_control)
      raises(OM_SUBTYPE_FACTORY_EXCEPTION);
      //ops. chamadas pelo Coordenador p/ instanciar ou substituir um
      //SUBTYPE
};//interface SGPOM_OM_SUBTYPE_FACTORY

interface SGPOM_OM_SUBTYPE_CONTROL:Om_base::Om::SGPOM_OM_CONTROL{
//DEFINIÇÕES
 enum OM_SUBTYPE_CONTROL_EXCEPTION_CODE{ //código das exceções
  subtype_invalid, //o rótulo para o formato de mídia não é válido (nulo)
  type_invalid, //tipo de mídia não válido (string nula)
  invalid_GENERIC_reference, //ObjRef ao GENERIC corrompida
  invalid_TYPE_reference, //idem para o TYPE ligado logicamente
            //ao SUBTYPE associado a este CONTROL
  operation_aborted,
  other_exception
 }; //OM_SUBTYPE_CONTROL_EXCEPTION_CODE
 exception OM_SUBTYPE_CONTROL_EXCEPTION
                  {OM_SUBTYPE_CONTROL_EXCEPTION_CODE error;};
//ATRIBUTOS
 attribute short COMP_index; //índice relativo dentro da instância
//OPERAÇÕES
 boolean OM_SUBTYPE_attributes(in string Subtype,
      in Om_type::Om::SGPOM_OM_TYPE Sgpom_om_type,
      in ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic) raises(OM_SUBTYPE_CONTROL_EXCEPTION);
      //similar a OM_TYPE_attributes
};//interface SGPOM_OM_SUBTYPE_CONTROL

interface SGPOM_OM_SUBTYPE:Om_base::Om::SGPOM_OM{
//DEFINIÇÕES
 enum OM_SUBTYPE_EXCEPTION_CODE{
  protocol_invalid, //não existe um SGPOM_DS para este protocolo
  DS_reference_invalid, //ObjRef de um SGPOM_DS corrompida
  operation_aborted
 };//OM_SUBTYPE_EXCEPTION_CODE
 exception OM_SUBTYPE_EXCEPTION {OM_SUBTYPE_EXCEPTION_CODE error;};
//ATRIBUTOS
 attribute string Subtype; //formato de mídia associado
 attribute Om_type::Om::SGPOM_OM_TYPE Sgpom_om_type;
      //atributos fixados via SUBTYPE_CONTROL
//OPERAÇÕES
 void Subscription_Ds(in ::Sgpom::Sgpom_ds::Ds::SGPOM_DS sgpom_ds,
in string protocol, in short DS_index) raises (OM_SUBTYPE_EXCEPTION);
      //um novo SGPOM_DS deve se registrar com o SUBTYPE a partir desta
      //op., passando-lhe a sua ObjRef e o seu protocolo associado
 oneway void Overload(in short DS_index);
      //op. chamada por um SGPOM_DS indicando sobrecarga de trabalho! Se
      //houver outro com o mesmo rótulo, o SUBTYPE deve repassar a este as
      //próximas ops. de registro. DS_index é o índice deste SGPOM_DS
      //dentro da instância do SGPOM
};//interface SGPOM_OM_SUBTYPE

}; //module Om

module Om_child{

interface SGPOM_OM_SUBTYPE_CHILD:Om_base::Om_child::SGPOM_OM_CHILD
{ };//interface SGPOM_OM_SUBTYPE_CHILD

};//module Om_child

};//module Om_subtype
module Om_generic{
```

```
interface SGPOM_OM_GENERIC_FACTORY{
//DEFINIÇÕES
 enum OM_GENERIC_FACTORY_EXCEPTION_CODE{ //significado das exceções
  SGPOM_wrong_index, //índice inválido p/ a instância a que pertence
  invalid_COORDINATOR_reference, //ObjRef ao Coordinator corrompida
  component_not_created, //não se pôde criar uma instancia de GENERIC
  operation_aborted
 };//OM_GENERIC_FACTORY_EXCEPTION_CODE
 exception OM_GENERIC_FACTORY_EXCEPTION
            {OM_GENERIC_FACTORY_EXCEPTION_CODE error;};
//OPERAÇÕES
 void Create_OM_Generic(in Coordinator::COORDINATOR coord,
 in short SGPOM_index, out SGPOM_OM_GENERIC_CONTROL generic_control)
 raises(OM_GENERIC_FACTORY_EXCEPTION);
      //op. chamada pelo Coordenador para criação de um novo GENERIC
      //NÃO HÁ SUBSTITUIÇÃO DE GENERICs
}; //interface SGPOM_OM_GENERIC_FACTORY

interface SGPOM_OM_GENERIC_CONTROL:Sgpom_ds::Ds::SGPOM_DS_CONTROL{
//DEFINIÇÕES
 enum OM_GENERIC_CONTROL_EXCEPTION_CODE{ //significado das exceções
  invalid_LEADER_reference, //ObjRef do LEADER passado pelo
                            //Coordenador está corrompida
  invalid_PO_reference, //idem para a Referência ao SGPOM_PO
  operation_aborted,
  other_exception
 };//OM_GENERIC_CONTROL_EXCEPTION_CODE
 exception OM_GENERIC_CONTROL_EXCEPTION
            {OM_GENERIC_CONTROL_EXCEPTION_CODE error;};
//OPERAÇÕES
 boolean OM_GENERIC_attributes( in Om_leader::Om::SGPOM_OM_LEADER
      Sgpom_om_leader, in Sgpom_po::Po::SGPOM_PO Sgpom_po)
      raises(OM_GENERIC_CONTROL_EXCEPTION);
      //op. chamada pelo Coordinator p/ passar as ObjRefs dos outros
      //componentes da estrutura básica (SGPOM_PO e LEADER).
}; //interface SGPOM_OM_GENERIC_CONTROL

interface SGPOM_OM_GENERIC:Sgpom_ds::Ds::SGPOM_DS{ //HERDA DE SGPOM_DS
//ATRIBUTOS
 attribute Sgpom_po::Po::SGPOM_PO Sgpom_po; //o SGPOM_PO associado
 attribute Om_leader::Om::SGPOM_OM_LEADER Sgpom_om_leader; //LEADER
      //atributos fixados pelo Coordenador
//OPERAÇÕES
void Create(out Om_child::SGPOM_OM_GENERIC_CHILD gen_child)
      raises(Sgpom_ds::Ds::SGPOM_DS::DS_EXCEPTION);
      //nova op. Create: é usada pelos comps. do SGPOM p/ realizarem
      //checkpoint
      //ATENÇÃO: NÃO É OVERRIDING, pois não mantém a mesma assinatura!!!
}; //interface SGPOM_OM_GENERIC

}; //module Om

module Om_child{

interface SGPOM_OM_GENERIC_CHILD:Sgpom_ds::Ds_child::SGPOM_DS_CHILD{
//DEFINIÇÕES
 enum OM_GENERIC_CHILD_EXCEPTION_CODE{ //significado das exceções
  component_invalid, //não existe um SGPOM_XX com tal denominação (string)
  invalid_COMP_index, //indica que o índice do objeto Central passado
                      //para o GENERIC é inválido (nulo)
  invalid_pid, //identificador p/ o subcomponente Central corrompido
  invalid_Comm_way, //objeto que encapsula a comunicação do estado do
                    //subcomponente está corrompido

  operation_aborted
 };
```

```
 typedef Sgpom::SGPOM_PID Pid; //aliases
 typedef Sgpom::Comm_way comm_way;
 exception OM_GENERIC_CHILD_EXCEPTION {OM_GENERIC_CHILD_EXCEPTION_CODE
                                       error;};
//OPERAÇÕES
void Register_Component(in string component,in short COMP_index,
      out Pid pid) raises (OM_GENERIC_CHILD_EXCEPTION);
void Comm_way_Store_Component(in Pid pid, inout comm_way cw)
      raises (OM_GENERIC_CHILD_EXCEPTION);
boolean Store_Component(in comm_way cw,in Pid pid)
      raises (OM_GENERIC_CHILD_EXCEPTION);
void Restore_Component(in Pid pid,inout comm_way cw)
            raises (OM_GENERIC_CHILD_EXCEPTION);
      //ops. Chamadas por qualquer um dos subcomponentes do SGPOM p/
      //Checkpoint
};//interface SGPOM_OM_GENERIC_CHILD

};//module Om_child

};//module Om_generic

};//module Sgpom_om
```

6) Módulo *Sgpom::Sgpom_ds*

SINOPSE: Onde são definidas, em 2 submódulos (*Ds* e *Ds_child*), as interfaces de 4 subcomponentes: SGPOM_DS_FACTORY, SGPOM_DS_CONTROL, SGPOM_DS e SGPOM_DS_CHILD.

```
module Sgpom_ds{

module Ds{

interface SGPOM_DS_FACTORY{
//DEFINIÇÕES
 enum DS_FACTORY_EXCEPTION_CODE{ //significado das exceções
  SGPOM_wrong_index, //índice inválido p/ a instância de SGPOM
  invalid_COMP_index, //índice inválido p/ um componente → p. ex. fora de
            //uma faixa aceitável (negativo ou nulo)
  invalid_COORDINATOR_reference, //a ObjRef passada está corrompida
  invalid_pid, //identificador p/ o SGPOM_DS corrompido
  component_not_created, //não se pôde criar uma instancia de SGPOM_DS
  operation_aborted
};
 typedef Sgpom::SGPOM_PID Pid; //alias
 exception DS_FACTORY_EXCEPTION {DS_FACTORY_EXCEPTION_CODE error;};
 void Create_DS(in Coordinator::COORDINATOR coord,
      in short SGPOM_index, in short COMP_index,
      out SGPOM_DS_CONTROL ds_control)
      raises(DS_FACTORY_EXCEPTION); //ops. chamada pelo Coordenador
 void Recreate_DS(in Coordinator::COORDINATOR coord,
      in short SGPOM_index, in short COMP_index, in Pid pid,
      out SGPOM_DS_CONTROL ds_control) raises(DS_FACTORY_EXCEPTION);
}; //interface SGPOM_DS_FACTORY

interface SGPOM_DS_CONTROL{
//DEFINIÇÕES
 enum DS_CONTROL_EXCEPTION_CODE{ //significado das exceções
  invalid_SUBTYPE_reference, //ObjRef do SUBTYPE associado corrompida"
  invalid_GENERIC_reference, //idem para a Referência ao GENERIC
  invalid_protocol, //a string do tipo de protocolo passada pelo
      //Coordenador é inválida
  operation_aborted,
```

```
  other_exception
 };
 exception DS_CONTROL_EXCEPTION {DS_CONTROL_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //aliases
 typedef Sgpom::Monitor_values monitor_values;
 typedef Sgpom::Check_values check_values;
//ATRIBUTOS
 attribute Coordinator::COORDINATOR Coord;
 attribute SGPOM_DS Sgpom_ds;
 attribute short SGPOM_index;
 attribute short COMP_index; //índices correspondente ao SGPOM e ao comp.
      //os valores destas variáveis são "fixados" pelo DS_FACTORY
//OPERAÇÕES
 boolean DS_attributes(in string Protocol,
      in Sgpom_om::Om_subtype::Om::SGPOM_OM_SUBTYPE Sgpom_om_subtype,
      in Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC Sgpom_om_generic)
      raises(DS_CONTROL_EXCEPTION);
      //op. chamada pelo Coordinator p/ passar as ObjRefs dos outros comps.
 boolean Startmonitor() raises(DS_CONTROL_EXCEPTION);
      //op. chamada pelo DS_FACTORY. Tal operação acarretará na criação de
      //uma thread de monitoração periódica, a qual, de tempo em tempo,
      //fará uma invocação à oper. monitored() de SGPOM_DS.
 oneway void Setparameters(in monitor_values params);
      //op. chamada pelo SGPOM_DS como resultado de uma invocação à sua op.
      //monitored()
 void Getparameters(out monitor_values params)
      raises(DS_CONTROL_EXCEPTION); //op. chamada pelo Coordenador p/
      //receber os valores de monitoração do SGPOM_DS associado
 void Checkpoint(out check_values values)
      raises(DS_CONTROL_EXCEPTION);
      //op. chamada pelo Coordenador p/ checkpoint
 oneway void Self_destroy();
      //quando da substituição do SGPOM_DS, o Coordinator irá notificar o
      //DS_CONTROL p/ que se destrua (outro CONTROL irá ser criado)
};//interface SGPOM_DS_CONTROL

interface SGPOM_DS{
//DEFINIÇÕES
 enum DS_EXCEPTION_CODE{ //significado das exceções
  DS_Child_not_created, //resultado de uma invocação à oper. Create()
  operation_aborted
 }; //DS_EXCEPTION_CODE
 exception DS_EXCEPTION {DS_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //alias
//ATRIBUTOS
 attribute Pid DS_pid; //este valor é fixado quando do registro do
      //SGPOM_DS junto ao GENERIC
 attribute SGPOM_DS_CONTROL control;
 attribute string Protocol;
 attribute Sgpom_om::Om_subtype::Om::SGPOM_OM_SUBTYPE
      Sgpom_om_subtype;
 attribute ::Sgpom::Sgpom_om::Om_generic::Om::SGPOM_OM_GENERIC
      Sgpom_om_generic;
      //valores destes campos são "fixados" pelo Coordenador
 oneway void monitored(); //op. chamada pelo DS_CONTROL p/ checkpoint
 void Checkpoint(out Pid pid) raises(DS_EXCEPTION);
 void Create(out Ds_child::SGPOM_DS_CHILD ds_child)
      raises(DS_EXCEPTION);
      //simboliza a criação de um obj. Filho p/ atender à req. de E/S
};//interface SGPOM_DS

};//module Ds

module Ds_child{

interface SGPOM_DS_CHILD{
//DEFINIÇÕES
```

```
 enum DS_CHILD_EXCEPTION_CODE{ //código das exceções
  invalid_DS_reference, //ObjRef do SGPOM_DS corrompida
  invalid_Comm_way, //objeto de comunicação inválido ou corrompido
  operation_aborted,
  other_exception
 }
 exception DS_CHILD_EXCEPTION {DS_CHILD_EXCEPTION_CODE error;};
 typedef Sgpom::SGPOM_PID Pid; //alias
 typedef Sgpom::Pid_sign pid_sign;
 typedef Sgpom::Comm_way comm_way;
 void Create(in Ds::SGPOM_DS sgpom_ds)raises (DS_CHILD_EXCEPTION);
      //op. chamada por um SGPOM_DS para a sua "criação" (passa-se a
      //Referência ao SGPOM_DS pai
 void Register(in string name,in string owner, in string type,
      in string subtype, in string protocol, out pid_sign ps)
      raises (DS_CHILD_EXCEPTION); //op. de Registro do OM
 void Comm_way_Store(in Pid pid, inout comm_way cw)
      raises (DS_CHILD_EXCEPTION);
      //op. chamada por uma das threads filhas vindas do SUBTYPE associado
 boolean Store(in comm_way cw,in Pid pid) raises(DS_CHILD_EXCEPTION);
      //idem
 void Restore(in Pid pid,inout comm_way cw)
      raises(DS_CHILD_EXCEPTION); //op. de recuperação do OM
 boolean Delete(in pid_sign ps) raises(DS_CHILD_EXCEPTION);
      //op. de resgate do OM
}; //interface SGPOM_DS_CHILD

}; //module Ds_child

}; //module Sgpom_ds
```

Apêndice B

Fluxos de Interações

Este anexo está reservado à apresentação dos fluxos de possíveis interações entre as entidades do SGPOM, consoante a notação MSC (*Message Sequence Chart*) [Mauw94]. Essas interações compreendem as tarefas de orquestração (instanciação, monitoração e *checkpoint*) dos subcomponentes, bem como as operações de E/S entre as camadas do serviço, introduzidas no quinto capítulo deste documento.

Um diagrama de fluxo de mensagens (doravante chamado de MSC) não é uma descrição completa do comportamento de um sistema; expressa, simplesmente, um delineamento (traçado) de execução entre alguns de seus componentes via uma notação gráfica ou textual conveniente. Um MSC, pois, contém a descrição da comunicação *assíncrona* entre *instâncias* desses componentes seguindo uma *linguagem* padronizada. Uma coleção de MSCs, por sua vez, pode ser usada para abranger o nível de detalhe de um sistema, garantindo a sua observação sob cenários distintos.

Em um MSC, uma instância é uma entidade abstrata da qual se pode observar (parte de) a interação com outras do mesmo jaez ou com o próprio ambiente a que pertence. Por outro lado, a linguagem de descrição de um MSC é rica em primitivas para representação das tarefas associadas às instâncias, tais como ações locais, temporizadores (com operações de *set*, *reset* e *time-out*), criação e cessação dos processos e coregiões.

Conforme a notação gráfica de um MSC, uma instância é denotada como um eixo vertical ao longo do qual o tempo corre no sentido norte-sul (de cima para baixo). A comunicação entre duas instâncias é representada por uma seta que se inicia no emissor da mensagem e termina no receptor. Embora as atividades ao longo de um eixo estejam completamente ordenadas, não se assume a noção de tempo global. A única restrição de dependência a ser seguida é: uma mensagem deve ser enviada antes de ser recebida. A seguir, o conjunto de MSCs para o SGPOM com a notação gráfica.

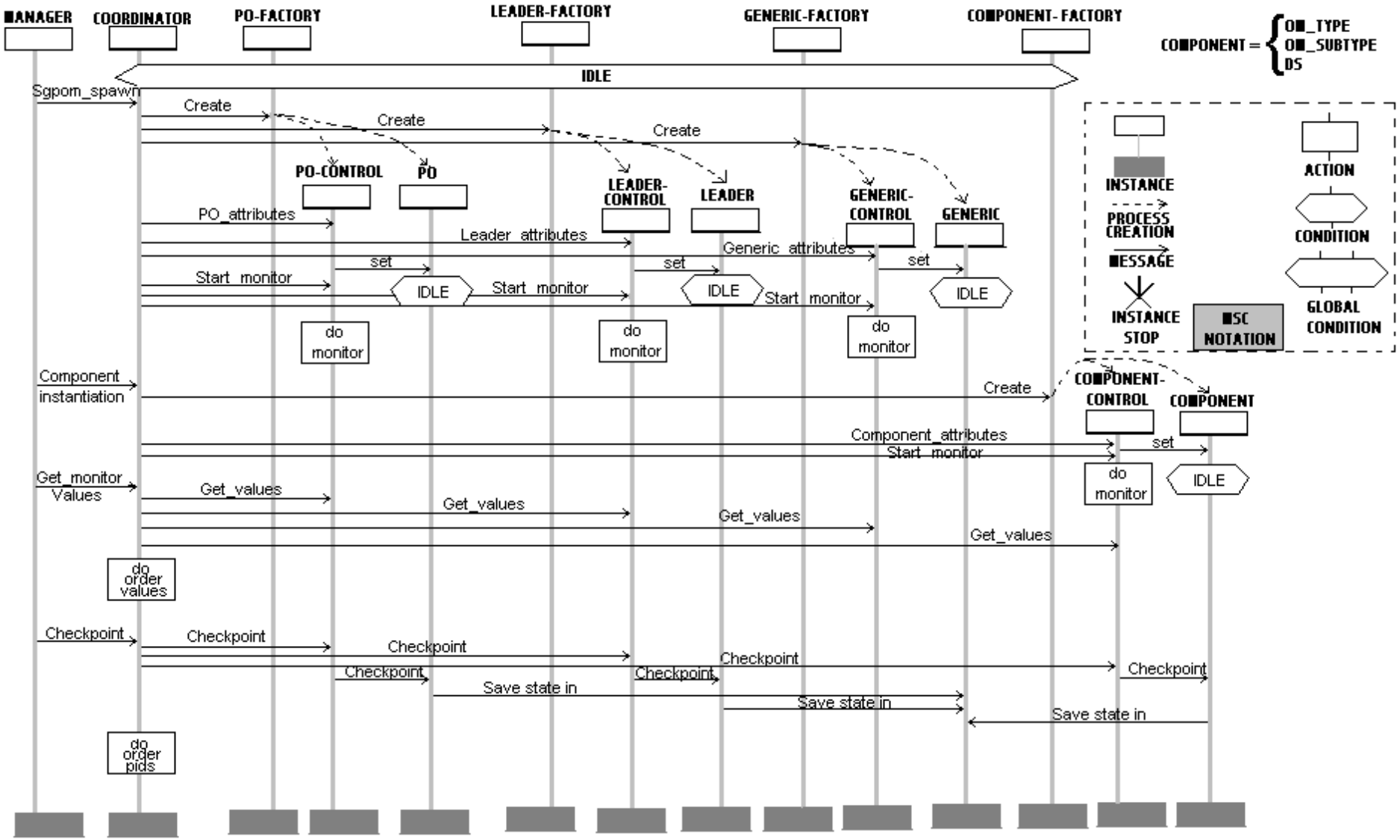

***Figura B.1**: Diagrama MSC para as tarefas de orquestração do elemento Coordenador do SGPOM.*

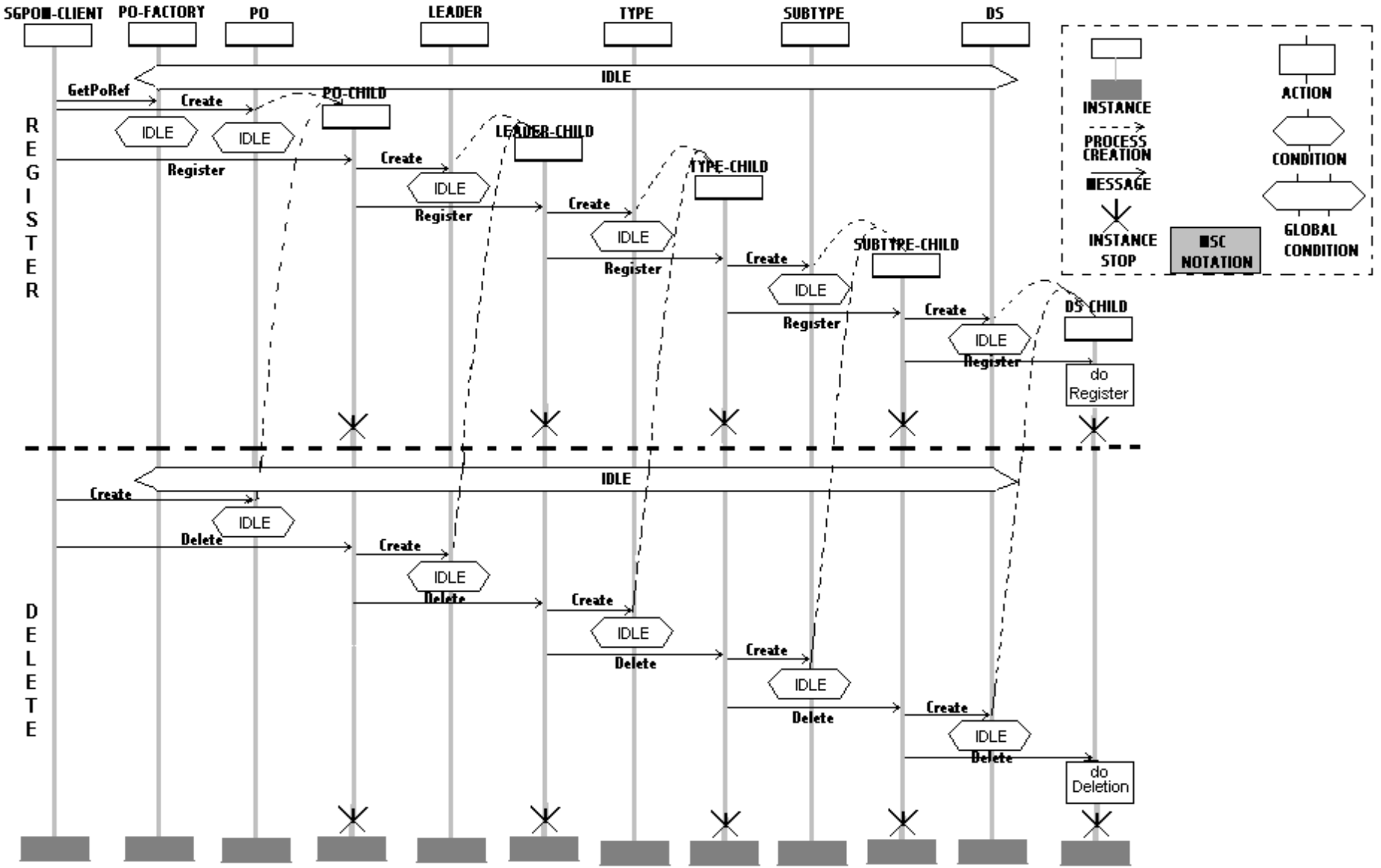

***Figura B.2**: Diagrama MSC para a propagação das operações de gerência entre as camadas do SGPOM.*

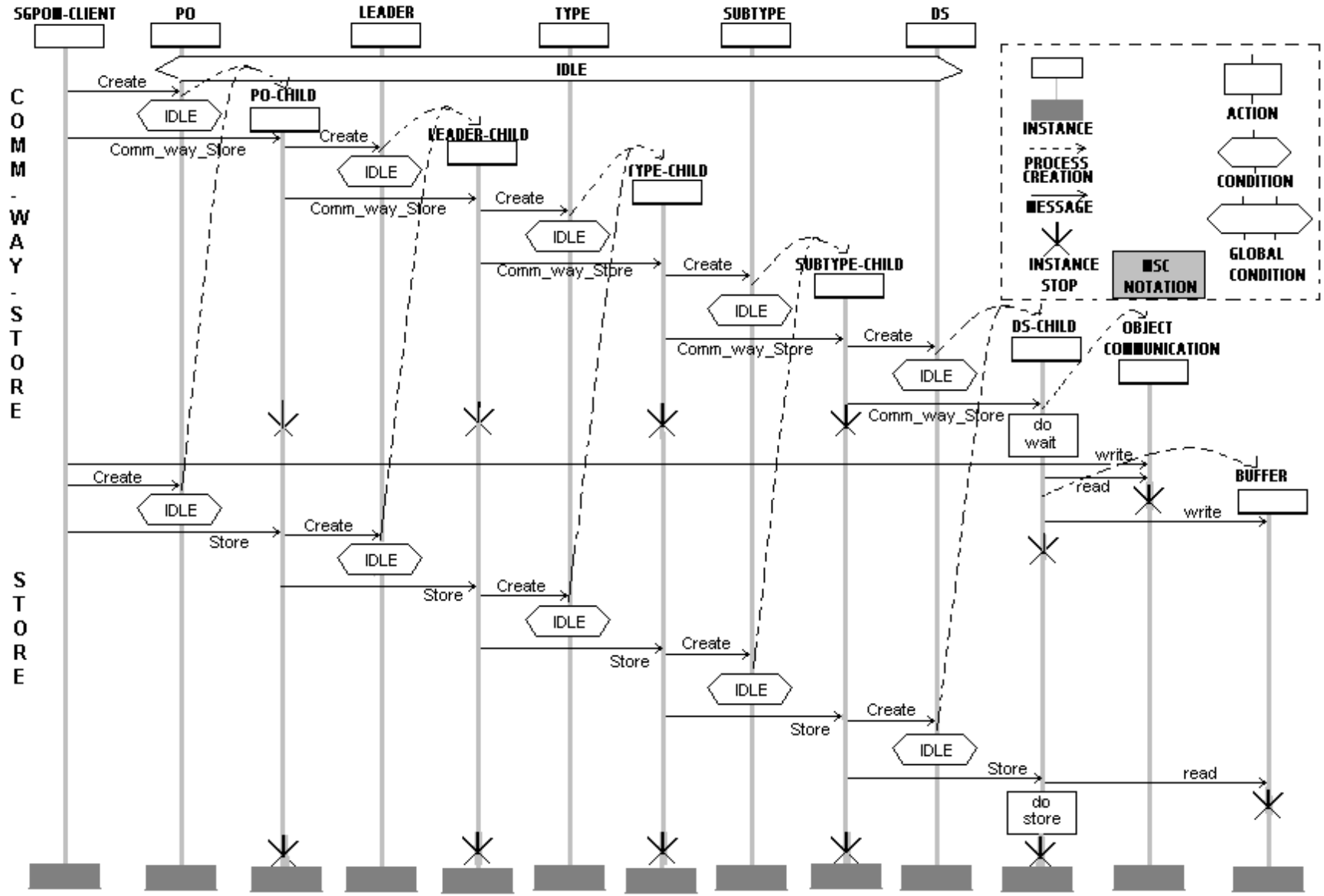

***Figura B.3**: Diagrama MSC para a propagação das operações de armazenagem entre as camadas do SGPOM.*

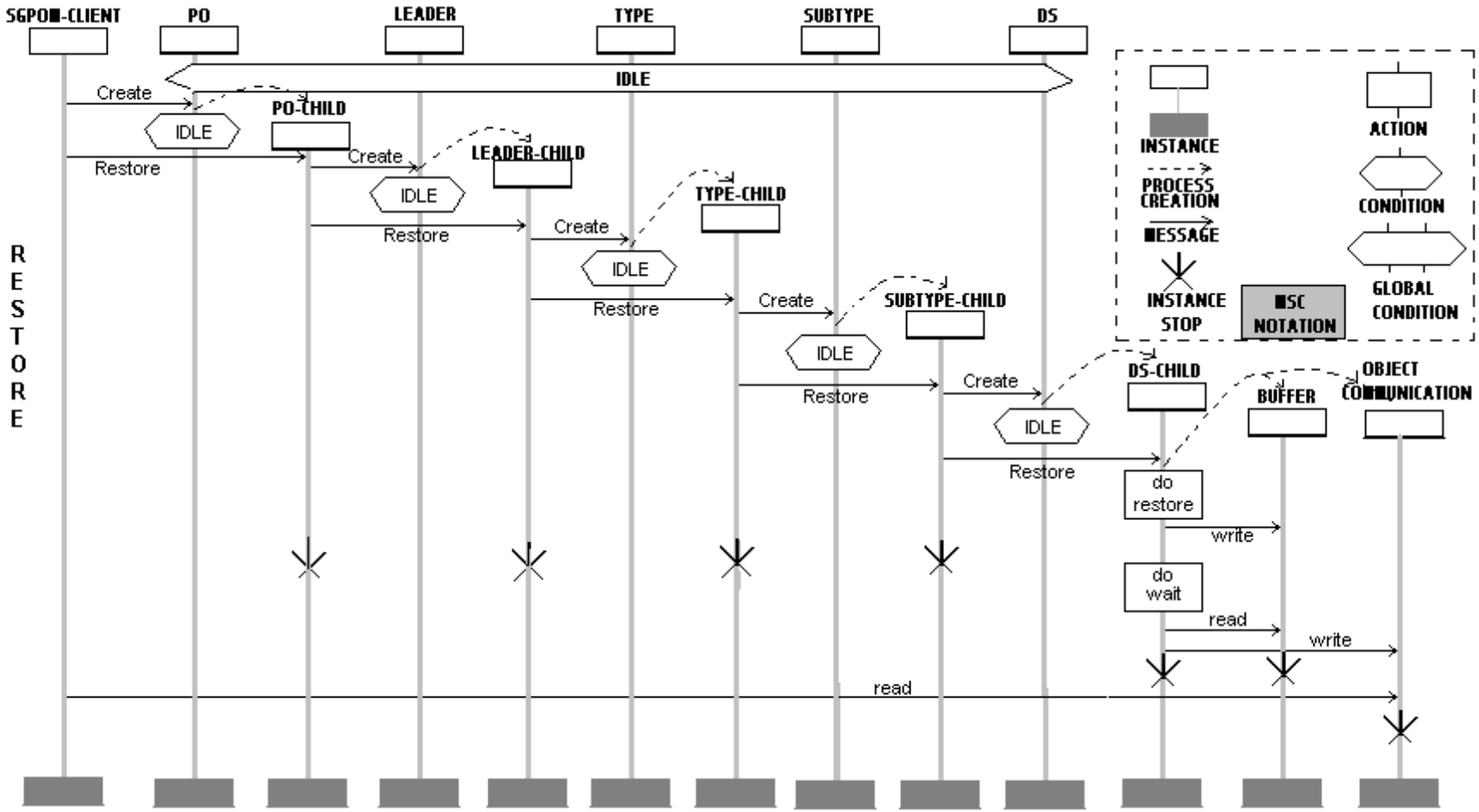

***Figura B.4**: Diagrama MSC para a propagação das operações de recuperação entre as camadas do SGPOM.*

Referências Bibliográficas

[**Adje97**] Adjeroh, D. A. e Nwosu, K. C. - *Multimedia Database Management (Requirements and Issues)*, IEEE Multimedia, 4(3), pp. 24-33, Julho-Setembro 1997.

[**Arau96**] Araújo, D. E. - *Serviços de Gerenciamento ODP utilizando a Arquitetura CORBA*, DCA-FEEC-UNICAMP, Dissertação de Mestrado, Setembro 1996.

[**Atki83**] Atkinson, M. P.; Bailey, P. J.; Chisholm,K. J. et al. - *An Approach to Persistent Programming*, The Computer Journal, 26(4), pp 360-365, Novembro 1983.

[**Atki96**] Atkinson, M. P.; Daynès, L.; Jordan, M.J.; Printezis, T. and Spence, S. - *An Orthogonally Persistent Java*, SIGMOD Record, 25(4), Dezembro 1996.

[**Bake97**] Baker, S.; Cahill, V. e Nixon, P. - *Bridging Boundaries: CORBA in Perspective*, IEEE Internet Computing, pp. 52-57, Setembro-Outubro 1997.

[**Blak96**] Blakowski, G. e Steinmetz, R. - *A Multimedia Synchronization Survey: Reference Model, Specification, and Case Studies*, IEEE Journal on Selected Areas in Communications, Fevereiro 1996.

[**Boll96a**] Boll, S. and Wäsch, Jürgen - *A Java Application Programming Interface to a Multimedia Enhanced Object-Oriented DBMS* - Proc. of the First International Workshop on Persistence and Java, Drymen, Escócia, Setembro 1996.

[**Boll96b**] Boll, S.; Klas, W. e Lörh, M - *Integrated Database Services for Multimedia Presentations*, em [Chun96].

[**Bore93**] Borenstein, N. S. - *MIME: A Portable and Robust Multimedia Format for Internet Mail*, ACM Multimedia Systems Journal, 1(1), pp. 29-36, 1993.

[**Brow88**] Brown A. L. - *Persistent Object Stores*, Ph.D. Dissertation, Department of Computational Sciences, University of St. Andrews, Outubro 1988.

[**Bufo94**] Buford, J.F.K. - *Multimedia Systems*, Addison-Wesley, 1994.

[**Card98**] Cardozo, E.; Prado, R. C. M.; Guedes, L. A. e Faina, L. F. - *ODP Channel: An Open Mechanism for Supporting Media Flows in Distributed Environments*, Proc. XXIV CLEI'98, pp. 111-122, Quito, Equador, Outubro 1998.

[**Chan96**] Channon, D. - *Persistence for C++*, Dr. Dobb's Journal, pp. 46-52, Outubro 1996.

[**Chri96**] Christodoulakis, S. e Koveos L. - *Multimedia Information Systems: Issues and Approaches*, em [Kim95].

[**Chun96**] Chung, S. M. - *Multimedia Information Storage and Management*, Kluwer Academic Publishers, 1996.

[**Cost98**] Costea, I.; Guenther U. e Nicolescu, R. - *Coping with File Formats on The Internet*, Computer Communications, 20(16), pp. 1437-1447, Janeiro 1998.

[**Coul92**] Coulson, G.; Blair, Davies; N. e Williams, N - *Extensions to ANSA for Multimedia Computing*, Computer Networks and ISDN Systems, 25, pp. 305-323, 1992.

[**Coul94**] Coulouris, G.; Dollimore, J. e Kindberg T. - *Distributed Systems, Concepts and Design,* Addison-Wesley, 2ª. edição, 1994.

[**Coul95**] Coulson, G.; Blair, G. S.; Stefani, J. B. et al. - *Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing*, Computer Networks and ISDN Systems, 27(7), pp. 1231-1246, Julho 1995.

[**Davi93**] Davies, N.; Davy, M.; Blair, G. S. e Mariani, J. A. - *Object Invocation and Management in the Zenith Distributed Multimedia Information System*, Information and Software Technology 35(5), pp. 259-266, Maio 1993.
[**Evan97**] Evans, E. e Rogers, D. - *Using Java Applets and CORBA for Multi-user Distributed Applications*, IEEE Internet Computing, pp. 43-57, Maio-Junho 1997.
[**Ferr96**] Ferraz, C. A. - *Distributed Object-based Continuous Media Applications*, Proc. of the 14th. SBRC, pp. 433-452, Fortaleza, Maio 1996.
[**Flan97**] Flanagan, D. - *Java in a Nutshell*, 2ª. edição, O'reilly &Associates, 1997.
[**Fluc95**] Fluckiger, F. - *Understanding Networked Multimedia: Applications and Technology*, Prentice Hall, 1995.
[**Gann96**] Gannod, G. C. e Cheng, B. H. C - *The Object-oriented Development of Multimedia Information Systems*, em [Chun96].
[**Gemm95**] Gemmel, D. J.; Vin, H. M.; Kandlur, D. D. et al. - *Multimedia Storage Servers: A Tutorial*, IEEE Computer, 28(5), pp. 40-49, Maio 1995.
[**Gosl96**] Gosling, J; Joy, B e Steele, G. - *The Java Language Specification*, Addison Wesley, 1996.
[**Gunj95**] - Gunji, T. - *Plataforma Multiware: Serviços de Objetos*, DCA-FEEC-UNICAMP, Dissertação de Mestrado, Agosto 1995.
[**Harr93**] Harris, J. e Ruben, I. - *Bento Specification, Revision 1.0d5,* Apple Computer Inc., Julho 1993.
[**Henn98**] Henning, M. - *Binding, Migration, and Scalability in CORBA*, Communications of ACM, 41(10), Outubro 1998.
[**Hong98**] Hong, J. W.; Shin, Y.; Kim M. e Kim, J. - *Design and Implementation of a Distributed Multimedia Collaborative Environment*, Special Issue on Multimedia Collaborative Environments of Cluster Computing: Networks Software Tools, and Applications, Baltzer Science, 1998.
[**IMA95**] Interactive Multimedia Association - *Recommended Practice for Multimedia Data Exchange, IMA 950701.1*, Setembro 1995.
[**IONA97a**] IONA Technologies PLC - *OrbixWeb Programmer's Guide, Release 3.0*, Novembro 1997.
[**IONA97b**] IONA Technologies PLC - *OrbixWeb Programmer's Reference, Release 3.0*, Novembro 1997.
[**ITU-T94a**] ITU-T Rec. X.901 | ISO/IEC 10746-1 - *Basic Reference Model of Open Distributed Processing: Part 1: Overview and Guide to Use*, 1994.
[**ITU-T94b**] ITU-T Rec. X.904 | ISO/IEC 10746-4 - *Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semantics*, 1994.
[**ITU-T95a**] ITU-T Rec. X.902 | ISO/IEC 10746-2 - *Basic Reference Model of Open Distributed Processing: Part 2: Descriptive Model*, 1995.
[**ITU-T95b**] ITU-T Rec. X.903 | ISO/IEC 10746-3 - *Basic Reference Model of Open Distributed Processing: Part 3: Prescriptive Model*, 1995.
[**Janz**] Janzantti Júnior, N. J. - *Gerência de Dados Multimídia em Banco de Dados Orientados a Objetos* (título provisório), Dissertação de Mestrado, em conclusão.
[**Karm96**] Karmouch, A. e Emery, J. - *A Playback Schedule Model for Multimedia Documents*, IEEE Multimedia 3(1), pp. 50-61, Spring 1996.
[**Kim95**] Kim, W. - *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Addison-Wesley, 1995.

[**Klei96a**] Kleindienst, J.; Plášil, F. and Tüma, P. - *What We Are Missing in the CORBA Persistent Object Service Specification*, OOPSLA`96 Workshop on Large Persistent and Distributed Systems, 1996.
[**Klei96b**] Kleindienst, J.; Plášil, F. e Tüma, P. - *Lessons Learned from Implementing the CORBA Persistent Object Service*, ACM SIGPLAN NOTICES 31(10), pp. 150-167, Outubro 1996.
[**Lini95**] Linington, P. F. - *RM-ODP: The Architecture*, Proc. of IFIP ICODP'95, pp. 15-33, Brisbane, Austrália, Fevereiro 1995.
[**Loyo93**] Loyolla, W.; Madeira, E.; Mendes M. J.; Cardozo, E. e Magalhães, M. F. - *Multiware Platform: An Open Distributed Environment for Multimedia Cooperative Applications*. IEEE COMPSAC, Taipei, Taiwan, Novembro 1994.
[**Maff97**] Maffeis, S. - *Piranha: A CORBA Tool for High Availability*, IEEE Computer, pp. 59-66, Abril 1997.
[**Mage97**] Magedanz, T. - *TINA - Architectural Basis for Future Telecommunications Services*, Computer Communications 20(4), pp. 233-245, Junho 1997.
[**Mauw94**] Mauw, S. e Reniess, M. A. - *An Algebric Semantics of Basic Message Sequence Charts*, The Computer Journal, 37(4), pp. 269-277, 1994.
[**Meye96**] Meyer, B. - *Schema Evolution: Concepts, Terminology, and Solutions*, IEEE Computer, Outubro 1996.
[**Mung99**] Mungee, S.; Surendran, N. e Schmidt D. C. - *The Design and Performance of a CORBA Audio/Video Streaming Service*, Proc. of the 32th Hawaii International Conference on Systems and Sciences (HICSS), Havaí, EUA, Janeiro 1999.
[**Nara96**] Narasimhalu, A. D.- *Multimedia Databases*, ACM Multimedia Systems Journal, 4(5), pp. 226-249, Outubro 1996.
[**Nwos94**] Nwosu K. C. e Thuraisingham, B. - *Extending an Object-Oriented Data Model for Representing Multimedia Database Applications: A Position Paper*, Proc. of OOPSLA Workshop on Precise Behavioral Specifications in Object-Oriented Information Modeling, pp. 77-83, Outubro 1994.
[**O'Bri96**] O'Brien, P. - *Java Data Management Using ObjectStore and PSE*, ObjectDesign White Paper, Novembro 1996.
[**Oliv98]** Oliver, H; Edwards, C. e Hutchison, D. - *The Role of Distributed Computing in Telecommunications: Experiences and Analyses*, Proc. of the 6th IEE Conference on Telecommunications, Edinburgo, Reino Unido, 29 March-1 April 1998.
[**OMG94a**] Object Management Group - *Persistent Object Service Specification 1.0*, OMG Document Number 97-12-12, Outubro 1994.
[**OMG94b**] Object Management Group - *Object Externalization Service Specification*, OMG Document Number 97-12-15, Dezembro 1994.
[**OMG94c**] Object Management Group - *Relationship Service Specification*, OMG Document Number 97-12-16, Dezembro 1994.
[**OMG94d**] Object Management Group - *Compound LifeCycle Addendum*, OMG Document Number 94-05-06, 1994.
[**OMG95a**] Object Management Group - *The Common Object Request Broker: Architecture and Specification (Revision 2.0)*, OMG Document Number 96-08-04, Julho 1995.
[**OMG95b**] Object Management Group - *Object Services Architecture*, OMG Document Number 95-01-47, Janeiro 1995.
[**OMG97a**] Object Management Group - *IDL/Java Language Mapping*, OMG Document Number 97-03-01, Março 1997.

[**OMG97b**] Object Management Group -*Persistent State Service 2.0 Request For Proposal*, OMG Document Number 97-06-07, Junho 1997.
[**OMG98a**] Object Management Group - *CORBAservices: Common Object Services Specification*, OMG Document Number 98-07-06, Março 1998.
[**OMG98b**] Object Management Group - *CORBA Facilities Architecture,* OMG Document Number 98-07-10, Julho 1998.
[**OMG98c**] Object Management Group - *Objects by Value Specification*, OMG Document Number 98-01-18, Julho 1998.
[**OMG98d**] Object Management Group - *Mobile Agents Facility*, OMG Document Number 97-10-05, Fevereiro 1998.
[**Orfa97**] Orfali, R. e Harkey, D. - *Client/Server Programming with Java and CORBA*, John Willey & Sons, 1997.
[**Orji96**] Orji, C. U. e Nwosu, K. C. - *Multimedia Object Storage and Retrieval*, Proc. of the International Symposium on Multimedia Systems, pp. 369-375, Japão, Março 1996.
[**Özde95**] Özden B.; Rastogi, R. e Silberschatz A. - *Research Issues in Multimedia Storage Servers*, ACM Computing Surveys, 27(4), Dezembro 1995.
[**Paza97**] Pazandak, P. e Srivastava, J. - *Evaluating Object DBMSs for Multimedia*, IEEE Multimedia, 4(3), pp. 34-49, Julho-Setembro 1997.
[**Pint99**] Pinto, A. de S. - *Projeto e Implementação de um Ambiente para Processamento Distribuído Baseado em TINA* (título provisório), DCA-FEEC-UNICAMP, Dissertação de Mestrado, em conclusão.
[**Pláš98**] Plášil, F.; Tüma, P. e Buble, A. - *CORBA Benchmarking*, Charles University Technical Report, OMG Document Number 98-10-04, 1994.
[Rako95] Rakow, T. C.; Neuhold, E. J. e Lohr, M. - *Multimedia Database Systems — the Notions and the Issues*, Database Systems in Office, Technology and Knowledge, pp. 1-29, Springer, Berlin, Março 1995.
[**Raym95**] Raymond, K. A. - *Reference Model of Open Distributed Processing (RM-ODP): Introduction.* Proc. of IFIP ICODP'95, pp. 3-14, Austrália, Fevereiro 1995.
[**Reve96**] Reverbel, F. - *Persistence in Distributed Object Systems: ORB/ODBMS Integration*, Ph.D. Dissertation, Department of Computer Science, University of New Mexico, Abril 1996.
[**Rica96**] Ricarte, I. L. M. e Tobar, C. M. - *Towards an Architecture for Distributed Multimedia Databases*, Proc. of the International Conference on Intelligent Information Systems, Washington, EUA, Junho 1996.
[**Rumb91**] - Rumbaugh, J., et. al. - *Object-Oriented Modeling and Design*, Prentice Hall, 1991.
[Schm98] Schmidt D. C., Levine D. e Mungee, S. - *The Design and Performance of Real-Time Object Request Brokers*, Computer Communications, 21(4), pp. 294-324, Abril 1998.
[**Shen95**] Shenoy, P., Goyal P. e Vin, H. M. - *Issues in Multimedia Servers Design*, ACM Computing Surveys, 27(4), pp. 636-639, Dezembro 1995.
[**Sess96**] Sessions, R. - *Object Persistence* (*Beyond Object-Oriented Databases)*, Prentice Hall, 1996.
[**Silv97**] da Silva, M. M. - *Mobility and Persistence*, Lecture Notes in Computer Science 1222, (eds. Vitek, J. e Tschudin, C.), pp. 157-175, Springer-Verlag, 1997.
[**Sole95a**] Soley, R. M. e Stone, C. M. - *Object Management Architecture Guide*, 3ª. edição, John Wiley & Sons, 1995
[**Sole95b**] Soley, R. M. e Kent, W. - *The OMG Object Model*, em [Kim95].

[**Spen97**] Spence, S. and Atkinson, M. P. - *A Scalable Model of Distribution Promoting Autonomy of and Cooperation between PJava Object Stores* - Proc. of the 30th HICSS, Havaí, EUA, Janeiro 1997.

[**Sun97a**] Sun Microsystems Inc. - *Java Object Serialization Specification*, Revisão 1.3, 1997.

[**Sun97b**] Sun Microsystems Inc. - *Java Media Players*, Versão 1.0.5, Maio 1998.

[**Tamb97**] Tambascia, C. A. - *Sistema Gerenciador de Acesso a Dados Multimídia*, DCA-FEEC-UNICAMP, Dissertação de Mestrado, Dezembro 1997.

[**Tezu96**] Tezuka, H. e Nakajima, T. - *Simple Continuous Media Storage Server on Real-Time Mach*, Usenix Technical Conference Winter, Janeiro 1996.

[**Toba95**] Tobar, C. e Ricarte, I. - *Multiware Database, a Distributed Object Database System for Multimedia Support.* Open Distributed Processing: Experiences with Distributed Environments, Proc. of the IFIP ICODP'95, pp. 439-450, Fevereiro 1995.

[**Tsch93**] Tschammer, V.; Mendes, M. J,; Souza, W. L.; Madeira, E. R. M. e Loyolla, W. P.: *Processamento Distribuído Aberto e o Modelo RM-ODP/ISO*, XI Simpósio Brasileiro de Redes de Computadores, UNICAMP, Campinas, Maio 1993.

[**Tüma97**] Tüma, P. - *Persistence in CORBA*, Ph.D. Dissertation, Charles University, Department of Software Engineering, 1997.

[**Verh98**] Verhoeven, A. e Warendorf, K. - *Java Hypermedia: On and Beyond The World Wide Web*, Computer Communications, 20(16), pp. 1481-1489, Janeiro 1998.

[**Viei95**] Vieira, M. T. P.; Santos, M. T. P. e Pavão, S. - *Armazenamento e Recuperação de Objetos Multimídia*, I Workshop em Sistemas Hipermídia Distribuídos, São Carlos, SP, Julho 1995.

[**Vino93**] Vinoski, S. - *Distributed Object Computing with CORBA*, C++ Report Magazine, Julho/Agosto 1993.

[**Vino97**] Vinoski, S. - *CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments*, IEEE Communications Magazine, 14(2), Fevereiro 1997.

[**Vino98**] Vinoski, S. - *New Features for CORBA 3.0*, Communications of ACM, 41(10), pp. 44-52, Outubro 1998.

[**Voge97**] Vogel, A. e Duddy, K. - *Java Programming with CORBA*, John Willey & Sons, 1997.

[**Will92**] Williams, N. e Blair, G. - *Distributed Multimedia Application Study*, Technical Report MPG-92-11, Faculty of Applied Sciences, Lancaster University.

[**Wolf97**] Wolfe, V. F., Cingiser L., DiPippo, L. C. et al. - *Real-Time CORBA*, Proc. of the 3rd. IEEE Real-Time Technology and Applications Symposium, Montreal, Canadá, Junho 1997.

[**Wu+95**] Wu, J. K.; Narasimhalu, A. D., Mehtre, B. M. et al. - *CORE: A Content-based Retrieval Engine for Multimedia Information Systems*, ACM Multimedia Systems Journal, 3(1), pp. 25-41, Fevereiro 1995.

[**Yang96**] Yang, Z. e Duddy, K. - *CORBA: A Platform for Distributed Object Computing (A State-of-the-Art Report on OMG/CORBA)*, ACM Operating Systems Review, 30(2), pp. 4-31, Abril 1996.

[**Yun+97**] Yun, T. H.; Kong J. Y. e Hong, J. W. - *A CORBA-based Distributed Multimedia System*, Proc. of 1997 Pacific Workshop on Distributed Multimedia Systems, pp. 1-8, Canadá, Julho 1997.