Modelo de Objetos Baseado no CORBA para Sistemas Hipermídia Abertos com Garantias de Sincronização*

Sérgio Colcher[1†]
colcher@inf.puc-rio.br

Luiz Fernando G. Soares[1]
lfgs@inf.puc-rio.br

Marco Antonio Casanova[2]
casanova@vnet.ibm.com

Guido Lemos S. Filho[1,3]
guido@dimap.ufrn.br

[1]Laboratório Telemídia
Depto. de Informática - PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro - RJ
22453-900, Brasil

[2]CC-Rio IBM Brasil
Caixa Postal 4524
Av. Presidente Vargas, 824/844
Rio de Janeiro - RJ
59072-930, Brasil

[3]DIMAp UFRN
Campus Universitário
Lagoa Nova
Natal - RN
20071-970, Brasil

Resumo

Esse trabalho apresenta algumas extensões ao modelo de objetos do CORBA, com o intuito de permitir o adequado tratamento de requisitos de sistemas multimídia e hipermídia. Alguns conceitos presentes em padrões para sistemas abertos que também se utilizam do paradigma de orientação por objetos, como ODP, ANSA e PREMO, são adaptados para o ambiente proposto, como o conceito de objeto ativo. Baseadas nos aspectos de sincronização colhidos do modelo de apresentação para sistemas hipermídia, as extensões propostas incluem o suporte para a especificação de atributos de tipos contínuos, como áudio e vídeo, e *MediaPipes*, que permitem a escolha das abstrações mais adequadas no tratamento do acesso e transferência de dados contínuos.

Abstract

This paper outlines some extensions to the object model underlying the CORBA specification to address the requirements of multimedia and hypermedia systems. These extensions include the adaptation of many concepts brought from other standards, such as ODP, ANSA and PREMO, like the concept of active objects. Based on the synchronization aspects of hypermedia models, the proposed extensions include support for specification of objects with continuous media attributes, like audio and video, and the *MediaPipes*, which allows for the sellection of suitable abstractions to deal with the access and transfer of continuous media information.

*Este trabalho foi desenvolvido com o apoio do CNPq através do projeto PROTEM-HyperProp.

†Aluno do programa de Doutorado.

1 Introdução

A construção de sistemas abertos depende da disponibilidade e da utilização de normas. Cada norma ou padrão definido por um órgão ou consórcio é, em geral, dedicado a definição de um aspecto específico dentro das necessidades globais de um sistema de informação. Padrões para interconexão entre sistemas, por exemplo, tratam de compatibilizar e promover a integração de diferentes equipamentos em um ambiente distribuído. Um dos mais populares padrões relativos à interconexão é o modelo de referência OSI (RM-OSI). Vários outros padrões, considerando a estruturação definida pelo RM-OSI, existem para definir determinados aspectos específicos, como, por exemplo, as estruturas de dados intercambiáveis entre aplicações. O padrão ISO MHEG [7] é um exemplo de padrão para definição de objetos multimídia/hipermídia destinados ao intercâmbio.

Apesar de essenciais para o desenvolvimento de sistemas abertos, padrões como os anteriormente citados ainda não são suficientes. Em ambientes distribuídos, as várias partes que compõem o sistema devem cooperar para que um comportamento esperado possa ser observado. Cada parte tem um papel determinado dentro do sistema e, para permitir o correto funcionamento e extensibilidade, a semântica envolvida na computação dessas partes, e do sistema como um todo, deve ser especificada. A especificação das unidades de informação intercambiáveis definidas pelo MHEG, por exemplo, em conjunto com os protocolos e serviços para a sua troca, podem não ser suficientes para o correto intercâmbio entre um cliente e um servidor, caso eles não concordem com a semântica envolvida na interpretação das informações — como modelos de composição, mecanismos de tratamento de versões, etc.

A maioria dos padrões são primordialmente voltados aos aspectos tecnológicos e às interfaces para serviços em ambientes de suporte. Um padrão para a especificação da semântica de aplicações é também necessário, como exemplificado, para permitir a construção adequada de sistemas abertos. O padrão *ISO ODP (Open Distributed Processing)* [5], dentro desse enfoque, fornece os mecanismos necessários para a especificação de cinco diferentes *pontos de vista (viewpoints)* de um sistema, cobrindo tanto os aspectos tecnológicos da plataforma, como também os aspectos computacionais e de organização de informação das aplicações. Em sistemas hipermídia, em particular, os pontos de vista de informação e computacional correspondem exatamente à descrição dos modelos *estrutural* e de *apresentação*, respectivamente.

O modelo estrutural para sistemas hipermídia adotado no projeto HyperProp, onde se insere este trabalho, é obtido da definição do *Modelo de Contextos Aninhados (Nested Context Model — NCM)* [1, 11], em conformidade com o modelo do padrão MHEG. O NCM contém um modelo que generaliza e estende a estruturação existente na maior parte dos sistemas hipermídia. Resumidamente, o modelo é baseado nos conceitos usuais de nós e elos. *Nós* são fragmentos de informação e *elos* interconectam nós na formação de redes de inter-relacionamentos de informação. O modelo permite a agregação de nós em composições mais complexas, através da distinção de uma classe especial de nó denominada nó de contexto. *Nós de contexto* (Composites, na terminologia do MHEG) agrupam outros nós (terminais ou de contexto, recursivamente) e elos. *Nós terminais* (Contents, na terminologia do MEHG) contém, tipicamente, fragmentos de informação de uma determinada mídia. Elos permitem a especificação de relacionamentos, que podem ser utilizados tanto para a navegação com interação do usuário, quanto para a especificação de sincronização temporal e espacial.

Um modelo estrutural trata documentos hipermídia como estruturas essencialmente passivas. Um sistema hipermídia deve, porém, fornecer ferramentas para acesso, manipulação e navegação na rede de informações. Essa funcionalidade é capturada pelo *modelo de apresentação*. A dinâmica da apresentação de documentos hipermídia impõe restrições severas de sincronização e qualidade de serviço (Quality of Service — QoS) que são frutos, basicamente, da presença de *tipos de dados contínuos*, como áudio e vídeo. O Modelo de Contextos Aninhados descreve, além do modelo estrutural, um modelo de apresentação, conforme descrito na referência [8].

Assim como na proposta do ODP para os pontos de vista de informação e computacional, os modelos estrutural e de apresentação do NCM se utilizam do paradigma de orientação por objetos. A realização de um sistema que corresponda a essas especificações é, portanto, bastante facilitada pela disponibilidade de uma plataforma de desenvolvimento orientada a objetos que forneça suporte a tipos de dados contínuos.

Esse trabalho apresenta uma proposta de realização de tal plataforma, isto é, de uma plataforma de desenvolvimento orientada a objetos com suporte para tipos de *dados contínuos*, baseada na definição de um *Object Request Broker (ORB)* [9], nos moldes da definição do CORBA, estendido de forma a acomodar o conceito de MediaPipes. *MediaPipes* são abstrações que representam conexões explícitas entre objetos com atributos de tipo contínuo, que podem ser criados, associados a esses objetos e controlados, através de sua interface, de forma a ajustar características de QoS desejadas. Apesar da implementação de um ORB como esse demandar grandes esforços, tanto do sistema operacional como do suporte de comunicação, este artigo tem como principal objetivo apenas a definição das abstrações de programação adequadas para a construção de uma plataforma orientada a objetos para sistemas hipermídia abertos. A descrição detalhada dos aspectos de implementação e arquitetura é tema de um outro trabalho em andamento.

O artigo está estruturado da seguinte forma. Inicialmente, na Seção 2, apresentam-se alguns conceitos básicos, os propósitos e o escopo de alguns padrões relacionados a definição de sistemas abertos com conceitos de orientação por objetos, como o ODP, o ANSA e o próprio CORBA. Ainda nessa seção, comenta-se um padrão, também apoiado pela definição de um modelo de objetos, relacionado à especificação de um ambiente de apresentação multimídia (ISO PREMO), que inclui algumas características para tratamento de sincronização. A Seção 3 apresenta os principais conceitos envolvidos na dinâmica de apresentação de documentos hipermídia, destacando os aspectos de sincronização e a necessidade de objetos de dados e objetos de representação. Na Seção 4 concentram-se as principais contribuições do trabalho, que incluem: a descrição do ambiente de desenvolvimento baseado no CORBA, o tratamento de herança nesse ambiente, a definiçao de MediaPipes e tipos de dados contínuos que, em conjunto permitem a recuperação e apresentação adequada de objetos multimídia/hipermídia, as orientações de projeto para a construção de objetos de dados e objetos de representação e, por fim, os principais tópicos envolvidos no suporte a objetos ativos, que permitem flexibilidade, concorrência e o tratamento de tempo real necessários em sistemas hipermídia. A Seção 5 é reservada às conclusões.

2 Modelos de Objetos e o Desenvolvimento de Sistemas Abertos

Vários padrões que tratam da definição de modelos de objetos para a construção de sistemas hipermídia têm sido propostos. Essa seção apresenta uma breve revisão de alguns dos mais importantes trabalhos nessa área e seu escopo. O CORBA, em particular, será tratado com especial interesse, já que será tomado como base para a definição do modelo proposto neste trabalho.

2.1 ODP e ANSA

O RM-ODP divide a especificação de uma sistema em cinco pontos de vista (*viewpoints*), juntamente com um conjunto de conceitos genéricos que expressam cada um deles. Para cada ponto vista, uma linguagem própria pode ser definida para a utilização em outros padrões ou em implementações que especifiquem sistemas particulares, com seus requisitos próprios. A especificação de cada ponto de vista corresponde a uma abstração do sistema como um todo, porém tomada de apenas um determinado ângulo. Os cincos pontos de vista são:

- o *enterprise viewpoint*, que lida com os aspectos relacionados aos objetivos e funções do sistema no seu ambiente e as relações com o mundo externo. Trata do sistema e do meio em que ele está inserido, focalizando o papel desse sistema nesse meio;
- o *information viewpoint*, que se concentra na definição das informações utilizadas, processadas ou trocadas pelo sistema; não trata, porém, da forma de concatenação ou articulação do sistema no tratamento dessa informação;
- o *computational viewpoint*, que descreve o sistema como um conjunto de objetos atuantes na formação do sistema, sem entrar em detalhes de localização ou distribuição de objetos;
- o *engineering viewpoint*, cuja principal foco é o mecanismo de distribuição dos objetos;
- o *technology viewpoint*, que cuida dos detalhes relativos a definição do hardware e software que compõem os nós de processamento do ambiente distribuído.

Um trabalho que influenciou fortemente a definição do ODP foi a definição da *ANSA (Advanced Network Systems Architecture)* [13]. Os conceitos apresentados no ODP são, na sua quase totalidade, originados na ANSA. Diferenças existem, embora não cheguem a ser significativas. Alguns trabalhos no sentido de incorporar o tratamento de tipos de dados contínuos, qualidade de serviço e sincronização no modelo de objetos do ANSA foram realizados por um grupo da universidade de Lancaster, como o trabalho descrito por Coulson et all [2]. Alguns conceitos apresentados no presente artigo são adaptações das idéias, lá apresentadas, para o ambiente do CORBA. As principais diferenças são encontradas nas abstrações para programação onde, na abordagem aqui apresentada, opta-se por permitir mais de um tipo de interface para a troca de informações de tipo contínuo, cabendo ao MediaPipe sua compatibilização, enquanto que Coulson define uma interface única, baseada no conceito de stream. Por não se dispor ainda de subsídios suficientes para afirmar que as abstrações de stream são adequadas em todos os casos, permitiu-se a caracterização da interface para acesso a dados contínuos através de alguns parâmetros qualificadores, como se verá ao longo do trabalho.

2.2 CORBA

A principal idéia do CORBA (Common Object Request Broker Architecture) é definir um conjunto de interfaces e mecanismos que permitam a interação entre objetos de forma transparente à sua distribuição e à linguagem de programação utilizada na sua codificação. Os objetivos são, portanto, a total interoperabilidade e portabilidade dos objetos, alinhando-se, assim, aos esforços de construção de sistemas abertos.

A transparência quanto a localização dos objetos é obtida através da atuação de um elemento denominado *ORB (Object Request Broker)*, que se encarrega de enviar as solicitações e devolver as eventuais respostas aos objetos (veja Figura 1). A portabilidade decorre diretamente da definição das interfaces dos clientes e implementações de objetos com o ORB.

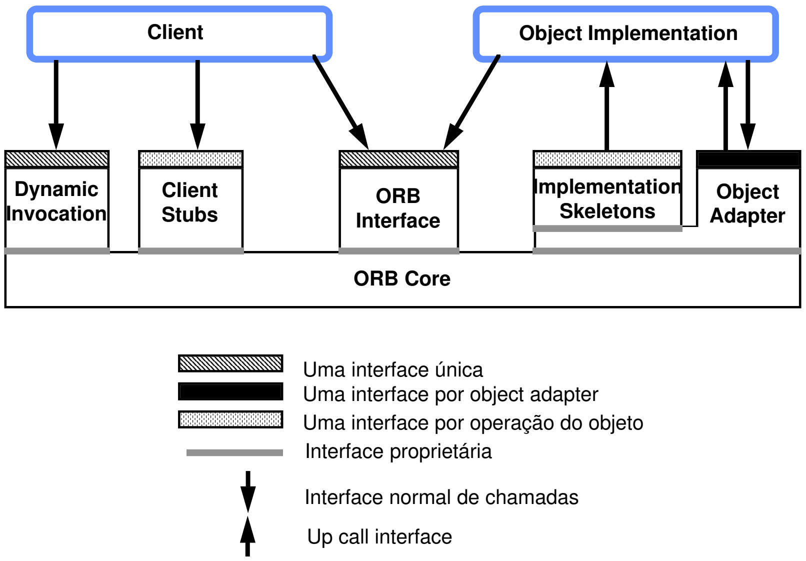

Figura 1: CORBA

A descrição da interface de uma classe de objetos é, tipicamente, fornecida pelo projetista ou implementador daquela classe, e é especificada segundo uma linguagem padronizada pelo CORBA, denominada *Interface Definition Language (IDL)*. A IDL permite a especificação da interface de forma abstrata e independente de linguagem de programação.

A partir da definição, em IDL, de uma interface, stubs e skeletons podem ser gerados para o cliente e para a implementação, de forma a tornar transparentes os mecanismos de distribuição dos objetos. Stubs e skeletons serão intermediários das solicitações que, respectivamente, ativarão e serão ativadas pelo núcleo do ORB. As interfaces dos stubs e skeletons são únicas para cada classe e a sua geração realizada por um compilador de IDL, que poderá gerar interfaces específicas para utilização em diferentes linguagens de programação, de acordo com o desejo do programador.

A partir de uma descrição em IDL, é possível também gerar uma representação da interface que pode ser consultada, em tempo de execução. Essa representação fica armazenada num *repositório de interfaces (Interface Repository — IR)*, podendo ser utilizada por clientes que, por ventura, não tenham tido acesso à IDL da classe que desejam utilizar (e, portanto, não têm os stubs correspondentes). Nesse caso, esses clientes podem se utilizar da DII (Dinamic Invocation Interface) para obter os serviços necessários. A DII permite a consulta ao repositório de interfaces em tempo de execução, o que habilita o cliente à construção de solicitações corretas (de acordo com a interface oferecida pela implementação da classe do objeto de destino) e o envio das mesmas ao objeto de destino.

Para ativar corretamente as implementações de objetos, o ORB necessita de informações sobre a localização do código das classes e suas características de ativação. Essas informações são fornecidas numa fase de instalação da implementação, depois da qual a classe fica disponível para utilização no ambiente. A Figura 2 ilustra a geração dos diversos componentes citados e suas relações com os clientes e implementações.

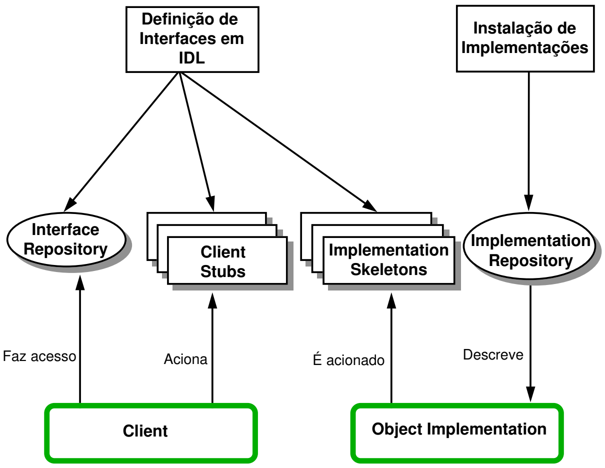

Figura 2: Geração de stubs, skeletons, repositório de interfaces e repositório de implementações.

O *Object Adapter (OA)* é o principal responsável por permitir a ativação das implementações pelo ORB. O OA se encarrega de ativar as implementações, fazendo os devidos acessos ao repositório de implementações, gerar identificadores únicos para objetos e disparar operações através dos seus esqueletos (skeletons). Em, particular, diversas políticas de ativação de implementações são definidas pelo CORBA, como se verá na Seção 4.5.

A definição das interfaces dos objetos que compõem o ORB, oferecendo diretamente serviços a implementações de objetos ou a clientes, é também feita em IDL (e especificada no CORBA), porém, por não se tratarem de objetos que utilizam o ORB como suporte, tais objetos são chamados *pseudo-objetos*. Em outras palavras, pseudo-objetos corresponderiam, no RM-ODP, a objetos do engineering viewpoint (engineering objects) que, através de seus serviços, oferecem os mecanismos de troca mensagens e transparência da distribuição. MediaPipes são pseudo-objetos propostos no presente trabalho.

2.3 PREMO

O PREMO (Presentation Environment for Multimedia Objects) [6] é um padrão da ISO que vem sendo desenvolvido para especificar um ambiente de programação que permita a construção de aplicações sobre um conjunto de abstrações relacionadas a apresentação de objetos multimídia. O PREMO se utiliza do modelo de objetos definido pelo CORBA, modificando alguns conceitos e adicionando outros específicos para ambiente em questão. Quanto a semântica da solicitação de operações, o PREMO apresenta algumas diferenças em relação ao CORBA, conforme descrito a seguir.

O PREMO apresenta três opções para o modo com que operações podem ser definidas:

- *síncrono*: a solicitação é colocada numa fila para atendimento do servidor e o cliente permanece suspenso até que o servidor retorne de seu serviço. A solicitação pode envolver parâmetros e ter um valor de retorno. O CORBA define uma semântica de operação similar através do conceito de *at-most-once*.[1]

- *assíncrono*: a solicitação é colocada numa fila para atendimento do servidor e o cliente é imediatamente liberado, sem permanecer esperando retorno. A solicitação pode envolver parâmetros, mas valores de retorno não são permitidos. No CORBA, uma operação com essa semantica é denominada *best-effort*.

- *amostrado*: idêntico ao modo assíncrono, mas a fila de atendimento tem tamanho para apenas uma requisição. Na presença de várias requisições sucessivas, a mais nova se sobrepõem à mais antiga assumindo seu lugar na fila. Não é prevista pelo CORBA. A referência [4] ilustra a utilização de operações com esse tipo de semântica em ambientes de apresentação de documentos multimídia.

O PREMO define ainda a utilização de *objetos ativos*, o que corresponde a existência de um fluxo de execução independente para cada objeto do sistema, e uma infra-estrutura de objetos básicos para tratamento de eventos. Em sistemas multimídia/hipermídia, como se verá na Seção 3, a concorrência que decorre da utilização de objetos ativos é fundamental para que os vários objetos possam ser apresentados independentemente, ao mesmo tempo em que eventos são registrados e sinalizados e, ainda, elos disparados, para permitir a sincronização das várias apresentações. O CORBA não faz menção explícita a objetos ativos, mas eles podem ser facilmente implementados através de uma das políticas de ativação previstas para o OA, conforme proposição apresentada na Seção 4.5.

3 Apresentação de Documentos Multimídia/Hipermídia

O trabalho apresentado em [8] descreve alguns aspectos do modelo de apresentação do NCM, que possui três características básicas. Primeiro, ele torna possível, aos autores, especificar restrições de sicronização entre eventos que ocorrem durante a apresentação de documentos hipermídia, utilizando os conceitos de âncora e elo. Segundo, ele permite um

[1] Operações com semântica at-most-once são operações que, se retornam normalmente, o cliente, ao sair do estado suspenso, tem a certeza de que a operação foi executada exatamente uma vez. Caso o retorno não seja normal (um erro de comunicação ou qualquer outra ocorrência de excessão), a operação pode ter sido executada uma ou nenhuma vez (i.e., no máximo uma vez).

tratamento homogêneo de elementos assíncronos, como interações de seleção de âncoras pelo usuário, e elementos síncronos, como a especificação do disparo da exibição de um vídeo em um instante específico. Por último, o modelo consegue manter a separação entre a estrutura original do documento e as especificações que definem sua apresentação. As facilidades de sincronização descritas pelo modelo de apresentação do NCM são genéricas o suficiente para serem aplicadas a qualquer modelo cuja visão estrutural englobe o conceito de nós de composição e, com pequenas modificações, até mesmo naqueles sem esse recurso. Em particular, os mecanismos adequam-se bem aos modelos estruturais que possam ser diretamente mapeados no padrão MHEG.

3.1 Sincronização

No NCM, a apresentação de um documento hipermídia é composto da apresentação de suas unidades de informação. Uma unidade de informação em um nó de contexto equivale a um de seus nós componentes. Uma unidade de informação em um nó terminal é uma região do seu conteúdo, onde "região" pode variar dependendo do tipo do nó (um conjunto de frames num vídeo, ou um conjunto de amostras num áudio, por exemplo). A exibição de uma unidade de informação define um *evento de exibição*. A seleção, através de uma interação com o usuário, de uma unidade de informação (como a seleção de uma âncora para navegação, por exemplo) define um *evento de interação*. Outros eventos podem ser definidos, correspondendo a mudanças no estado de um objeto.

Relacionamentos entre eventos são capturados por elos do modelo. Em tempo de execução, elos são interpretados como uma asserção lógica sobre um conjunto de eventos. A definição de elos no NCM é compatível com a definição dos elos do MHEG, contendo um conjunto de *condições de disparo* (baseadas em eventos) e um conjunto de *ações* associadas. O modelo estabelece que, sempre que as condições de disparo de um elo qualquer forem satisfeitas, as ações correspondentes devam ser imediatamente executadas.

3.2 Objetos de Dados e Objetos de Representação

Concentrando a atenção sobre os nós terminais, em especial sobre aqueles cujo conteúdo é de um tipo contínuo, percebe-se um aspecto que pode levar a diferentes abstrações em termos de interface e programação: a noção de um *recurso associado* ao nó ou *recurso hospedeiro*. Fragmentos de áudio ou vídeo não existem livremente, mas sim armazenados em arquivos, ou sendo capturados por uma câmera (ou microfone), ou temporariamente depositados em um buffer, etc. Algumas características inerentes ao recurso associado podem influenciar na interface e comportamento de um nó. Para tornar esse ponto mais claro, considere, por exemplo, uma operação *seek(position)* sobre um nó de aúdio armazenado em um arquivo, que pode ser definida e utilizada para reposicionar a recuperação a partir de algum ponto indicado pelo parâmetro *position*. A mesma operação de seek não faria sentido se aplicada sobre um nó de áudio associado a um microfone. Logo, apesar de se tratarem, em ambos os casos, de nós de áudio, as interfaces devem ser distintas, dependendo do recurso associado. Vários exemplos poderiam ainda ser citados para reafirmar a influência do recurso hospedeiro sobre a interface de objetos com atributos contínuos. Em particular, é importante concentrar o interesse em três aspectos: o *sentido do fluxo de informação*, a *segmentação do fluxo de informação* e a *iniciativa*, como se verá mais adiante, na Seção 4.3.

A noção de *Objeto de Dados — Data Objects (DOs)*, definida em [10], corresponde a uma representação interna de objetos MHEG, estendidos para acomodar a interface para operações de recuperação, armazenamento e outros mecanismos relacionados a geração e manutenção de informações, de forma independente do ambiente de apresentação. No caso de nós NCM, objetos de dados incorporam a noção do recurso hospedeiro. A definição de atributos de tipo contínuo deve levar em consideração as características do recurso hospedeiro, e os mecanismos para essa definição também serão apresentados na Seção 4.3.

A apresentação da informação hipermídia requer a associação de um nó, que representa o repositório do conteúdo a ser apresentado, a uma série de atributos e operações que são específicos do ambiente de apresentação. Um nó de vídeo, por exemplo, deve ser exibido em uma janela específica adequadamente posicionada em algum dispositivo visual; adicionalmente, uma aplicação ou ferramenta específica de apresentação de vídeo deve ser selecionada para tratar dos aspectos específicos da colocação das imagens na tela, no local apropriado. A associação de um conjunto de informações relativas ao ambiente da apresentação a um DO dá origem a um novo objeto, denominado *Objeto de Representação (Representation Object — RO)* [10, 8]. Uma descrição mais completa de como essa associação toma forma será feita na Seção 4.4.

A informação contida originalmente nos DOs é, essencialmente, independente do ambiente de sua apresentação. Essa informação inclui o conteúdo propriamente dito, que no caso de nós terminais corresponde ao áudio ou vídeo em algum formato (MPEG, etc.), juntamente com atributos que permitem uma descrição ou caracterização do nó, como seu dono, direitos de acesso, data da criação e, genericamente, todos os atributos definidos para a classe de *components* do MHEG. Do ponto de vista estático definido pelo modelo estrutural, não há significado algum associado a essas informações. Durante o processo de apresentação, quando então os ROs são criados a partir dos DOs, essa informação ganha seu significado próprio.

4 Modelo de Objetos

Essa seção apresenta as contribuições relativas às extensões ao modelo de objetos do CORBA, para permitir o tratamento adequado dos requisitos de sistemas multimídia e hipermídia. Inicialmente, descreve-se o funcionamento do ambiente para desenvolvimento e especificação dos objetos que utilizarão o suporte do ORB. Esses objetos poderão fazer uso dos mecanismos de definição de tipos contínuos, cujo suporte é, primordialmente, obtido através da criação de MediaPipes, que representam conexões explícitas entre esses objetos. Depois de tratados os aspectos de continuidade, propõe-se uma solução para o problema de herança em ambientes distribuídos, permitindo ao projetista configurar a geração de stubs e skeletons de acordo com suas necessidades e compromissos de desempenho. A partir da solução do problema da herança, mostra-se a forma com que os objetos básicos da plataforma (ROs e DOs) podem ser projetados, relacionando-os com a arquitetura apresentada em [10]. Levando-se em conta os requisitos de tempo real e concorrência inerentes ao tratamento da continuidade e sincronização de objetos multimídia e hipermídia, aponta-se, ao final da seção, a necessidade do uso de objetos ativos e a forma com que eles são tratados na plataforma proposta.

4.1 Ambiente de Desenvolvimento

Um ambiente baseado no CORBA necessita, além de todo o aparato de tempo de execução, de um compilador que gere stubs e skeletons, utilizados em alguma linguagem de programação, a partir da descrição em IDL da interface de uma classe. No ambiente do HyperProp, surgem aspectos relacionados a desempenho que não estão diretamente refletidos na interface de um classe. A forma do tratamento de herança em ambientes distribuídos é uma delas, como será visto na Seção 4.2. Surge, assim, a necessidade de se permitir a especificação da *configuração da implementação*, que consiste numa especificação abstrata das características da implementação, a ser utilizada para a geração dos esqueletos para o implementador e também fornecer informações importantes sobre a ativação dos objetos a serem registrados no repositório de implementações. Com esse propósito, definiu-se uma linguagem, denominada *ICL (Implementation Configuration Language)*.

A ICL permitirá que o implementador especifique aspectos da implementação como: quais operações, se for o caso, deverão ser enviadas a superclasse (para resolver o problema da herança, conforme descreve-se na Seção 4.2) e qual a política de ativação utilizada e seus parâmetros (número de threads, etc., como veremos na Seção 4.5).

O ambiente proposto sugere que o processo de desenvolvimento inclua os seguintes procedimentos. Inicialmente, o implementador de uma classe produz uma descrição em IDL (estendida conforme os mecanismos a serem apresentados na Seção 4.3) da interface da classe. Da compilação dessa interface, é tudo de que dependem os clientes para ter acesso aos serviços oferecidos. O compilador de IDL é denominado, neste projeto, de `clic` (*client compiler*). O implementador deve, então, fazer uma descrição da configuração da implementação em ICL e passar por outro processo de compilação, que, agora, gerará os esqueletos em uma determinada linguagem de programação e outras informações para utilização no momento da instalação. O compilador de ICL, no âmbito deste projeto, é denominado `serc` (*server compiler*). ICL é, na verdade, mais uma extensão sobre a IDL, que insere diretivas próprias para a configuração da implementação. Tipicamente, uma descrição em ICL começa com a inclusão de uma descrição em IDL (através de um `include`, por exemplo) que permite o aproveitamento de toda a definição da interface. Clientes não deverão ter acesso à ICL, ficando restritos aos aspectos diretamente refletidos na interface.

De posse dos esqueletos, o implementador poderá codificar as operações da classe propriamente dita, utilizando a linguagem de programação escolhida. Por fim, a instalação pode ser efetuada, registrando, junto ao ORB, as informações necessárias para a ativação da implementação, como a sua localização.

4.2 Herança em Ambientes Distribuídos

Herança é geralmente considerada como um dos aspectos mais importantes de orientação a objetos. Ao se falar de herança, pode-se considerar duas possibilidades:

- *herança de interface*, onde a relação de herança se dá exclusivamente entre tipos. A interface do supertipo é herdada pelo subtipo, mas a implementação de cada operação pode ser definida de forma totalmente independente nas classes de tipo e supertipo.

- *herança de implementação*, na qual a relação de herança permite que a implementação de um supertipo seja automaticamente aproveitada para os subtipos.

Para facilitar o entedimento da exposição que se segue, utiliza-se o termo *tipo* para designar a definição de uma interface comum a um grupo de objetos ou valores, e *classe* para designar uma implementação de um tipo de objeto.

O CORBA permite a especificação de tipos e subtipos em IDL sem fazer qualquer menção sobre o aspecto da implementação, permitindo total liberdade àqueles que implementam o ORB. Já linguagens como C++ fazem uso de herança da implementação, que permite grande facilidade na definição de subclasses. Note que, um determinado ambiente baseado no CORBA pode optar por mapear a definição da herança de IDL em classes e subclasses de C++ (caso o compilador de IDL possa gerar stubs e skeletons para C++). Dessa forma, tal implementação utilizará também herança de implementação. Essa atitude, porém, apresenta algumas desvantagens.

Com o mapeamento da criação de subtipos diretamente sobre mecanismos de herança de implementação, é necessário que o código da implementação do supertipo esteja, de alguma forma, ligado ao código do subtipo. Sendo assim, não é possível a definição de subtipos a partir de tipos cuja o código que não esteja disponível para a máquina onde o código do subtipo está localizado e deverá ser ativado.

A alternativa para o mapeamento da herança de interfaces é não usar herança de implementação. Deixa-se a cargo do próprio implementador, se ele assim o desejar, a criação, interna ao código do subtipo, de uma instância do supertipo e da chamada explicita, dentro da implementação de cada operação, à operação correspondente no supertipo. Dessa forma, é possível emular a herança de implementação evitando os problemas mencionados. Paga-se, porém, o preço da queda de desempenho como descrito a seguir. Utilizar chamadas explícitas às operações do supertipo significa transformar os mecanismos de ligação direta dos códigos de tipo e subtipos (utilizados em C++, por exemplo)[2] em mensagens entre objetos. Caso a hierarquia de objetos de uma determinada aplicação seja muito grande, a solicitação de uma operação à um objeto pode acarretar no envio de diversas mensagens, sucessivamente, até que finalmente atinja-se a classe que realmente implementa aquela operação. O ideal seria que fosse possível descobrir diretamente a qual superclasse recorrer (caso a classe do próprio objeto não atenda) para atender a uma determinada solicitação.

Para permitir a localização direta do tipo cuja implementação atende a solicitação de uma operação sem passá-la à superclasse, é necessário que essa informação esteja no repositório de implementações. Para tanto, a ICL deve ser capaz de expressar o desejo do implementador de uma classe em: ou simplesmente aproveitar o código de uma operação da superclasse ou, alternativamente, redefini-lo. Dessa forma, além de permitir a localização mencionada, a ICL conterá informação valiosa também para a geração dos skeletons, pois o `serc` poderá gerar automaticamente todo o código para a criação da instância da superclasse, para a consulta em tempo de execução ao repositório de implementações, a fim de obter a localização mencionada, e para as chamadas às operações necessárias. Como se verá mais adiante, com a introdução dos MediaPipes, o ônus da consulta prévia ao repositório será mais do que compensatório, pois será feito no momento da conexão (anterior à transferência propriamente dita). Após o estabelecimento

[2]Subentende-se, a partir deste ponto, que o modelo de herança de implementação refere-se ao esquema semelhante ao existente em C++, no qual o código de classe e subclasse devem ser de alguma forma associados, seja na compilação, ligação estática ou ligação dinâmica.

do MediaPipe, uma conexão direta existirá entre o cliente e a implementação do objeto que realmente atenderá a solicitação.

Apesar das desvantagens indicadas no mapeamento direto da herança de tipos (descrita em IDL) em mecanismos de herança de implementação como os de C++, em determinadas situações pode ser interessante permitir tal mapeamento, possibilitando um maior desempenho. Para tanto, o `serc` é provido de parâmetros que permitem ao implementador especificar o tipo de compilação que ele deseja para uma determinada configuração de implementação (descrita em ICL). Note que é de total responsabilidade do implementador, ao utilizar a opção que determina a geração de código para utilização da herança de implementação, assegurar que o código de classe e subclasses estão acessíveis no momento da ligação.

4.3 Tipos Contínuos

No CORBA, a definição de um `attribute` em IDL corresponde a definição de duas operações de acesso (tipicamente *set* e *get*) a uma informação de estado de um objeto. Tais operações, aplicadas sobre objetos com tipos convencionais (não contínuos), são, em geral, tratadas por operações síncronas que lidam sempre com um bloco de informações de um determinado tamanho. O tempo de execução dessas operações, durante o qual clientes permanecem suspensos (operações síncronas), é, em geral, indeterminado. Já para o tratamento de tipos contínuos, desejamos ter suporte de operações assíncronas, pois a duração da transferência da informação pode ser longa, o que torna indesejável a permanência do cliente em estado suspenso. Mais ainda, objetos que trocam informações com as características de fluxo menciondas, devem ter a possibildade de exercer o controle sobre esse fluxo sem, no entanto, a necessidade de tratarem sempre com pequenos blocos de informação. Considerando esses requisitos, propõe-se aqui um mecanismo análogo ao da definição de atributos comuns, mas que resultará em funções de acesso mais adequadas a forma de funcionamento de cada recurso associado. Essas, funções, em conjunto com o MediaPipe, permitirão o tratamento de fluxo de informação.

Ao definir interfaces de acesso para objetos com atributos do tipo áudio, vídeo, ou qualquer mídia contínua, que são modelados como fluxos de informação, deve-se levar em consideração a existência de qualificadores para os atributos, que distinguirão o sentido do fluxo, a iniciativa do objeto e a segmentação do fluxo. Tais qualificadores correspondem a uma determinada característica do recurso associado, como mencionado na seção 3.2.

Segundo o *sentido do fluxo de informação*, um recurso determina três alternativas para a interface de acesso ao atributo:

1. somente como *entrada (in)*;

2. somente como *saída (out)*;

3. *entrada e saída (inout)*.

Segundo a *segmentação do fluxo de informação*, um recurso determina duas alternativas para a interface de acesso ao atributo:

1. *blocado (blocked)*, no qual o acesso ao atributo é feito em blocos, através de sucessivas leituras ou escritas;

2. *contínuo (streamed)*, caso em que, uma vez disparado o processo de troca de informação, ele se dá de maneira contínua sem a necessidade de interferência.

A presença das alternativas de segmentação representa a existência de dois modelos para o tratamento da transferência de informação. Alguns autores [12] consideram esses modelos como sendo pertencentes a dois níveis de abstração diferentes, um lidando com a abstração de blocos de informação e outro operando diretamente sobre fluxos contínuos de informação. No presente trabalho, adota-se a estratégia de permitir ambas as possibilidades de interface de acesso visando facilitar as implementações, pois alguns dispositivos podem já ser definidos, em baixo nível (hardware e sistema operacional), com interfaces mais próximas da blocada (como arquivos) ou com interfaces mais próximas da contínua.[3] O MediaPipe pode se encarregar de compatibilizar as transferências entre recursos com diferentes características, sem ser necessário que o implementador assuma a responsabilidade de codificar todas as transferências através de uma única interface.

Segundo a *iniciativa*, um recurso determina duas alternativas para a interface de acesso ao atributo:

1. *mestre (master)*, que toma a iniciativa na transferência de informação;

2. *escravo (slave)*, necessitando da iniciativa de outras entidades para fornecer ou receber informações.

Um qualificador existe para definir a semântica das solicitações de acesso ao atributo, que pode ser, como definido na Seção 2.3, síncrona, assíncrona ou amostrada.

Em IDL, a definição de atributos contínuos utilizará uma nova palavra reservada `flow`, ao invés de `attribute`. Precedendo a palavra `flow`, pode-se encontrar qualificadores que corresponderão às alternativas anteriores. Palavras reservadas foram escolhidas para uma IDL estendida que permitirá a descrição de atributos como, por exemplo, um áudio associado a um recurso de entrada e saída, escravo e blocado.

```
interface MyMusic {
. . .
   synchronous inout slave blocked flow music;
. . .
};
```

A interação entre os objetos ocorre através de bindings explícitos, representados localmente, para cada objeto participante, como um pseudo-objeto denominado MediaPipe, como definido anteriormente. A interface do MediaPipe é perceptível aos objetos através do OA. A interface também se fará sentir na interação entre o OA e os Skeletons, embora, para os objetos, seja transparente.

As diferenças de interface entre os objetos com atributos `flow` deverão ser tratadas pelo compilador de IDL, que gerará o código para stubs e skeletons adequadamente. Por

[3] Câmeras ou outros dispositivos de captura podem, muitas vezes ter hadware capaz de efetuar acesso direto à memória (DMA) sem fazer chamadas ao sistema operacional para passar informações. Nesses casos, a informação pode ser transferida simplesmente habilitando o dispositivo que, a partir de então, realizará a transferência dos dados sem precisar recorrer a várias solicitações de escrita ou leitura de blocos. Placas de rede podem também fazer uso desse tipo de interface, de forma que, ao desejar transmitir informações, uma aplicação pode simplesmente habilitar a transferência e fornecer o endereço de um buffer que contenha as informações a serem transmitidas.

exemplo, para o compilador de IDL, a definição de um atributo `synchronous inout slave blocked flow` representa a produção de uma interface que contenha simplesmente operações de read e write síncronas usuais, que serão invocadas localmente, pelo MediaPipe associado ao objeto, através dos skeletons gerados.

4.4 Plataforma Básica

A plataforma básica para o desenvolvimento de um sistema hipermídia aberto distribuído inclui um ORB que dê suporte às extensões do modelo de objetos proposto neste trabalho, acompanhado de um conjunto de classes e objetos básicos, já construídos sobre esse ORB, responsáveis por funções específicas do ambiente hipermídia.

A arquitetura interna para a construção do ORB será enfocada em um trabalho futuro. Nesse momento, o interesse concentra-se nas classes básicas e suas interfaces na utilização dos serviços do ORB. As classes dos objetos de dados, DOs, e objetos de representação, ROs, são as duas classes básicas mais importantes enfocadas nesse artigo (Figura 3).

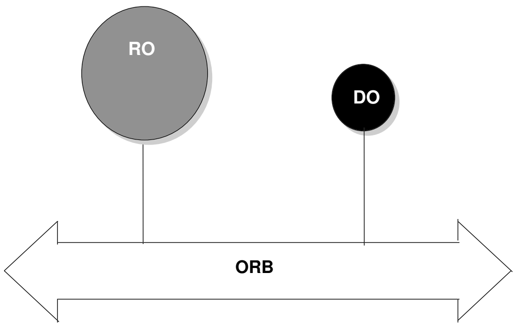

Figura 3: DOs e ROs.

O projeto dos DOs e ROs segue as seguintes orientações. DOs contêm atributos que correspondem à estrutura definida para os objetos do MHEG. Em algumas classes específicas de DOs, encontram-se atributos de tipo contínuo, cuja definição corresponde tão somente à geração de interfaces de acesso específico, que permitirão a comunicação adequada com o MediaPipe para a transferência da informação. A definição desses atributos deverá incorporar a descrição das características do recurso hospedeiro, como foi visto na Seção 4.3. O conteúdo propriamente dito poderá estar armazenado em arquivos, sendo capturado em tempo real, etc. Cabe à implementação de cada DO recuperar/entregar o conteúdo referente ao atributo, quando assim for necessário.

O modelo de apresentação do NCM determina que o processo de apresentação de um nó envolve a criação de um RO a partir de um DO, e a transferência da informação desse DO para o RO, onde ela poderá ser apresentada. As classes de ROs podem ser projetadas de várias formas. Uma, porém, tem se mostrado especialmente interessante, motivo pelo qual o restante da explanação nela se baseia. A proposta reside em projetar ROs como subtipos de determinados DOs. Mais ainda, especifica-se a utilização dos mecanismos de herança de implementação. Ou seja, após descrever ROs em IDL (como subtipos de DOs) e ICL, passa-se ao processo de compilação, quando então configura-se o `serc` para a geração de código referente a ligação direta entre os códigos de tipo e subtipos. Assim, garante-se que existe uma interação local de cada RO com um DO. Para ser possível a realização do projeto dessa forma, é necessário disponibilizar o código das classes de DOs a todos os nós de processamento do ambiente.[4]

A geração de ROs a partir de DOs através de herança de implementação é similar à idéia de que cada RO "contém" um DO, com a peculiaridade de que a comunicação entre eles é local (sem interferência do ORB). Adicionalmente, esse RO é cliente de algum outro DO — o DO que contém a informação que se deseja apresentar, que pode estar arbitrariamente localizado, e cuja interação com o RO será feita, no caso de objetos com atributos contínuos, através de um MediaPipe. Assim, o processo de interação para a apresentação de um nó arbitrariamente localizado envolve a criação de um RO a partir do DO correspondente ao nó a ser apresentado e a criação de um MediaPipe entre o RO e o DO, conforme ilustra a Figura 4.

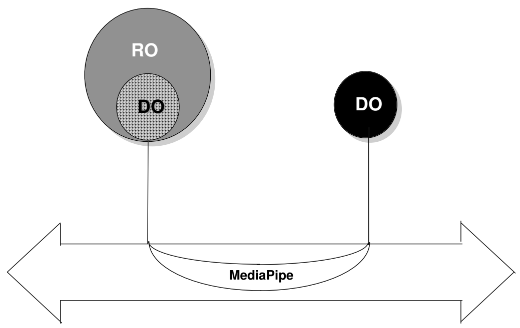

Figura 4: Interação entre DOs e ROs.

Na simbologia da Figura 4, o posicionamento de um DO dentro do RO bem junto a sua fronteira significa que a interface do RO naquele ponto está sendo diretamente oferecida pela interface de DO devido ao mecanismo de herança de implementação. Pela simbologia pode-se ter exata a idéia de que a comunicação se dá, de fato, entre DOs. Na realidade essa interpretação é também correta, pois sendo o RO obtido por herança do DO, ele *é um* DO. DOs utilizados diretamente na implementação de ROs (ou DOs clientes) têm características diferentes dos DOs associados a fonte das informações (DOs servidores). No caso de nós com atributo de tipo contínuo, ambos definirão, em IDL, um atributo `flow`. No cliente, porém, o qualificador de sentido do fluxo será sempre `out`, enquanto que no servidor será `in` ou `inout`.

O MediaPipe atua entre cliente e servidor no sentido de controlar o fluxo de informações, mantendo a transmissão de dados dentro de níveis de QoS aceitáveis, permitindo, inclusive, o ajuste dinâmico desses parâmetros considerados aceitáveis.

[4]Note que, a obrigatoriedade da disponibilidade local do código referente à implementação não implica na necessidade de que todas as instâncias de DOs estejam localmente disponíveis. Ao contrário, instâncias de DOs podem estar arbitrariamente distribuídas.

A arquitetura, em termos de níveis de comunicação pode ser representada pela Figura 5, que corresponde a um refinamento da apresentada em [10].

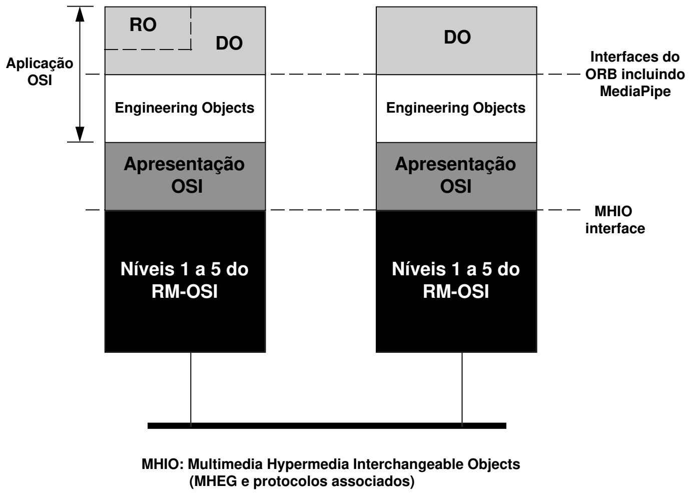

Figura 5: Arquitetura de comunicação para sistemas hipermídia.

O projeto de DOs como subtipos de ROs facilita também a implementação dos mecanismos de controle de versões dos objetos hipermídia [11], embora os detalhes relativos a esta afirmação estejam fora do escopo deste trabalho.

4.5 Objetos Ativos

Um sistema que oferece suporte a objetos ativos permite que cada objeto tenha, conceitualmente, seu próprio processador virtual, com controle de execução independente de qualquer outro. Objetos representam, assim, unidades de concorrência que podem ser ativadas e desativadas independentemente das demais. O CORBA não faz menção explícita a objetos ativos, mas eles podem ser facilmente implementados através de uma das políticas de ativação previstas para o OA.

Segundo o CORBA, o OA deve permitir que implementações de objetos sejam construídas de várias formas diferentes, podendo ser associadas a quatro diferentes políticas de ativação, conforme ilustrado na Figura 6.

As quatro políticas de ativação, rotuladas na Figura 6 de **A** a **D** correspondem: a ativação de uma unidade de execução para grupos de objetos (casos **A** e **B**),[5] a ativação de uma unidade de execução para cada objeto (caso **C**), ou a ativação de uma unidade de execução para cada operação (caso **D**). Para obter-se a independência desejada para a implementação de objetos ativos, a escolha da ativação deverá recair sobre uma das duas

[5] A diferença entre as alternativas A e B corresponde à necessidade, ou não, da ativação da unidade de execução pelo OA. Na alternativa B, o próprio servidor se encarrega de se registrar como ativado toda a vez que o sistema entra em funcionamento, sem a necessidade da ativação prévia por parte do OA. Servidores que seguem a alternativa B são denominados *persistentes*.

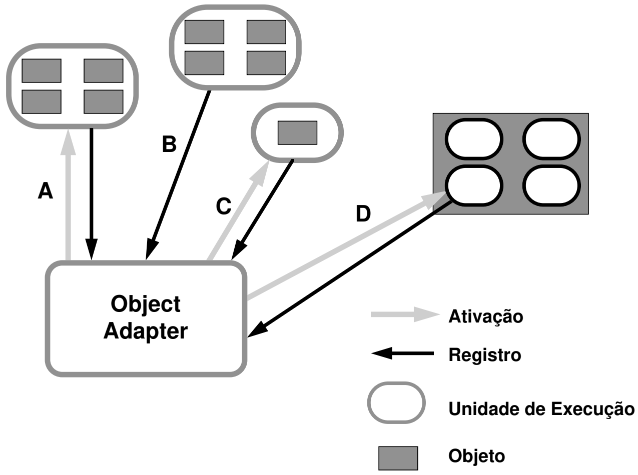

Figura 6: Políticas de ativação de implementações do CORBA.

últimas formas (**C** ou **D**). Para se decidir, deve-se levar em conta algumas características de ambientes para programação concorrente, como descrito a seguir.

Num ambiente distribuído, é possível que um objeto receba várias solicitações para seus serviços, incluindo casos onde várias solicitações são enviadas a uma mesma operação (como, por exemplo, para múltiplas exibições de uma imagem em diferentes janelas). A política de ativação impacta diretamente na granularidade de concorrência do sistema no momento do atendimento de várias solicitações simultâneas. Nos sistemas operacionais tradicionais, a entidade que possui independência suficiente para representar uma unidade de execução é o processo. Se a opção de implementação de objetos ativos recair sobre a forma **D**, um processo será disparado para cada operação solicitada. Entretanto, o ônus da ativação de processos é relativamente alto, principalmente quando necessita-se do melhor desempenho que se possa obter, inclusive no que diz respeito ao escalonamento, fator importante em sistemas hipermídia (que tratam com requisitos de tempo real). Soma-se a isso, o fato de que processos não têm, em geral, mecanismos que os permitam compartilhar recursos. Tais mecanismos são extremamente importantes, uma vez que operações definidas em um objeto, em geral, compartilham informações sobre o seu estado.

Percebe-se que a quantidade de recursos empregada e fornecida por um processo priva-o da flexibilidade desejada [14]. Uma flexibilidade maior provém de uma entidade complementar ao processo, empregada em sistemas operacionais mais recentes, denominada *thread*. Threads podem ser definidas como seções de um processo que podem compartilhar parte de seus recursos com outras threads do mesmo processo, além de serem escalonadas individualmente e possuir recursos próprios. Threads podem ser escalonadas de forma a permitir a abstração de vários processadores virtuais, mas não exigem a troca de contexto do processo, uma vez que pertencem, todas, ao mesmo espaço de endereçamento.

Com a utilização de threads, a política de ativação de objetos ativos recai sobre a forma descrita como **C** na Figura 6. Ao ativar o objeto, o OA dispara um processo, que poderá então receber solicitações para suas operações e ativar adequadamente suas threads. Um objeto de dados ativo, por exemplo, pode receber solicitações provenientes de vários objetos de representação diferentes, para a transferência de um vídeo. Tais solicitações podem ser atendidas concorrentemente através do disparo de várias threads relativas à mesma operação, permitindo, nesse caso, múltiplas exibições do mesmo vídeo.

Um objeto ativo de tempo real corresponde a um processo que pode comportar um número máximo (possívelmente infinito) de threads e é composto pelos seus dados encapsulados e um conjunto de operações. A cada operação ou conjunto de operações pode-se associar uma fila de entrada (denominada, por simplicidade, de entrada). As threads são ativadas para atender às solicitações que estejam enfileiradas em cada entrada, permitindo a concorrência no atendimento das solicitações ao objeto.

A utilização de um processo com várias threads para a implementação de objetos requer ainda alguns cuidados adicionais no que diz respeito à concorrência e aos requisitos de tempo de real, conforme descrito em [3]. O acoplamento do conceito de threads ao modelo de objetos ativos adiciona um nível de concorrência interno ao objeto, que deve ser tratado com os devidos cuidados relativos a controles de acesso a dados compartilhados, tipicamente atributos do objeto. Implementações de operações que façam acesso direto a atributos deverão tratar de acionar mecanismos como semáforos ou semelhantes. Alternativamente, caso o acesso seja feito através de operações específicas (tipicamente, a definição de attributes do CORBA, que corresponde, em termos de interface, à definição de operações *set* e *get*), é possível encapsular tais mecanismos nessas operações.

O grau da concorrência é limitado, em primeira instância, pelo número de threads definido para o processo, em segunda instância, pelo tamanho das filas de entrada e, por último, pelo número máximo de threads (possivelmente ilimitado) que podem ser associadas a uma mesma entrada. Esse último número, em conjunto com o tamanho das filas, é ainda particularmente responsável pelo grau de paralelismo que uma mesma operação (ou conjunto de operações) pode oferecer. Note que, um controle de disparo de threads é feito pelo objeto de forma a respeitar os limites de número de threads total e por operação. O OA, portanto, continua a trabalhar no nível de ativação de objetos, que corresponde ao nível de processo, enquanto que o objeto, uma vez a par dos limites impostos, é capaz de exercer o controle sobre suas threads.

Nas entradas, pode-se considerar parâmetros adicionais para controlar o enfileiramento de solicitações e o escalonamento das threads. Parâmetros comuns utilizados para escalonamento incluem nível de prioridade da solicitação e deadline do serviço [3]. Portanto, o suporte de sistemas operacionais de tempo real torna-se indispensável.

5 Conclusões

Apresentou-se, neste trabalho, o resultado de um conjunto de extensões ao modelo de objetos do CORBA com o objetivo principal de permitir a definição de tipos de dados contínuos, cuja interface de acesso pode ser configurada pelo implementador de acordo com as características próprias dos recursos associados à informação. Atributos contínuos, segundo a descrição aqui apresentada, permitem a conexão explícita entre objetos através da abstração do MediaPipe, que pode ser utilizado para controlar o fluxo e a qualidade de serviço da transferência.

Foram brevemente revistos alguns padrões para o desenvolvimento de sistemas abertos que utilizam o paradigma de orientação a objetos, sobre os quais já se encontram algumas propostas para o tratamento dos requisitos de sistemas multimídia. Um deles inclui a única extensão sobre o CORBA que se tem notícia, descrita pelo padrão PREMO, que inclui novos conceitos para tratamento de eventos, operações amostradas e objetos ativos, também presentes na proposta aqui apresentada. O modelo, juntamente com a descrição genérica do ambiente de desenvolvimento e da plataforma de aplicação, generaliza os conceitos encontrados nesses padrões e adapta-os ao ambiente do CORBA. Tratou-se ainda de considerar, em linhas gerais, o problema da herança em sistemas distribuídos e a concorrência no atendimento a solicitações para um objeto.

Até agora, o grupo de pesquisa do laboratório Telemídia tem desenvolvido a plataforma básica para sistemas hipermídia (o HyperProp) para um ambiente centralizado, utilizando os conceitos usuais presentes em linguagens de programação como C++. Na fase atual, o objetivo é construir o arcabouço para a migração dessa plataforma para um ambiente distribuído, interconectado através de uma rede de alta velocidade capaz de fornecer os serviços necessários. A arquitetura da plataforma descrita nesse artigo encontra-se em fase final de especificação e uma implementação começa a ser conduzida.

Referências

[1] M. A. Casanova, L. Tucherman, M. J. Lima, J. L. Rangel Netto, N. de La Roque Rodriguez, and L. F. G. Soares. The nested context model for hyperdocuments. In *Proceedings of Hypertext'91*, Texas, december 1991.

[2] G. Coulson, G. S. Blair, N. Davies, and N. Williams. Extensions to ANSA for multimedia computing. *Computer Networks and ISDN Systems*, (25):305–323, 1992.

[3] L. Guangxing. Distributing real-time objects: Some early experiences. Technical report, ANSA, october 1994.

[4] I. Herman, G. J. Reynolds, and J. Van Loo. PREMO: an emerging standard for multimedia presentation. Technical report, CWI, Computer Science/Department of Interactive Systems, 1995. CS-R9554.

[5] ISO/IEC JTC1/SC21/WG7. *ISO/IEC DIS 10746-1/ITU-T X.901 — Reference Model of Open Distributed Processing, Part 1: Overview*, may 1995. output from the editing meeting in Helsink (Finland).

[6] ISO/IEC JTC1/SC24. *ISO/IEC 14478-1 — Presentation Environment for Multimedia Objects (PREMO), Part 1: Fundamentals*, april 1995.

[7] ISO/IEC JTC1/SC29/WG12 Multimedia and Hypermedia Information coding Expert Group (MHEG). *ISO/IEC DIS 13522-1 — Coded Representation of Multimedia and Hypermedia Information (MHEG), Part I: Base Notation (ASN.1)*, october 1994.

[8] G. Lemos, L. F. G. Soares, and M. A. Casanova. Synchronization Aspects of an Hypermedia Presentaion Model with Composite Nodes. In *ACM Workshop on Effective Abstractions in Multimedia*, São Francisco, EUA, november 1995. In connection with the ACM Multimedia'95.

[9] Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification*, 1991. Document number 91.12.1.

[10] L. F. G. Soares, M. A Casanova, and S. Colcher. An architecture for hypermedia systems using MHEG standard objects interchange. *Information Services & Use*, 13(2):131–139, 1993. Special Issue: Hypermedia and Hypertext Standards.

[11] L. F. G. Soares, M. A. Casanova, and N. de La Rocque Rodriguez. Nested Composite Nodes and Version Control in a Open Hypermedia System. *International Journal on Information Systems*, 20(6):501–520, september 1995. special issue on Multimedia Information Systems.

[12] R Steinmetz and K. Nahrstedt. *Multimedia Computing, Communications and Applications*. Prentice Hall, 1995.

[13] R. van der Linden. An overview of ANSA. Technical report, ANSA, July 1993.

[14] J. A. W. Werner. RIO: Metodologia e suporte para sistemas distribuídos configuráveis. Master's thesis, Departamento de Engenharia Elétrica — PUC-rio, Março 1995.