

Capítulo 5

Hipervisão: Um estudo de caso

nada se torna real até que seja experimentado
John Keats

5.1 - Introdução

Ao longo dos capítulos anteriores, esse trabalho tem sido estruturado de forma que cada capítulo pudesse suprir uma demanda gerada no capítulo anterior. Do mesmo modo, o objetivo desse capítulo é suprir a demanda de vários estudos de caso gerada por todas as propostas que têm sido feitas ao longo dessa dissertação. No Capítulo 2, foi proposto um modelo de processo genérico de desenvolvimento de software para/com reuso, que contempla, de forma complementar, as disciplinas de engenharia de software e engenharia de domínio. Além disso, o capítulo defende o fato de que o desenvolvimento de software deve ser encarado como uma disciplina de engenharia, ou seja, deve agrupar atividades gerenciais como avaliação da qualidade, documentação, análise de risco e planejamento, além de pregar que a escolha das atividades de construção deve ser baseada nas características da equipe de desenvolvimento, nos recursos disponíveis, no conhecimento disponível acerca do domínio e, principalmente, nos requisitos funcionais e não funcionais do produto a ser desenvolvido. Para desenvolvimento das atividades referentes à Engenharia de Domínio, uma metodologia foi proposta nos capítulos 3 e 4. O capítulo 3 apresenta um método sistemático para a realização da fase de análise de domínio através do uso de ontologias formais e o capítulo 4 introduz uma linguagem matemática para formalização de ontologias e um conjunto de ferramentas metodológicas para a derivação sistemática de infra-estruturas de domínio (no caso *frameworks* orientados a objetos).

Dessa forma, esse capítulo é estruturado de modo a agrupar de forma complementar os vários estudos de caso requeridos, culminando no desenvolvimento de uma aplicação. Primeiramente, a seção 5.2 promove uma discussão detalhada a respeito dos requisitos metodológicos da área de sistemas multimídia distribuídos, a fim de instanciar o modelo proposto no capítulo 2. Como resultado, é proposto um processo de desenvolvimento, que supre inúmeras demandas não cobertas pelos métodos atualmente empregados para o desenvolvimento dessa classe de aplicações.

O domínio de Vídeo sob Demanda é o sub-domínio dos sistemas multimídia distribuídos escolhido para ser alvo da aplicação do processo instanciado. Por isso, esse domínio é profundamente discutido na seção 5.3. Após essa fase, na seção 5.4, são

apresentados dois estudos de casos, referentes às propostas feitas nos capítulos 3 e 4, abordando a perspectiva de desenvolvimento para reuso. Primeiramente, é desenvolvida uma ontologia de gerência de Vídeo sob Demanda e em seguida é gerado o *framework* correspondente.

Por fim, a seção 5.5 apresenta um estudo de caso de desenvolvimento com reuso, ao desenvolver uma aplicação de vídeo sob demanda a partir de dois componentes existentes: o *framework* de gerência, desenvolvido na seção 5.4, e um outro componente chamado *Java Media Framework*. A seção 5.6 apresenta as conclusões do capítulo.

5.2 - Sistemas Multimídia Distribuídos

Os anos recentes experimentaram um enorme avanço nas tecnologias de computadores e comunicação de dados, os quais constituíram o suporte tecnológico necessário para o surgimento de uma nova e promissora classe de aplicações - as **aplicações multimídia** - e abriram espaço para a concepção de novos serviços nas mais variadas áreas do conhecimento humano, como, por exemplo, educação, medicina, entretenimento, turismo, publicidade, negócios e indústria, entre outras.

A multimídia constitui uma tecnologia interdisciplinar que orienta aplicações que visam atender às necessidades multi-sensoriais da natureza humana. Sistemas multimídia tornam isso possível devido às novas alternativas de integração, manipulação, armazenamento, transmissão, exibição e interação de diferentes formatos de mídia (texto, gráfico, animação, áudio e vídeo).

Os sistemas distribuídos são caracterizados pela existência de vários nós autônomos de processamento, possivelmente distribuídos geograficamente, que cooperam através da coordenação de atividades e troca de informações para a realização de objetivos comuns - o provimento de um conjunto de serviços. O casamento dessas duas tecnologias - sistemas multimídia e sistemas distribuídos - tendo como suporte de comunicação, idealmente, uma rede multi-serviços de alta-velocidade, resulta nos sistemas multimídia distribuídos (Gonçalves, 1996).

Estas aplicações revelam novos problemas e introduzem conceitos que afetam profundamente as arquiteturas de comunicação clássicas. Nesses últimos anos, várias pesquisas vêm sendo realizadas nestas áreas, porém, boa parte dos trabalhos aborda o problema do ponto de vista tecnológico, tentando obter desempenhos ótimos de meios de armazenamento e comunicação. A seção seguinte discute essa classe de sistemas sob uma visão diferente, focando nos aspectos metodológicos de desenvolvimento.

5.2.1 – Aspectos metodológicos de Sistemas Multimídia Distribuídos

À primeira vista, os principais requisitos dos sistemas multimídia distribuídos estão ligados a problemas de projeto, como sincronização de mídias contínuas, fortes relações temporais, fortes requisitos de tempo-real e de desempenho, forte comunicação entre objetos - especialmente objetos distribuídos - e grande presença de objetos baseados em estado. Requisitos dessa natureza necessitam de um apoio metodológico robusto, capaz de especificá-los, validá-los e simulá-los. Dessa forma, especificação e validação, incluindo verificação, simulação e teste, são de importância crucial no ciclo de vida dos sistemas distribuídos, sobretudo para o desenvolvimento de suas aplicações. A especificação não pode ser ambígua e deve ser livre de erros tanto quanto possível. Para que haja uma integração bem sucedida de novos produtos, é necessário que testes criteriosos sejam realizados em relação à especificação (Souza, 1999).

As metodologias de desenvolvimento vigentes apresentam sérias deficiências nessas áreas. Técnicas informais ou semi-formais, normalmente geram especificações incompletas, contendo problemas como ambigüidades, contradições, falta de precisão e falta de controle da separação de níveis diferentes de abstração (Pressman, 1997). A dificuldade de uma abordagem formal, também para verificação e validação, faz com que erros sejam freqüentemente detectados somente em fases tardias do processo de desenvolvimento.

Com o aumento da complexidade de tais sistemas e com a necessidade de estabelecimento de padrões, Técnicas de Descrição Formal (TDFs) foram desenvolvidas, visando a produção de especificações claras, concisas e consistentes. Com base na sintaxe e semântica formais da TDF utilizada para a especificação, validações da própria especificação e do projeto do sistema podem ser realizadas. Utilizando um compilador para a TDF empregada, implementações semi-automáticas podem ser geradas. Com base na especificação formal do sistema, seqüências de teste podem ser geradas e linhas de execução da implementação, podem ser verificadas (Souza, 1999).

Vários exemplos de uso de TDFs no ciclo de vida de sistemas distribuídos complexos são relatados na literatura, entre eles o projeto ESPRIT LOTOSPHERE (Bolognesi et al., 1995) e o projeto francês "Conception Formelle de Systèmes Multimédias Coopératifs à Hauts Débits (Cesame)", fruto de uma colaboração entre o Centre National d'Études de Télécommunications (CNET) e o Centre National de Recherche Scientifique (CNRS), sendo este último um dos mais significativos no momento (Diaz & Pays, 1994).

Desse modo, tradicionalmente, os modelos de processo propostos pela comunidade de sistemas multimídia distribuídos são fundamentados no uso de refinamentos sucessivos de especificações construídas com técnicas de descrição formal, como LOTOS (Language of Temporal Ordering Specification) (ISO IS 8807, 1989), ESTELLE (Extended State Transition Language) (ISO IS 9074, 1989) e SDL (Specification and Definition Language) (ITU-T Z 100, 1994), entre outras, altamente difundidas em áreas como engenharia de protocolos e sistemas de tempo real. Essas técnicas tratam apropriadamente requisitos como controle de sincronização e de tempo real, possibilitando sua simulação e verificação com alto formalismo e rigor matemático.

Seguindo a mesma filosofia, em 1996 foi criado no Brasil um projeto multi-institucional denominado DAMD (Design de Aplicações Multimídia Distribuídas) com objetivo de abordar aspectos metodológicos da construção dessa classe de sistemas. Este projeto, financiado pelo CNPq, programa temático ProTem-CC, Fase III, contou com a presença das seguintes instituições: Universidade de São Carlos (UFSCar), Universidade Federal de Santa Catarina (UFSC), UFRGS (Universidade Federal do Rio Grande do Sul), Universidade Federal do Espírito Santo (UFES) e a Universidade de Twente (Enschede, The Netherlands).

O projeto DAMD foi concebido visando três linhas mestras:

1. desenvolvimento de uma metodologia, suportada por um modelo formal (E-LOTOS¹) e ferramentas para a especificação, validação, tradução e teste de aplicações multimídia distribuídas;

¹ E-LOTOS: Enhancements to LOTOS - É uma extensão temporizada de LOTOS

2. desenvolvimento de aplicações para a avaliação da metodologia e teste dessas aplicações em ambiente distribuído;

3. pesquisa visando propor uma nova TDF para dar suporte ao design de futuras aplicações distribuídas.

O projeto foi concluído em dezembro de 1998, produzindo os seguintes resultados:

- (a) Uma ferramenta de autoria de apresentações multimídia;
- (b) Um ambiente para criação de especificações em E-LOTOS usando um linguagem gráfica (E-DART);
- (c) Um Simulador e um Verificador para especificações E-LOTOS;
- (d) Um conjunto de diretivas para especificação de Documentos Multimídia em E-LOTOS;
- (e) Um Tradutor E-LOTOS/MHEG-5²;
- (f) Estudos de caso de modelagem de aplicações multimídia distribuídas usando diferentes formalismos.

O sistema de vídeo sob demanda Hipervisão, apresentado ao longo deste capítulo, foi construído neste contexto (Guizzardi & Gonçalves, 1999b), a fim de explorar a segunda linha de atuação do projeto. No processo de desenvolvimento desta aplicação, foi observado que a metodologia proposta era mais adequada para o desenvolvimento de uma classe específica de aplicações multimídia distribuídas. Além disso, devido ao escopo do projeto, o processo proposto não contemplava de forma sistemática um desenvolvimento para/com reuso e nem previa a adoção de atividades não tecnológicas como avaliação da qualidade, planejamento e análise de risco.

A metodologia proposta no DAMD, assim como a enorme maioria das metodologias existentes, não considera o fato de que essas aplicações são geralmente utilizadas em domínios imaturos do conhecimento, de natureza complexa e pouco conhecidos, e que, portanto, definem um novo conjunto de requisitos de natureza completamente diferente - os requisitos relacionados ao domínio do problema. Quando é considerada, a fase de análise de requisitos é tratada por esses métodos de duas maneiras: (1) através do uso técnicas informais complementares, ou (2) empregando as mesmas TDFs usadas para resolver os problemas de projeto.

No primeiro caso, a especificação de requisitos será exposta a todos os problemas já discutidos no uso de abordagens informais, tornando possível que a presença de inúmeras restrições, inconsistências e contradições presentes no domínio, façam do modelo de requisitos - que em um sistema desse tipo é produto de uma atividade extensa e custosa - um artefato praticamente impossível de ser validado e reutilizado.

No segundo caso, como foi observado no desenvolvimento do sistema Hipervisão, o tipo de semântica formal empregado nas TDFs utilizadas, embora apropriado para modelagem de problemas de projeto, dificulta seu uso na modelagem de requisitos em níveis mais altos de abstração como, por exemplo, os níveis equivalentes às fases de análise e especificação de requisitos e análise de domínio. Essa limitação deve-se ao fato de ser altamente indesejável que a construção de um modelo do problema esteja limitada a fortes características semânticas de uma linguagem específica, assumindo compromissos ontológicos que não existem no mundo real.

² MHEG: Multimedia and Hypermedia Expert Group - Padrão ISO para Modelo de Documentos Hipermídia (ISO 13522-1, 1994)

Dessa forma, a área de sistemas multimídia distribuídos, devido a sua natureza complexa e heterogênea, apresenta inúmeras características que justificam a existência de metodologias específicas para a modelagem e construção de suas aplicações. A concepção dessas metodologias apresenta-se como um grande desafio, principalmente pela necessidade de comportar, em um mesmo processo de desenvolvimento, uma flexibilidade de modelagem que permita o emprego de diversos níveis de formalismo.

Em (Bowen & Hinchley, 1994) são apresentados os chamados "Dez mandamentos dos métodos formais" - um conjunto de diretivas para aplicação de métodos formais em engenharia de software. As diretivas reforçam a necessidade de manutenção das atividades de estimativas de custos, documentação e avaliação da qualidade, e defendem a idéia de que o processo de desenvolvimento tem que ser encarado do ponto de vista de engenharia, ou seja, a escolha de métodos, técnicas e ferramentas deve ser feita de forma apropriada, dependendo principalmente do produto a ser construído. Desta forma, várias conclusões importantes são apresentadas:

- (a) Métodos formais não têm necessariamente que ser aplicados a cada aspecto de um sistema;
- (b) É bastante desejável pensar na integração de métodos formais com métodos tradicionais orientados a objetos. Cada um deles tem vantagens, que associadas de forma complementar podem produzir excelentes resultados;
- (c) É interessante pensar em uma abordagem multi-formalismo, na qual vários métodos formais diferentes possam ser associados para resolver problemas específicos a diferentes fases do processo de desenvolvimento.

Várias metodologias vêm sendo criadas, a fim de integrar métodos formais em um ciclo tradicional de desenvolvimento orientado a objetos. Alguns exemplos são: INSYDE (Holz et al., 1996, Sinclair et al., 1995) que, por ser uma metodologia de co-design de hardware e software, integra o método OMT (Rumbaugh et al., 1991) com as TDFs SDL para especificação de sistemas de software e VHDL (ANSI/IEEE, 1993) para sistemas de hardware, ARENA/SOMT (Auchter, 1997, Telelogic, 1996), que integra OMT com a uma combinação de SDL e MSC³, e Metamorphosis (Araújo & Sawyer, 1998) - integra OMT a Object-Z. Apesar disso, nenhuma das metodologias apresenta um suporte sistemático ao desenvolvimento para/com reuso, nem tão pouco aborda questões relativas aos problemas de domínio .

Desse modo, a fim de suprir os requisitos não atendidos pelos métodos atualmente existentes, este trabalho instancia o modelo genérico de desenvolvimento para/com reuso proposto no Capítulo 2, aplicando-o ao desenvolvimento de sistemas multimídia distribuídos. O modelo de processo instanciado é discutido detalhadamente na subseção 5.2.2 e utiliza de maneira complementar conceitos oriundos das áreas de métodos formais, engenharia de software orientada a objetos e sistemas baseados em conhecimento.

É importante ressaltar que, apesar deste modelo ter sido concebido a partir de necessidades identificadas na construção de sistemas multimídia distribuídos (vídeo sob demanda, sistemas de conferência multimídia, educação a distância, entre outros), acredita-se que os benefícios por ele oferecidos não são limitados a esta classe de sistemas.

³ *Message Sequence Charts* : linguagem formal, gráfica e textual, para a descrição e especificação do comportamento de interações entre componentes de um sistema. Pode ser usada para especificação de requisitos, simulação, validação e documentação de sistemas de tempo-real (ITU-T Z 120, 1994).

5.2.2 - Um processo alternativo de desenvolvimento

Tradicionalmente, o modelo de processos da engenharia de software orientado a objetos é composto das seguintes fases: *Análise e Especificação de Requisitos*, *Construção (projeto e implementação)* e *Testes*. A fase de Análise e Especificação de Requisitos é construída com base no estudo dos requisitos conceituais necessários para o desenvolvimento do sistema. Nessa fase, o modelo de requisitos é desenvolvido usando mecanismos informais (linguagem textual) e semi-formais (modelo de objetos do domínio do problema, modelo de interação entre objetos), levando em conta a perspectiva do usuário, isto é, como o sistema irá oferecer o serviço aos seus potenciais clientes. O sistema é expresso em termos de seqüências de eventos com comportamentos relacionados, que encapsulam um serviço atômico oferecido ao usuário (os chamados *casos de uso*). Uma vez concluída, esta especificação representa uma versão estruturada da visão do cliente acerca do domínio. Neste mesmo estágio, a especificação é organizada em termos de objetos lógicos, os quais, então, serão adequados para o universo computacional e implementados na fase de construção.

Nessa seção, um modelo orientado a objetos para o desenvolvimento de sistemas multimídia distribuídos é instanciado a partir do modelo genérico proposto no Capítulo 2. Esse modelo visa contemplar soluções tanto para os problemas de domínio, quanto para os problemas de projeto existentes nessa classe de sistemas, conforme discutido anteriormente (Guizzardi & Gonçalves, 1999c,d).

Primeiramente, para abordar os problemas de domínio, o modelo utiliza a abordagem de engenharia de domínio presente no modelo genérico. Para a realização dessa fase, será utilizada a metodologia proposta nos Capítulos 3 e 4. Esta abordagem permite a comunicação a respeito do domínio e a sua validação de maneira formal, além de incentivar a prática de reutilização em um nível de conhecimento. Por fim, vale salientar a importância do modelo de domínio produzido no auxílio à elicitação de requisitos específicos da aplicação.

Em segundo lugar, a fim de abordar os problemas de projeto, o modelo propõe uma extensão do processo tradicional, através da formalização de casos de uso selecionados. Na fase de projeto de um processo convencional de desenvolvimento OO, geralmente são empregados diagramas semi-formais para abordar os problemas descritos anteriormente. Estes diagramas podem ser de três tipos: (a) *diagramas de atividade* - para modelar atividades concorrentes; (b) *diagramas de transição de estado* - para modelar a mudança de estado de alguns objetos; (c) *diagramas de interação* - para modelar a troca de mensagens entre objetos (Fowler, 1997). Desta maneira, são identificadas as funcionalidades que devem ser oferecidas, isto é, as operações a serem implementadas por cada classe. No entanto, esses diagramas são limitados por sua natureza informal.

O modelo que será instanciado propõe que na fase de projeto, os casos de uso que apresentam a existência dos problemas citados anteriormente, sejam selecionados e descritos em uma combinação de SDL/MSD e que, portanto, possam ser devidamente formalizados. Além de impor um maior formalismo às relações dinâmicas envolvendo estes objetos, o uso de uma TDF baseada em um modelo de MEFES⁴, cujas interações acontecem através da troca de mensagens, permite, de maneira cômoda, agrupar as funcionalidades dos três tipos de diagramas citados (atividade, transição de estados e interação) em uma só

⁴ Máquinas de estados finitos estendidas

especificação. Esta combinação de TDFs é escolhida por suportar diretamente os conceitos de orientação a objetos e por sintetizar de maneira conveniente as funcionalidades de todos esses diagramas geralmente empregados. É importante ressaltar que o emprego de métodos formais é proposto como uma alternativa à formalização dos casos de uso, empregando esta técnica somente naqueles casos em que ela se mostra necessária.

A figura 5.1 mostra as atividades de construção da fase do desenvolvimento com reuso desse processo. Os retângulos na figura representam as atividades, enquanto que os retângulos com bordas arredondadas representam os diversos modelos gerados por essas atividades. Como pode ser observado, na fase de projeto, dois caminhos alternativos podem ser seguidos, sendo a escolha feita baseada no grau de formalidade necessário ao tratamento de um caso de uso.

Um vez definido, este processo será utilizado nas seções seguintes para o desenvolvimento de uma ontologia de gerência de vídeo sob demanda, de um *framework* derivado dessa ontologia e, finalmente, de um aplicação de vídeo sob demanda.

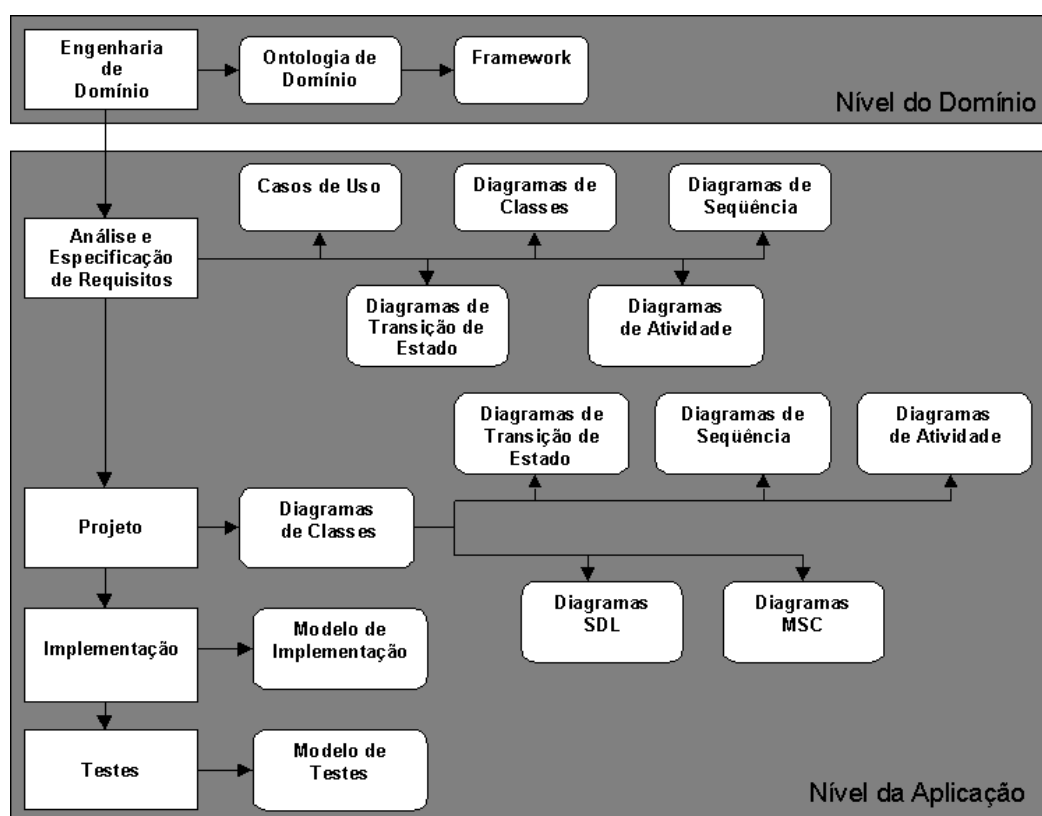


Figura 5.1 - Atividades de construção de um processo orientado a objetos para construção de sistemas multimídia distribuídos

Por fim, um aspecto importante da utilização conjunta de métodos OO e TDFs são as regras de transformação que permitem as transições entre esses dois paradigmas. Estas transformações ocorrem em duas etapas. Primeiramente, é necessário que se possa derivar de forma sistemática especificações formais em nível de projeto, para os diagramas de classes de análise dos casos de uso selecionados. Em segundo lugar, é também necessário

que, a partir dessas especificações, possa ser gerado código em uma linguagem orientada a objetos (idealmente de forma automática).

Nesse trabalho, apenas para a primeira etapa de transformação foram utilizadas algumas diretivas propostas no projeto ESPRIT INSYDE. Nesse projeto, são criadas várias regras formais de transformação de modelos OO em diagramas SDL. O projeto apresenta uma notação intermediária entre as fases de análise e projeto - OMT* - que apresenta um subconjunto das expressões válidas de OMT, mas que contém uma semântica bem definida (Wosowski et al., 1996, Jonckers et al., 1995, Verschaeve et al., 1996).

Um outro método que aborda estas questões é o SOMT (Telelogic, 1996) que apresenta as diversas formas de interação entre modelos OO, diagramas SDL e MSC. No entanto, apesar dos trabalhos existentes, a definição da rastreabilidade (*traceability*) entre os diversos modelos produzidos nesse contexto não é uma tarefa trivial, podendo ser explorada em diversos trabalhos futuros.

5.3 - O Domínio de Vídeo sob Demanda

Diferentemente de um sistema de televisão tradicional, onde o telespectador é um agente passivo, que tem acesso ao serviço sem nenhuma interação com as informações a ele transmitidas, um sistema de vídeo sob demanda (*VoD – video on demand system*), por outro lado, é um sistema multimídia que oferece aos seus usuários a funcionalidade de acesso a um servidor remoto de vídeos (por exemplo, a partir de um terminal presente em casa ou escritório), atuando sobre o mesmo em vários níveis de interação e com ampla liberdade de escolha de programação. As conexões de vídeo são estabelecidas sob demanda, através de uma rede de comunicação que liga o cliente ao servidor de vídeo.

Sistemas de vídeo sob demanda têm um potencial enorme de aplicação e alguns protótipos vêm sendo desenvolvidos, em particular nas áreas de educação, medicina e entretenimento. Diversos estudos de comportamento feitos com usuários nos EUA mostram que o serviço de APPV (*Advanced Pay-Per-View*), que é um dos tipos de serviço de vídeo sob demanda, será o serviço mais utilizado nos próximos anos, sendo no futuro superado pelos serviços de jogos interativos sob demanda e o verdadeiro serviço de vídeo sob demanda, ambos sempre se apresentando com um potencial de mercado muito grande (Cecilio & Rodrigues, 1996b) (figura 5.2).

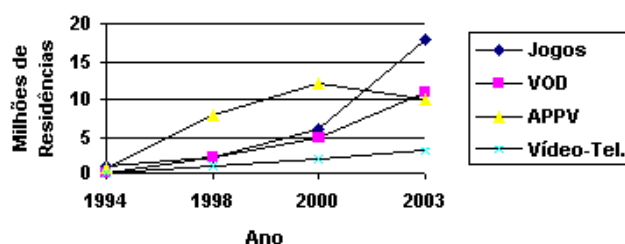


Figura 5.2- Evolução da Demanda dos Serviços de TV Interativa (Cecilio & Rodrigues, 1996b)

Os sistemas de VoD podem ser classificados como a seguir, de acordo com seu grau de interatividade (Little & Venkatesh, 1994):

- 1 **Broadcast (No VoD):** Sistema de difusão de TV sobre o qual o usuário não tem nenhuma interação. Dessa forma, não constitui na verdade um sistema de VoD, sendo classificado apenas em caráter de comparação.
- 2 **PPV (Pay-per-view):** Sistema similar ao que existe hoje na TV por assinatura, em que o usuário escolhe e paga por uma determinada programação que deseja assistir (similar aos serviços PPV atualmente existentes nos ambientes de CATV e DBS - *Direct Broadcast Satellite*).
- 3 **QVod (Quasi VoD):** Esta modalidade é baseada em segmentos temporais, múltiplos da duração do filme. Cada usuário tem direito a escolher o vídeo que deseja assistir. Imediatamente após a escolha, o sistema tenta agrupar dentro de um mesmo segmento temporal todos os usuários que demandam por um mesmo vídeo. Se no exato momento em que o usuário escolhe um vídeo para assistir a exibição deste filme estiver começando, o usuário irá recebê-lo imediatamente, caso contrário será obrigado a esperar até o início do próximo segmento.
- 4 **NVoD (Near VoD):** Este caso se apresenta como uma extensão do QVoD, no que diz respeito ao poder de controle disponibilizado para o usuário. Apesar de ainda não possuir uma transmissão exclusiva para cada usuário, possibilita algum controle sobre a exibição do vídeo. Porém, esta interação está associada aos segmentos temporais, no sentido de que o usuário só pode retroceder, adiantar ou fazer uma pausa no vídeo em intervalos temporais múltiplos destes segmentos. É importante ressaltar a preocupação que deve ser tomada na escolha do segmento temporal, pois o que temos aqui é uma troca entre um maior nível de interação (segmentos menores) e uma maior economia de largura de banda (segmentos maiores).
- 5 **TVoD (True VoD):** Neste caso, o usuário tem acesso a todas as funções de um videocassete virtual, podendo então exercer o controle total sobre a execução do vídeo. É como se existisse um canal exclusivo para cada usuário, pois mesmo que dois usuários decidam assistir a um mesmo filme simultaneamente, muito provavelmente os dois irão interagir com o mesmo de formas diferentes.

O domínio modelado neste trabalho é um serviço de TVoD. Apesar disso, é importante ser dito que apenas a nomenclatura vídeo sob demanda ou simplesmente VoD será usada daqui em diante, com o intuito de endereçar esta classe específica de sistemas de vídeo sob demanda.

Para que um serviço de vídeo sob demanda possa ser oferecido, vários são os requisitos tecnológicos necessários. Alguns deles exercem uma influência direta nas possibilidades de concretização dos subsistemas que compõem a arquitetura do serviço, nos seus modelos de distribuição e, em última instância, na ontologia de VoD que será produzida. A seguir, são discutidos os principais requisitos deste tipo e as alternativas existentes para endereça-los, tanto do ponto de vista tecnológico quanto econômico. Por fim, é interessante observar a influência que requisitos dessa natureza têm na proposição de arquiteturas conceituais.

5.3.1 - Requisitos Tecnológicos

A Figura 5.3 mostra uma ilustração simplificada de como um serviço de vídeo sob demanda é visto do ponto de vista do usuário. O cenário é composto de três subsistemas: o

terminal através do qual o usuário tem acesso ao serviço, o servidor de vídeos digitalizados e o sistema de comunicação responsável por fazer a conexão entre os dois.

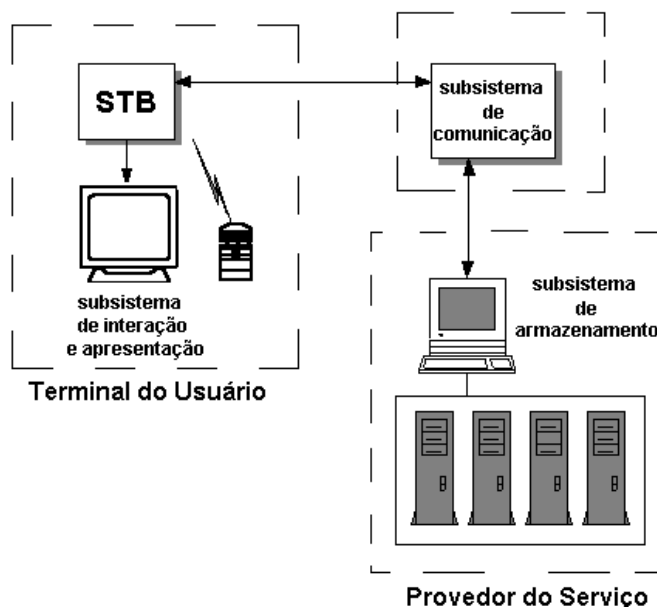


Figura 5.3 - Visão simplificada de um cenário de VoD

Os principais requisitos tecnológicos impostos a um sistema desse tipo estão ligados ao seu comprometimento com o transporte, armazenamento e apresentação de mídias contínuas, como áudio e vídeo, e dessa forma, exercem diretamente um impacto em todos os subsistemas citados. Esses tipos de mídia são notórios pela alta demanda de desempenho que impõem ao sistema, principalmente no que diz respeito ao espaço de armazenamento e largura de banda de transmissão requeridos e pela validade temporal de seus dados. Para melhor exemplificar esta necessidade, o seguinte exemplo será usado:

1. Suponha um *frame* de vídeo com resolução de 640x480 pixels e com profundidade de 24 bits. Este frame sozinho ocupa um espaço de $640 \times 480 \times 3$ bytes = 900 Kbytes.
2. Considerando uma taxa de *frames* de 30 frames/s (padrão NTSC), então um segundo deste vídeo necessita de uma largura de banda de transmissão de aproximadamente 27 Mbytes/s.
3. Em média, um filme tem duração de 90 minutos (1 1/2 hora), portanto, um único filme requer, em média, 143 GB de espaço de armazenamento.
4. Agora suponha que este vídeo possui também um áudio *stereo* associado, com qualidade de CD (16 bits/amostra e taxa de amostragem de 44100 Hz). Um segundo deste áudio ocupa aproximadamente 88 KB e, portanto, o áudio de um único filme de 90 minutos ocupa cerca de 475 MB de espaço de armazenamento.

Por estas razões proibitivas, o desenvolvimento de bons algoritmos de compressão específicos para estes sistemas tornou-se crucial para o desenvolvimento de aplicações

multimídia distribuídas em geral. Os principais formatos usados para codificação de mídias contínuas são os padrões ITU-T H.263 e ISO MPEG - Motion Pictures Expert Group (Saywood, 1996), sendo esse último o recomendado pelo DAVIC(Digital..., 1998)⁵ para aplicações de VoD. Taxas de compressão típicas alcançadas pelo MPEG são de 100:1 para vídeo e 10:1 para áudio. Apesar da redução significativa alcançada, a largura de banda de transmissão requerida ainda é grande o bastante para influenciar diretamente na escolha das alternativas de implementação dos subsistemas que compõem o serviço. Os padrões MPEG atualmente existentes para codificação de vídeo são: MPEG-1, MPEG-2 e MPEG-4 (o padrão MPEG-3 que seria usado nas aplicações de HDTV ⁶foi descartado pelo grupo). A tabela 5.1 faz uma comparação entre estes padrões, considerando a qualidade suportada e a taxa de compressão de vídeo endereçada (Rowe et al.,1994).

<i>Padrão MPEG</i>	<i>Qualidade suportada</i>	<i>Taxa de bits (MB/s)</i>
MPEG-1	Super-VHS	1.5 a 2.0
MPEG-2	TV e HDTV	2.0 a 6.0
MPEG-4	Vídeofone	0.0048 a 0.064

Tabela 5.1 - Comparação entre os padrões MPEG existentes

5.3.1.1 - Subsistema de Apresentação e Interação

O terminal através do qual o usuário tem acesso ao serviço deve ser capaz de receber, descompactar, decodificar e apresentar o fluxo contínuo de vídeo vindo do sistema de comunicação. Outra função do terminal é a de enviar comandos de interação ao servidor de vídeos remoto, de acordo com as instruções de um usuário humano, a fim de controlar a exibição do vídeo. O dispositivo que aparece na figura 5.3 ligado ao terminal é chamado de *set-top box (STB)* e é citado desde as idéias preliminares de concepção deste serviço como responsável pelo conjunto de requisitos funcionais ligados ao terminal. Várias são as possibilidades de implementação de um STB. Tradicionalmente, eles são implementados como dispositivos de hardware, no entanto, atualmente, com o crescimento do poder computacional disponível do lado do cliente, implementações híbridas ou puramente de software têm se tornado viáveis (Rowe et al.,1994).

5.3.1.2 - Subsistema de Comunicação

A rede pela qual o usuário tem acesso ao serviço é chamada de rede de acesso local (RAL). Ao contrário de conexões típicas usadas no ambiente tradicional de computação, caracterizadas por curtos períodos de duração e tráfego em rajadas, o esquema usado para a ligação do usuário ao provedor do serviço deve suportar conexões de algumas horas de duração, tráfego contínuo de dados e grande largura de banda. As alternativas de implementação mais viáveis do ponto de vista tecnológico e econômico são as abordagens que utilizam a rede pública de telefonia, devido à alta capilaridade alcançada por esta rede e, conseqüentemente, a possibilidade de grande cobertura e baixo custo do serviço. Entre as

⁵ DAVIC: Digital Audio Visual Council – associação sem fins lucrativos, contando com 222 empresas em 25 países, comprometida com a padronização da interoperabilidade fim-a-fim de aplicações audiovisuais interativas.

⁶ High Definition Television (Cecilio & Rodrigues, 1996a)

principais propostas existentes atualmente está a família de tecnologias ADSL (*Asymmetrical Subscriber Digital Line*), que oferecem tipicamente taxas entre 1.544 e 6 Mbps para *downstream* (do provedor para o usuário) e entre 128 e 384 Kbps no sentido inverso. Porém, essas taxas são alcançadas somente em um raio de distância fixo, geralmente em torno de 5,5 Kilômetros da fonte de transmissão.

5.3.1.3 - Subsistema de Transmissão e Armazenamento

Por fim, o sistema de armazenamento tem como responsabilidade cuidar do espaço necessário para armazenar a programação em formato comprimido, além da transmissão do fluxo de vídeo, quando requisitado para exibição. O servidor de vídeo deve ser capaz também de responder aos comandos de interação do usuário, a fim de prover a ele o controle da sessão de exibição do vídeo, implementando as funções de um videocassete virtual (iniciar, interromper, dar uma pausa, adiantar, voltar). Para atender a muitos usuários e oferecer uma grande variedade de informações, o sistema de armazenamento do servidor de vídeos deve possuir alta capacidade, além de suportar o acesso concorrente de vários usuários simultaneamente. Para o projeto de um servidor de vídeos, várias considerações devem ser tomadas:

- Mesmo comprimido a uma taxa de 100:1 um único filme comercial ainda ocupa cerca de 1.45 GB de espaço de armazenamento (sem o áudio correspondente). Supondo que o servidor possua 500 filmes disponíveis, seria preciso um espaço de armazenamento de aproximadamente 725 Gbytes.
- O número de sessões interativas que um servidor pode suportar é proporcional, entre outras coisas, à taxa de transferência do seu sistema de entrada/saída. Mesmo com tecnologias como SCSI-2 e FC-AL que podem alcançar taxas de 100 MB/s, um servidor só pode atender a um número de sessões simultâneas relativamente limitado para um serviço comercial de larga escala.
- Um fato conhecido através da experiência com outros serviços sob demanda (locadoras de vídeo, bibliotecas públicas) é que nem todos os itens disponíveis em um catálogo são igualmente populares. De acordo com a “lei de Zipf”(Chervenak, 1994), experimentalmente, quando existem N filmes disponíveis, o percentual de todas as requisições para o k-ésimo filme mais popular é de aproximadamente C/k , onde C é computado para normalizar a soma em 1.

$$C/1 + C/2 + C/3 + C/4 + \dots + C/N = 1$$

$$C = 1 / (1 + 1/2 + 1/3 + 1/4 + \dots + 1/N)$$

Com isso, o filme mais popular é sete vezes mais requisitado que o sétimo filme mais popular.

Com estas considerações, podemos identificar duas características desejáveis que devem ser contempladas de alguma forma por nossa arquitetura. São elas:

- (a) **Estrutura Hierárquica:** É extremamente custoso para o provedor que todos os vídeos que os seus usuários possam querer assistir estejam armazenados em seus servidores. Portanto, os servidores de um provedor devem armazenar apenas aqueles filmes mais populares entre o público alvo daquele provedor (por exemplo, 50 ou 100). Os outros filmes devem estar armazenados em um outro dispositivo que o provedor tenha acesso,

podendo ser um dispositivo de memória secundária (como por exemplo, uma fita), ou ainda um outro servidor pertencente a outro provedor.

(b) Flexibilidade: É previsível que durante a vida de um provedor, vários servidores venham a ser inseridos em seu domínio. Desta forma, a introdução de um novo servidor neste esquema não deve ser custosa a ponto de afetar a operação normal do sistema.

As exigências de flexibilidade impostas pelos subsistemas de armazenamento e comunicação terão um grande impacto tanto nos modelos de distribuição do serviço (como será visto na próxima subseção), quanto nos mecanismos de gerência do sistema. A fim de prover escalabilidade, adaptatividade e robustez, o mecanismo de gerência implementado deverá controlar de forma eficiente todas as relações e restrições envolvendo os agentes pertencentes a esse cenário - vídeos, servidores, provedores, terminais e centrais.

5.3.2 - Configurações de Distribuição do Serviço

Em qualquer sistema de distribuição de conteúdo de massa, uma das principais preocupações é como este conteúdo será distribuído. As duas principais alternativas de configuração são (Sampson et al., 1997):

1. Um sistema centralizado, com poucos servidores de altíssima capacidade de armazenamento e capacidade para atender a um número enorme de sessões simultâneas;
2. Um sistema descentralizado, com muitos servidores estrategicamente bem localizados.

A fim de caracterizar essas configurações de distribuição, deve-se fazer a distinção entre duas sub-redes presentes no contexto: a Rede de Acesso Local (*RAL*) e a Rede de Acesso Remoto (*RAR*). A *RAL* é a sub-rede na qual o terminal do usuário está conectado, enquanto que a *RAR* conecta servidores distribuídos geograficamente. Em um ambiente de VoD comercial, a *RAL* poderia ser um canal *ADSL*, ao qual o terminal do usuário teria acesso, e a *RAR* poderia ser uma rede ATM de alta velocidade, interligando os servidores de vídeo. Deste modo, podemos observar que a *RAL* se apresenta como uma alternativa de estrutura mais simples e lenta que a *RAR*, mas também mais acessível ao usuário final, o que é interessante em um sistema comercial. Já em um ambiente educacional, a *RAL* pode ser uma LAN pertencente a um departamento e a *RAR* uma WAN conectando departamentos ou universidades.

5.3.2.1 - Distribuição Centralizada

Em uma configuração centralizada (Figura 5.4), temos um número limitado de enormes servidores de vídeo (*video warehouses*) ligados às centrais telefônicas através de uma rede de alta velocidade (consideraremos aqui esta rede como sendo uma rede ATM). Neste cenário, a central telefônica possui tanto o equipamento de terminação da rede ATM, quanto *buffers* locais para converter o tráfego ATM em rajadas, para um fluxo contínuo de dados até o terminal do usuário. No futuro, com uma possível evolução e queda de preço da tecnologia ATM, o equipamento de terminação ATM poderá ser movido para a casa do usuário.

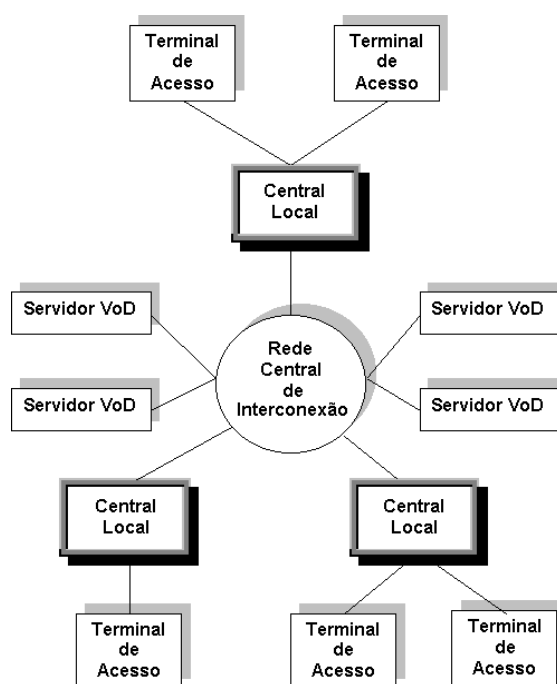


Figura 5.4 - Configuração de distribuição centralizada

Uma clara desvantagem da abordagem centralizada em um serviço inerentemente interativo como o que queremos modelar é que o comportamento de interação de cada usuário demanda trocas automáticas do conteúdo dos *buffers* (tanto os *buffers* locais, quanto os presentes nos nós da rede central), que vão interferir diretamente no tráfego da rede de interconexão. Neste cenário, o mercado de vídeo sob demanda seria dominado por grandes cadeias de comunicação com a participação direta das empresas de telefonia.

5.3.2.2 - Distribuição Descentralizada

Na abordagem descentralizada (figura 5.5), existem servidores menores, com capacidade de armazenamento de, por exemplo, 50 a 100 vídeos, ligados diretamente às centrais telefônicas locais. Desta forma, o usuário acessa diretamente os servidores locais através da rede de acesso local. Estes servidores podem ser *stand-alone* ou podem estar conectados via rede de alta velocidade a outros servidores presentes em outras centrais telefônicas.

Caso um usuário opte por um vídeo que não esteja presente em um servidor local, então este vídeo pode ser copiado via rede de alta velocidade de um servidor remoto qualquer, seguindo políticas de gerência estabelecidas para o serviço. A fim de minimizar o tráfego na rede que conecta as centrais telefônicas, uma política de escalonamento deve ser adotada, premiando os usuários que optem por fazer esta cópia em um horário de baixo tráfego na rede.

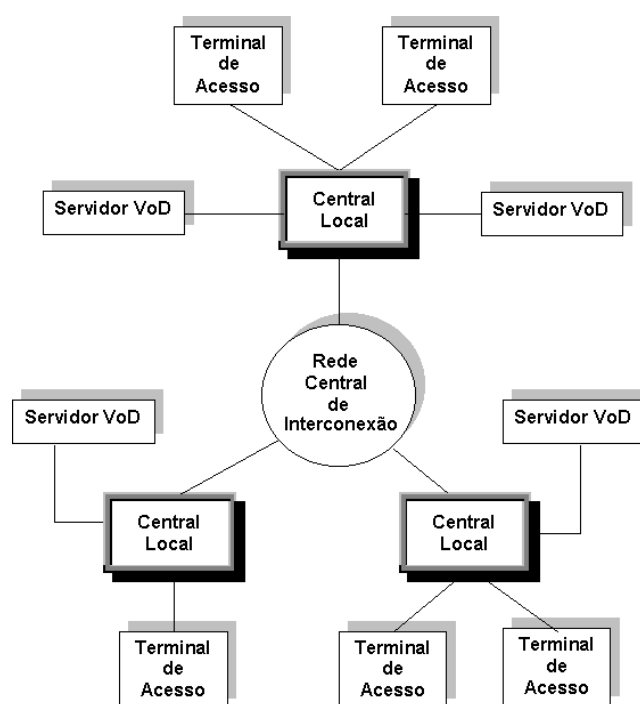


Figura 5.5 - Configuração de distribuição descentralizada

5.3.2.3 - Comparação Entre as Duas Abordagens

Além da já citada desvantagem da abordagem centralizada, no que diz respeito à influência que o comportamento interativo de usuários individuais tem sobre o tráfego da rede, acreditamos que existem várias outras vantagens da abordagem descentralizada. As principais são:

- As empresas de telefonia não têm necessariamente participação direta como provedoras do serviço de VoD, podendo participar apenas com o oferecimento das funcionalidades das centrais locais para outras empresas que serão, por sua vez, as provedoras do serviço para o usuário final;
- Os provedores podem começar com um serviço em pequena escala (servidores *stand-alone*), que crescerá de acordo com a demanda, possibilitando um investimento inicial mais baixo;
- Possibilidade de adequação a públicos locais, no sentido de prover uma programação (e também propagandas) direcionadas para um público alvo específico;
- Taxas de transmissão mais baixas;
- Maior separação dos recursos de aplicação e redes;
- Maior flexibilidade de configuração.

5.3.3 - Os Níveis de Gerência e de Sistema

Independentemente da configuração de distribuição adotada, fica claro que o subsistema de comunicação será considerado um serviço à parte, muito provavelmente oferecido por um provedor diferente daquele do serviço de VoD propriamente dito. Desta forma, do ponto de vista deste trabalho, o desenvolvimento de aplicações de vídeo sob demanda deve estar comprometido apenas com os subsistemas de Apresentação/Interação e

Armazenamento/Transmissão. A infra-estrutura de comunicação é vista como uma *caixa-preta* capaz de encapsular as funcionalidades requeridas. Além disso, o conjunto de responsabilidades atribuídas à interação entre os subsistemas abordados será distribuída entre dois níveis funcionais: **o nível de gerência e o nível de sistema**. Os três principais sistemas funcionais participantes dessa arquitetura - Terminal do usuário, Servidor de Gerência e Servidor de Sistema - receberão nessa seção a mesma nomenclatura que em (Eilley et al., 1994).

No nível de gerência, o terminal do usuário - chamado de *EUT (End-User Terminal)* - interage com um servidor funcional chamado de *AMS (Application Management System)*. Esse servidor é uma espécie de gerente dos agentes envolvidos no serviço (provedores, servidores, terminais, centrais e vídeos). Os usuários inicialmente se conectam a um AMS para escolher o serviço que vão usar e o vídeo que vão assistir. Para isso, o AMS se comunica com um banco de meta-dados que contém os catálogos de todos os servidores gerenciados por ele. Uma vez escolhido o vídeo, o AMS conecta o usuário ao servidor de vídeo correspondente que faz a distribuição do vídeo. O AMS é responsável por todas as funções de gerência dos servidores de vídeo, como controle de acesso, *marketing*, controle comercial e serviços de nomeação, além de saber responder a questões como que servidor pertence a que provedor, que servidor oferece qual serviço e que servidor armazena qual vídeo, além de outras coisas. O AMS também arquiva informação sobre todos os vídeos assistidos e registra os dados estatístico para propósitos futuros.

Uma vez escolhido, o vídeo a ser assistido, o EUT passa a interagir com outro servidor, chamado de *VoDS (Video on Demand Server)*. Juntos eles cooperam na realização de funções do chamado nível de sistema, como por exemplo sincronização e transporte de mídias contínuas, decodificação e apresentação de mídias em vários formatos e provimento das funções de interação do usuário, implementando as funcionalidades de videocassete virtual.

É importante, ainda, salientar que esta abordagem contribui diretamente para se conseguir uma arquitetura robusta no que diz respeito à escalabilidade e à flexibilidade. Algumas de suas principais vantagens em relação a termos um único tipo de servidor realizando ambas as tarefas são:

1. O AMS funciona como um gerente de provedores (e, conseqüentemente, dos seus VoDS), encapsulando em um único subsistema todos aqueles serviços não relativos à exibição do vídeo propriamente dita e que são comuns a todos os provedores, como serviços de concretização dos requisitos relacionais (que provedor possui qual servidor, que servidor armazena qual vídeo, que cliente contrata qual provedor), segurança e controle de acesso, *marketing*, entre outros.
2. Com o AMS funcionando como servidor no nível de gerência para vários VoDS, seria extremamente fácil a inclusão, por exemplo, de um novo VoDS no sistema.
3. Ao encapsular todas as funções do nível de gerência, o AMS livra o usuário final da preocupação de saber que VoDS armazena que vídeo e da preocupação de saber os parâmetros necessários para se conectar a cada servidor separadamente.

Por fim, é importante salientar como a existência desses dois níveis ressalta a adequação deste estudo de caso para a experimentação da metodologia proposta. O nível de gerência deve encapsular restrições complexas de relacionamento entre os diversos agentes que atuam no sistema e a análise e abstração de suas necessidades será a base para a construção de uma ontologia de gerência de vídeo sob demanda na próxima seção. O nível

de sistema, por outro lado, encapsula restrições complexas de problemas de projeto comuns aos sistemas multimídia distribuídos. Para endereçar esses requisitos de projeto, na seção 5.5, é utilizado um *framework* chamado *JMF (Java Media Framework)*. Desta forma, as duas próximas seções apresentam, respectivamente, exemplos de desenvolvimento para e com reuso no contexto de um mesmo sistema.

5.4 - Aplicação da metodologia ao Domínio de Estudo

A seção anterior discutiu de forma detalhada o domínio de Vídeo sob Demanda. Primeiramente, foram identificados os requisitos tecnológicos necessários para que o provimento do serviço possa ser realizado. Em segundo lugar, foram analisadas as influências que esses requisitos têm sobre os modelos de distribuição do serviço e sobre as alternativas de implementação. Finalmente, foi apresentada uma proposta de divisão funcional do serviço de VoD em níveis de gerência e sistema. Essa seção tem como objetivo especificar uma ontologia de VoD comprometida com esse nível de gerência. Uma vez especificada, essa ontologia será submetida à metodologia proposta nos capítulos anteriores, a fim de gerar uma infra-estrutura reutilizável desse domínio.

Ao fim desse processo de aquisição e análise de dados, foram identificados os principais agentes participantes desse cenário, as relações existentes entre eles, as pré-condições que devem ser cumpridas para que possam se relacionar e as principais questões de competência que devem ser respondidas. É importante salientar que, por se tratar de uma ontologia, a representação do domínio produzida deve ser a mais genérica possível, capturando somente os elementos comuns aos diversos membros de uma família de aplicações.

A Figura 5.6 apresenta um modelo em LINGO que identifica os *conceitos* e *relações* nesse cenário (Guizzardi & Gonçalves, 1998). Em seguida é apresentada a descrição da intenção de cada um dos conceitos. Algumas dessas descrições expõem a intenção das relações em que estão envolvidos, sendo que para aquelas relações em que isso não acontece, a descrição de suas intenções também é dada.

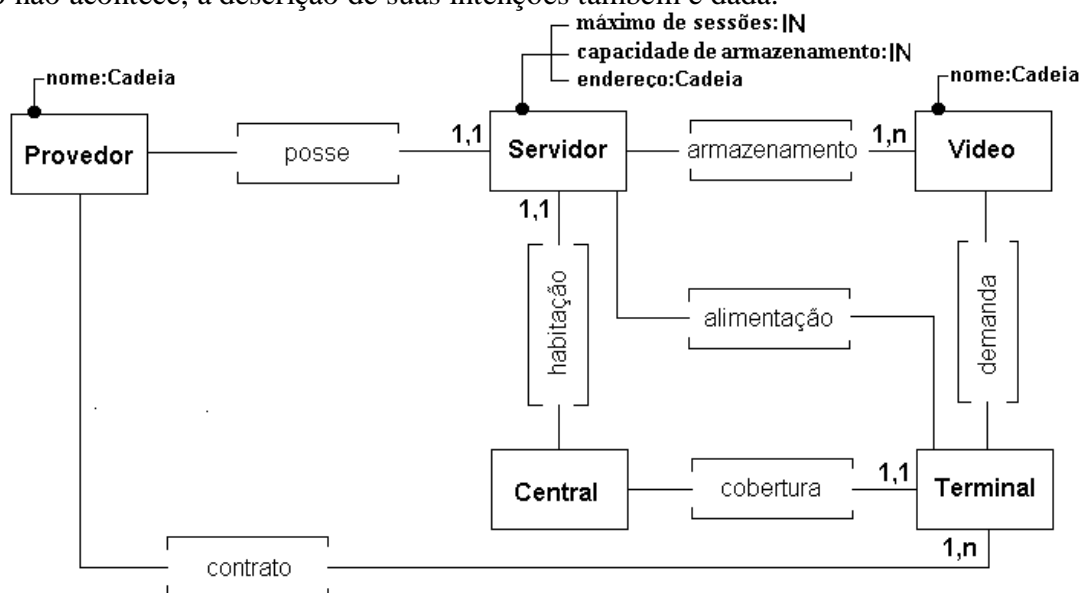


Figura 5.6 - Diagrama conceitos e relações do domínio de gerência de um serviço de VoD

5.4.1 - Descrição dos Conceitos

- **Terminal:** O *Terminal* representa o equipamento através do qual o usuário final tem acesso ao sistema. É através dele que o usuário escolhe o *Vídeo* que irá assistir e, uma vez escolhido, interage com a exibição desse *Vídeo*. Independentemente de como será materializado, um *Terminal* deve ser capaz de receber e apresentar o fluxo contínuo de *Vídeo* proveniente de sua comunicação com o *Servidor*, sendo compatível com o formato de codificação utilizado por ele. Se necessário, deve ser capaz de realizar as funções de decodificação e descompactação dos dados recebidos. Por fim, com o objetivo de prover as funcionalidades de um videocassete virtual (pausar, adiantar, retroceder, parar), ele deve ser capaz de interagir com este *Servidor*, enviando-lhe os resultados dos comandos de interação do usuário.
- **Servidor:** Armazena os *Vídeos* e os transmite aos *Terminais*, em um formato suportado por qualquer *Terminal* participante. Deve suportar o acesso aleatório a posições dos *Vídeos* armazenados, com o objetivo de responder aos comandos de interação de um ou mais *Terminais* simultaneamente. Cada *Servidor* é identificado através de um sistema de endereçamento, próprio da tecnologia de comunicação empregada. Além disso, cada *Servidor* suporta um número máximo de *Vídeos* armazenados e de sessões de transmissão. No entanto, é importante salientar que essa descrição não define como os *Vídeos* são armazenados ou transmitidos, nem tampouco quantos deles podem ser armazenados ou transmitidos simultaneamente, sendo essas decisões tomadas por cada uma das aplicações específicas.
- **Vídeo:** É, provavelmente, o principal conceito neste domínio. Um *Vídeo* diz respeito a uma entidade que identifica univocamente um título que compõe o catálogo de um *Provedor*. Um *Vídeo*, portanto, existe no escopo de um *Provedor*⁷. O conceito não referencia diretamente o material digitalizado, codificado em um formato específico e armazenado nos *Servidores*. Um mesmo *Vídeo* pode possuir várias cópias (padrões de magnetismo em um meio de armazenamento) em *Servidores* diferentes, desde que esses *Servidores* pertençam a um mesmo *Provedor*, ou de maneira mais formal

$$\forall v:V, s_1, s_2:S \text{ armazenamento}(v, s_1) \wedge \text{armazenamento}(v, s_2) \rightarrow \text{provedor}(s_1) = \text{provedor}(s_2)$$

- **Provedor:** o *Provedor* é uma entidade que possui *Servidores* e é contratada por *Terminais*, sendo responsável por manter os catálogos de *Vídeo* em seus *Servidores* e por gerenciar os usuários que os contratam.
- **Central:** Essa é a única entidade que aparece no modelo devido aos requisitos tecnológicos necessários para o oferecimento do serviço. Apesar de ser uma entidade cujo controle está fora do alcance dos provedores do serviço, essa entidade é modelada devido a sua importância do ponto de vista lógico (como será observado na descrição

⁷ Em uma modelagem conceitualmente realista, o conceito de *Vídeo* é definido de forma completamente dissociada do conceito de *Provedor*. Por esse motivo, uma instância de *Vídeo* (por exemplo “*E o vento levou...*”) existe independente de qualquer provedor em particular e a mesma para qualquer provedor que por ventura possa vir a possuí-la em seu catálogo. No entanto, nesse capítulo a modelagem desse domínio tem como principal propósito servir de estudo de caso para a metodologia proposta. Por esse motivo, nesse caso em particular, a correção conceitual é sacrificada em prol de uma oportunidade de exercitar a descrição e posterior derivação de um axioma de consolidação.

das relações em que participa). Sem a sua presença, o modelo se tornaria inadequadamente livre, no sentido de que qualquer *Servidor* poderia transmitir uma sessão de exibição para qualquer *Terminal* - o que não é verdade em muitos dos casos. Portanto, essa entidade desempenha o papel de uma central telefônica, central de TV a cabo (CATV) ou, ainda, uma rede local (LAN) no cenário educacional, associando *Servidores* e grupos de *Terminais* mutuamente excludentes.

A extensão desses conceitos é representada pelos seguintes conjuntos:

- (D1) P = Provedor
- (D2) S = Servidor
- (D3) C = Central
- (D4) V = Vídeo
- (D5) T = Terminal

5.4.2 - Descrição das Relações

- **cobertura**: Enfatiza a necessidade que deve existir, em um sistema de comunicação, entre *Terminal* e *Central* para que esse possa assistir a *Vídeos* transmitidos por um *Servidor*. Esse sistema de comunicação deve garantir a qualidade e confiabilidade do sinal de *Vídeo* recebido.
- **habitação**: Um *Servidor* deve possuir uma conexão com uma *Central* para que possa transmitir *Vídeos* para os *Terminais* cobertos por ela. O *Servidor* pode estar localizado fisicamente ou não na *Central*, desde que exista entre eles uma conexão que garanta que o sinal de *Vídeo* enviado seja de qualidade e confiabilidade suficientes para que possa ser interpretado pelos *Terminais*. A escolha de que *Servidor* habitará que *Central* e de que maneira, é resultante de decisões do ponto de vista de negócio realizada pelos *Provedores* que os possuem e, portanto, sua discussão foge ao escopo dessa definição.
- **alimentação**: Esta relação encapsula um conjunto de pré-condições que devem ser satisfeitas para que um *Terminal* possa ser alimentado por um *Servidor*, ou seja, para que uma sessão de exibição de *Vídeo* possa ser firmada entre duas instâncias desses conceitos. A primeira delas diz que, para um *Servidor* alimentar um *Terminal*, ele deve pertencer a um dos *Provedores* contratados por esse *Terminal*, ou seja,

$$\forall s:S, \forall t:T \text{ alimentação}(t,s) \rightarrow s \in (\text{Im}(\text{Im}(t,\text{contrato}),\text{posse}))$$

Além disso, o *Servidor* deve estar conectado à mesma *Central* que cobre o *Terminal* para que os requisitos de qualidade necessários ao oferecimento do serviço (ex. largura de banda disponível) possam ser endereçados. De maneira mais formal, temos:

$$\forall s:S, \forall t:T \text{ alimentação}(t,s) \rightarrow \exists c:C \text{ habitação}(s,c) \wedge \text{cobertura}(c,t)$$

ou de forma equivalente,

$$\forall s:S, \forall t:T \text{ alimentação}(t,s) \rightarrow s \in (\text{Im}(\text{Im}(t,\text{cobertura}),\text{habitação}))$$

Esses axiomas de pré-condições podem ser compostos em um único axioma ontológico que responde a seguinte questão de competência: *dado um Terminal, que Servidores são capazes de alimentá-lo?*

$$\forall t:T, s:S \text{ alimentação}(t,s) \leftrightarrow$$

$$s \in (\text{Im}(\text{Im}(t, \text{contrato}), \text{posse}) \cap \text{Im}(\text{Im}(t, \text{cobertura}), \text{habitação}))$$

Por fim, essa questão pode ser também vista do ponto de vista do Servidor: *dado um Servidor, que Terminais são alimentados por ele?*

$$\forall s:S, t:T \text{ alimentação}(s,t) \leftrightarrow \\ t \in (\text{Im}(\text{Im}(s, \text{posse}), \text{contrato}) \cap \text{Im}(\text{Im}(s, \text{habitação}), \text{cobertura}))$$

- **demanda:** Essa relação é definida em função de outras relações e é responsável por responder à principal questão de competência da ontologia: *Que Vídeos podem ser assistidos por um dado Terminal?* A resposta a essa questão é dada pelo axioma abaixo:

$$\forall t:T, v:V \text{ demanda}(t,v) \leftrightarrow v \in \text{Im}(\text{Im}(t, \text{alimentação}), \text{armazenamento})$$

Ou seja, o catálogo de Vídeos disponível a um Terminal é composto pelo conjunto de Vídeos armazenados nos Servidores que alimentam aquele Terminal.

A seguir, são apresentados, para todas as relações e propriedades, os axiomas de definição, e de restrição de suas cardinalidades:

(D6) posse = (Provedor, Servidor, posse(p,s))

(D7) contrato = (Provedor, Terminal, contrato(p,t))

(D8) armazenamento = (Servidor, Vídeo, armazenamento(s,v))

(D9) habitação = (Servidor, Central, habitação(s,c))

(D10) cobertura = (Central, Terminal, cobertura(c,t))

(AC1) $\forall s:S \# \text{Im}(s, \text{posse}) = 1$

(AC2) $\forall s:S \# \text{Im}(s, \text{habitação}) = 1$

(AC3) $\forall t:T \# \text{Im}(t, \text{contrato}) \geq 1$

(AC4) $\forall t:T \# \text{Im}(t, \text{cobertura}) = 1$

(AC5) $\forall v:V \# \text{Im}(v, \text{armazenamento}) \geq 1$

(AP1) $\forall s:S \exists! \underline{e}: \text{Cadeia endereço}(\underline{e}, s)$

(AP2) $\forall s:S \exists! \underline{c}: \text{IN capacidade de armazenamento}(\underline{c}, s)$

(AP3) $\forall s:S \exists! \underline{m}: \text{IN máximo de sessões}(\underline{m}, s)$

(AP4) $\forall v:V \exists! \underline{n}: \text{Cadeia nome}(\underline{n}, v)$

(AP5) $\forall p:P \exists! \underline{n}: \text{Cadeia nome}(\underline{n}, p)$

Ao tornar explícito o conhecimento inerente a um domínio de interesse, uma ontologia permite a compreensão e a comunicação a respeito desse domínio. Por possuir uma descrição rigorosa, feita em uma linguagem matemática, seus axiomas podem ser formalmente simulados e validados. Além disso, por se tratar de uma estrutura de representação em um nível de conhecimento, uma ontologia se torna um componente altamente confiável e com grande potencial de reutilização. Por fim, através de uma metodologia sistemática de projeto, a partir dela, uma infra-estrutura de domínio pode ser gerada.

Nas seções seguintes, os axiomas que definem restrições e que derivam conhecimento nesse modelo de domínio são formalmente especificados. Em seguida, através da aplicação do conjunto de diretivas, regras de transformação e padrões de projeto anteriormente discutidos, um *framework* de gerência de Vídeo sob Demanda é gerado a

partir da ontologia. Os conceitos e relações, com suas respectivas características e restrições de cardinalidade, são mapeados em classes, relacionamentos, atributos e parâmetros embutidos nos construtores das classes adequadas. Além disso, os axiomas ontológicos e de consolidação derivam, respectivamente, invariantes e pré-condições, incorporados aos métodos do *framework*. O resultado desse processo é mostrado na figura 5.7.

5.4.3-Axiomas ontológicos

(AO1) $\forall s:S, t:T$ alimentação(s,t) \leftrightarrow
 $t \in (\text{Im}(\text{Im}(s,\text{posse}),\text{contrato}) \cap \text{Im}(\text{Im}(s,\text{habitação}),\text{cobertura}))$

(AO2) $\forall t:T, s:S$ alimentação(t,s) \leftrightarrow
 $s \in (\text{Im}(\text{Im}(t,\text{contrato}),\text{posse}) \cap \text{Im}(\text{Im}(t,\text{cobertura}), \text{habitação}))$

(AO3) $\forall t:T, v:V$ demanda(t,v) $\leftrightarrow v \in \text{Im}(\text{Im}(t,\text{alimentação}),\text{armazenamento})$

A seguir, as regras de transformação definidas no capítulo anterior são aplicadas a esses axiomas, a fim de promover a geração dos métodos do *framework* que responderão as questões de competência levantadas anteriormente.

(AO1) $\forall s:S, t:T$ alimentação(s,t) \leftrightarrow

$t \in (\text{Im}(\text{Im}(s,\text{posse}),\text{contrato}) \cap \text{Im}(\text{Im}(s,\text{habitação}),\text{cobertura}))$

1. alimentação(s,t) $\equiv (\text{Im}(\text{Im}(s,\text{posse}),\text{contrato}) \cap \text{Im}(\text{Im}(s,\text{habitação}),\text{cobertura}))$ A01, T0
2. s.alimentacao():Set $\equiv (\text{Im}(s.\text{posse}(),\text{contrato}) \cap \text{Im}(s.\text{habitação}(),\text{cobertura}))$ 1, T2
3. s.alimentacao():Set $\equiv \text{Set.Im}(s.\text{posse}(),"contrato") \cap \text{Set.Im}(s.\text{habitação}(),"cobertura")$ 2, T5
4. public class Servidor 3, T7

```

{
    public Set alimentacao()
    {
        Set aux = Set.Im(this.posse(),"contrato");
        return aux.intersection(Set.Im(this.habitação(),"cobertura"));
    }
}

```

(AO2) $\forall t:T, s:S$ alimentação(t,s) \leftrightarrow

$s \in (\text{Im}(\text{Im}(t,\text{contrato}),\text{posse}) \cap \text{Im}(\text{Im}(t,\text{cobertura}),\text{habitação}))$

1. alimentação(t,s) $\equiv (\text{Im}(\text{Im}(t,\text{contrato}),\text{posse}) \cap \text{Im}(\text{Im}(t,\text{cobertura}), \text{habitação}))$ A01, T0
2. t.alimentacao():Set $\equiv (\text{Im}(t.\text{contrato}(),\text{posse}) \cap \text{Im}(t.\text{cobertura}(),\text{habitação}))$ 1, T2
3. t.alimentacao():Set $\equiv \text{Set.Im}(t.\text{contrato}(),"posse") \cap \text{Set.Im}(t.\text{cobertura}(),"habitacao")$ 2, T5
4. public class Terminal 3, T7

```

{
    public Set alimentacao()

```

```

    {
        Set aux = Set.Im(this.contrato(),"posse");
        return aux.intersection(Set.Im(this.cobertura(),"habitacao"));
    }
}

```

(AO3) $\forall t:T, v:V \text{ demanda}(t,v) \leftrightarrow v \in \text{Im}(\text{Im}(t,\text{alimentação}),\text{armazenamento})$

1. $\text{demanda}(t,v) \equiv \text{Im}(\text{Im}(t,\text{alimentação}),\text{armazenamento})$ A01, T0
2. $t.\text{demanda}():\text{Set} \equiv \text{Im}(t.\text{alimentação}(),\text{armazenamento})$ 1, T2
3. $t.\text{demanda}():\text{Set} \equiv \text{Set.Im}(t.\text{alimentação}(),"\text{armazenamento}")$ 2, T5
4. public class Terminal 3, T7

```

    {
        public Set demanda()
        {
            return Set.Im(this.alimentação(),"armazenamento");
        }
    }

```

5.4.4-Axioma de consolidação

(AR1) $\forall v:V, s_1,s_2:S \text{ armazenamento}(v, s_1) \wedge \text{armazenamento}(v, s_1) \rightarrow \text{provedor}(s_1) = \text{provedor}(s_2)$

Aplicando a esse axioma o padrão *Pré-Condição* definido no capítulo anterior, o seguinte código é gerado:

```

public class Video
{
    public boolean setArmazenamento (S s2)
    {
        boolean result = false;
        if (result = checkArmazenamento(S2))
        {
            armazenamento.add(S2);
            S2.setArmazenamento(this);
        }
        return ok;
    }

    public boolean checkArmazenamento(Servidor s2)
    {
        Set aux = this.armazenamento();
        Servidor s1 = Set.any(aux);
        return s1.equals(s2);
    }
}

```

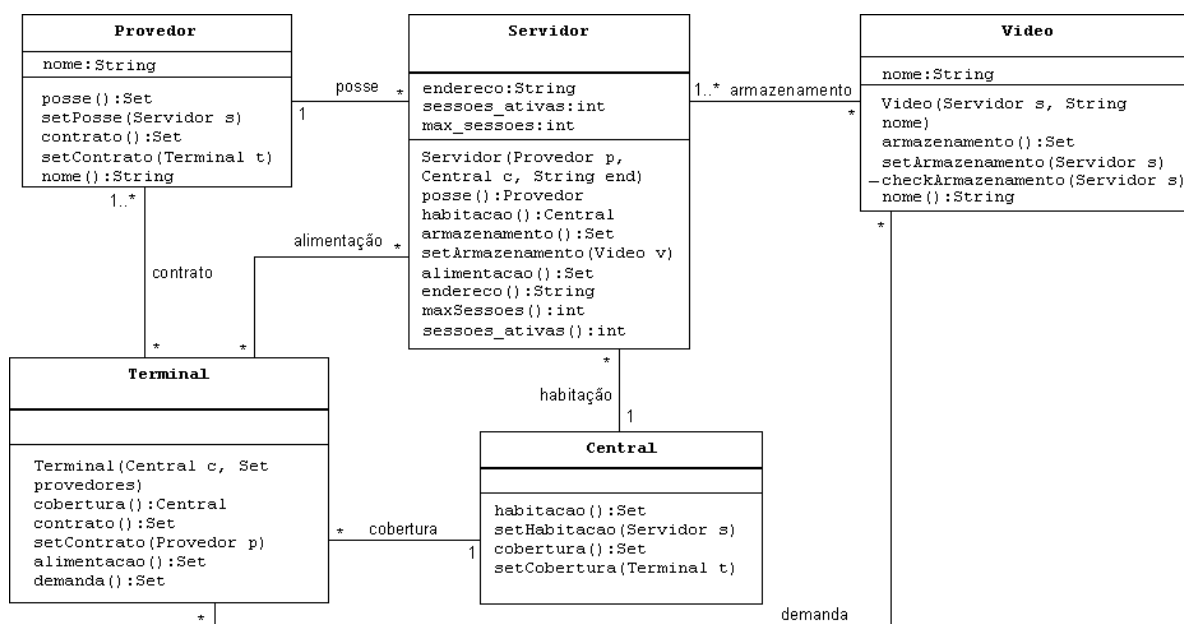


Figura 5.7 - Framework de gerência de Vídeo sob Demanda derivado a partir de uma ontologia de domínio

5.5 - Hipervisão: de uma ontologia de domínio a aplicação de vídeo sob demanda

Como citado anteriormente (5.2.1), o sistema Hipervisão foi desenvolvido no contexto do projeto DAMD, com o objetivo de experimentar os diversos aspectos da metodologia proposta. O sistema tem como requisito geral criar uma estrutura interdisciplinar, conectando vários departamentos acadêmicos, possivelmente de várias instituições em diferentes países, a fim de oferecer aos seus usuários finais (alunos) o acesso a vídeos digitalizados. Para isso, foi instanciada a arquitetura conceitual, sugerida pela ontologia da seção anterior, em um contexto educacional e, em seguida, foram desenvolvidas as fases pertencentes ao nível de aplicação apontadas pelo modelo de processo concebido na seção 5.2 (análise, projeto arquitetural, projeto detalhado e implementação). Desta forma, o *Hipervisão* será mostrado aqui como um estudo de caso da utilização desse processo.

O objetivo dessa seção não é mostrar toda a especificação do sistema construído (mesmo porque isto ocuparia bem mais que uma seção), mas sim salientar pontos importantes na transição de uma perspectiva de domínio para a de uma aplicação, no que diz respeito ao nível de gerência, bem como ilustrar o desenvolvimento orientado a objetos de uma aplicação multimídia distribuída, abordando aspectos do nível de sistema.

Apesar da ontologia desenvolvida ser genérica e, portanto, independente da configuração de distribuição adotada, daqui em diante a discussão será centrada na distribuição descentralizada. Isso é devido ao fato dessa ter sido a opção escolhida no projeto Hipervisão, pela sua adequação ao propósito do sistema e ao cenário de implementação.

A figura 5.8 ilustra a disposição dos elementos envolvidos nessa arquitetura. Os terminais do usuário são agrupados em centrais (Redes de Domínio Local), que são, por sua vez, conectadas umas com as outras através de uma Rede de Acesso Remoto (RAR).

Independentemente das escolhas tecnológicas para a materialização de cada um desses elementos, as relações e restrições em que estão envolvidos são da mesma forma válidas.

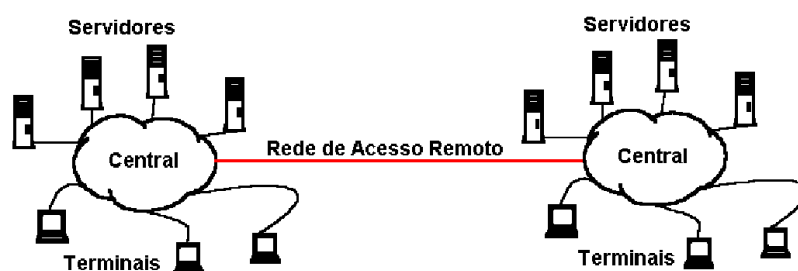


Figura 5.8 – arquitetura conceitual de vídeo sob demanda

Abaixo são exemplificados dois diferentes cenários de concretização dessa arquitetura:

1. **Cenário comercial:** Numa implementação comercial, os terminais são materializados como televisões e *set-top boxes* (STB), e a central é uma central telefônica conectando através de um enlace ADSL usuários dentro de um mesmo círculo de vizinhança (distância máxima exigida pela tecnologia ADSL para que se alcance a taxa de dados necessária para transmissão de vídeo digital). As centrais telefônicas são conectadas por um *backbone* ATM de alta velocidade (metropolitano ou mesmo nacional). Neste cenário, provedores aparecem como empresas de telecomunicações e de entretenimento (figura 5.9).

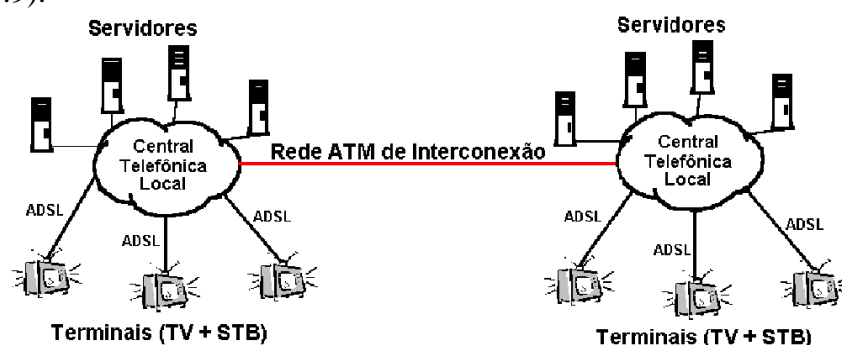


Figura 5.9 – Implementação de um serviço de vídeo sob demanda em um cenário comercial

2. **Cenário educacional:** Por um outro lado, em um cenário educacional, no qual os terminais são computadores ligados a uma rede local departamental. Provedores podem ser colegiados de cursos acadêmicos, departamentos ou centros. A RAR pode ser um *backbone* institucional conectando os vários departamentos de uma mesma instituição ou até mesmo a Internet conectando centros educacionais ao redor do mundo (figura 5.10).

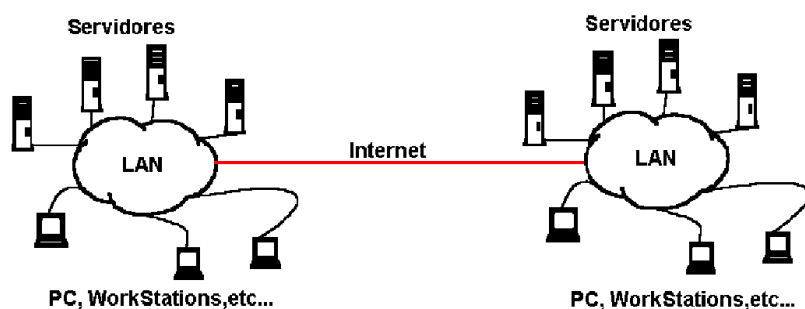


Figura 5.10 – Implementação de um serviço de vídeo sob demanda em um cenário educacional

O projeto *Hipervisão*, por se tratar de um projeto acadêmico, o último cenário foi o escolhido para ser implementado. Portanto, as centrais neste caso, são redes locais TCP/IP conectadas umas as outras através da Internet. A figura 5.11 exemplifica este cenário. Neste exemplo, as centrais C_1 e C_2 representam, respectivamente, os departamentos de Computação e Engenharia Elétrica de uma mesma universidade. Neste caso, P_2 é o curso de mestrado em Ciência da Computação e P_3 o curso de graduação em Engenharia Elétrica. O provedor P_1 é do curso de especialização em Redes de Computadores, oferecido de maneira conjunta pelos dois departamentos. Suponha que um aluno X do Departamento de Informática cursa o mestrado em ciência da computação e o curso de especialização em Redes de Computadores, X contrata os provedores P_1 e P_2 e tem acesso ao sistema se conectando à central C_1 .

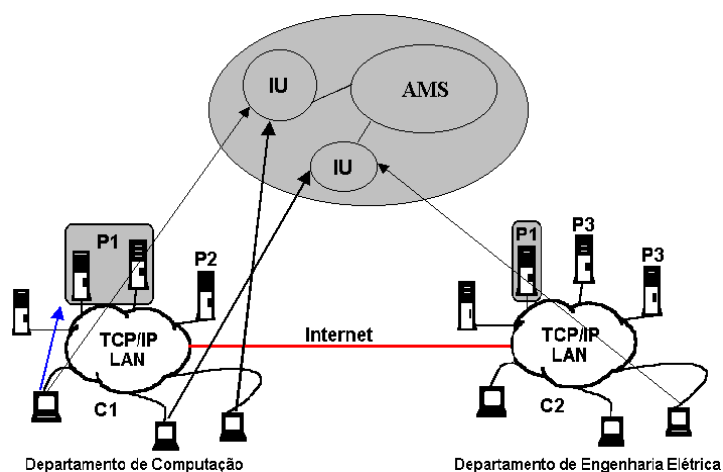


Figura 5.11 – Arquitetura do sistema Hipervisão

Este sistema de vídeo sob demanda possui três tipos de usuário: o administrador do AMS⁸, os administradores de cada um dos provedores (ex. P_1) e o usuário final do sistema (EUTUser), esse último se dividindo em *ParentUser* - que é o usuário responsável pelo terminal - e *ChildUser*. Um usuário do tipo *ChildUser* está sempre ligado (e é de alguma forma dependente) do primeiro. Esta distinção é feita para que restrições de gênero de vídeo (erótico, guerra, entre outros) possam ser impostas a usuários *ChildUser* por usuários

⁸ A seção 5.3.3 apresenta uma descrição detalhada dos papéis do AMS e do EUT

do tipo *ParentUser*. Na figura 5.12, são mostradas as principais funcionalidades do sistema oferecidas a estes usuários (casos de uso).

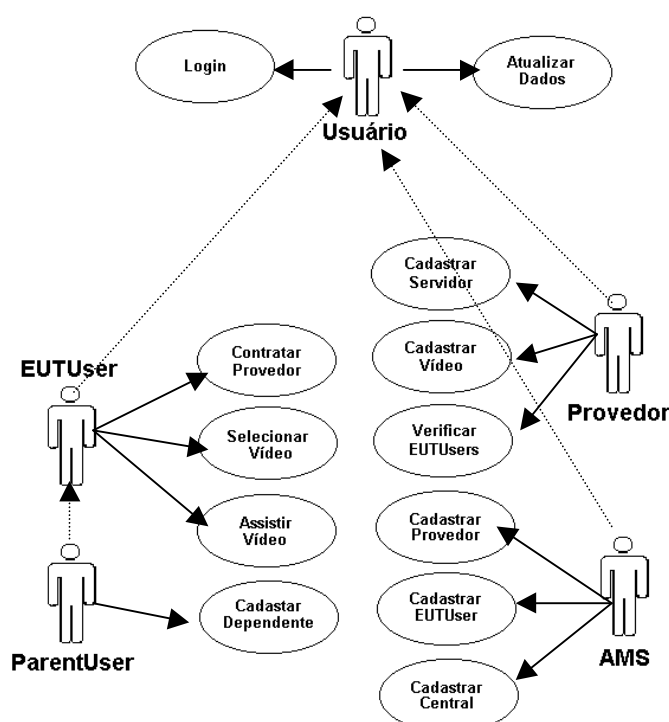


Figura 5.12 - Casos de Uso identificados na fase de análise da aplicação específica - Hipervisão

Como pode ser observado, existem dois casos de uso que são compartilhados por todos os usuários do sistema. O primeiro diz respeito ao processo de autenticação a que todos devem ser submetidos para terem acesso ao sistema (caso de uso *login*). Se este processo não for bem sucedido, uma mensagem de erro é retornada pelo sistema, caso contrário a entrada do usuário é permitida (e registrada) e a interface do ambiente é gerada, oferecendo as funcionalidades adequadas ao tipo do usuário.

O outro caso de uso comum aos diversos tipos de usuário é *Atualizar Dados*. Como o próprio nome já diz, através dele os usuários têm acesso aos seus dados, ou os dados da instituição que representam, como é o caso de provedor e AMS.

Os demais casos de uso são específicos a um dos tipos de usuário e suas descrições são sucintamente apresentadas a seguir, agrupadas pelos tipos de usuário. A única exceção é *Assistir Vídeo*, que pertence ao nível de sistema e é apresentado na seção 5.5.2.

a) AMS:

- **Cadastrar Provedor:** Para fazer parte do sistema Hipervisão, os provedores devem ser cadastrados no AMS. O sistema deve conhecer os dados do provedor (nome, localização e a instituição a que pertence), bem como de seu administrador (nome, email, telefone e senha de acesso ao sistema). O administrador do AMS também deve ser capaz de verificar os provedores cadastrados e seus respectivos servidores.
- **Cadastrar EUTUsers** Da mesma forma, para poderem utilizar o sistema, contratando provedores e, conseqüentemente, tendo acesso ao seus catálogos de

vídeo, os usuários finais devem possuir um cadastro no AMS. Os dados necessários para o cadastro são nome, telefone, data de nascimento, email, login e senha. Na verdade, o administrador do AMS cadastra apenas o usuário responsável pelo terminal (*ParentUser*), sendo ele responsável por cadastrar seus dependentes. No momento do cadastro, os usuários são associados à central através da qual terão acesso ao sistema. O AMS deve, também, ser capaz de verificar os usuários responsáveis cadastrados com seus respectivos dependentes.

- **Cadastrar Central:** É fundamental que, inicialmente, existam centrais cadastradas no sistema. São para essas centrais que os provedores alocam servidores de vídeo para que possam oferecer seus serviços. É também através delas que os usuários finais têm acesso ao sistema. Uma central também possui um nome, uma localização e pertence a uma instituição. Possui também um administrador cadastrado, cujos dados são idênticos ao dos administradores dos provedores.

b) Provedor:

- **Cadastrar Servidor:** O administrador de um provedor deve ser capaz de cadastrar seus servidores de vídeo, associando-os a uma das centrais cadastradas no sistema. Cada servidor deve possuir um endereço IP único. Deve também ser configurado para aceitar um número máximo de sessões de exibição simultânea, a fim de não degradar a qualidade mínima exigida pelo serviço. O administrador do provedor, deve ser capaz de verificar todos os seus servidores cadastrados, a quais centrais estão ligados e vídeos que cada um armazena.
- **Cadastrar Vídeo:** Os vídeos devem ser cadastrados, contendo nome, duração, descrição, gênero (terror, comédia, ficção, entre outros), diretor, elenco, nacionalidade e ano. Além disso, a cada vídeo é associada uma imagem (possivelmente de alguma cena do vídeo), para que possa compor a apresentação do vídeo vista pelos usuários finais. Como mencionado na seção 5.3.1, as mídias contínuas como áudio e vídeo, demandam dos servidores altos requisitos de capacidade de armazenamento. A fim de minimizar estes requisitos, bem como aumentar as possibilidades de associação dessas mídias, os arquivos digitalizados de vídeo, áudio e legenda (entre outras possíveis mídias no futuro) devem ser cadastrados separadamente. Para cada vídeo é inicialmente cadastrado o seu arquivo de mídia digitalizado, seu formato de codificação e as dimensões recomendadas para a tela de exibição. Uma vez feito isso, é possível cadastrar para um mesmo vídeo várias opções de áudio e legenda. Da mesma forma, para cada uma dessas opções deve ser cadastrado o arquivo de mídia digitalizado com o respectivo formato de codificação e uma descrição sucinta. Finalmente, um vídeo pode ser cadastrado em vários servidores, desde que pertençam ao mesmo provedor ($\forall v:V, s_1, s_2: S \text{ armazenamento}(v, s_1) \wedge \text{armazenamento}(v, s_2) \rightarrow \text{Im}(s_1, \text{posse}) = \text{Im}(s_2, \text{posse})$).
- **Verificar EUTUsers:** O administrador de um provedor p_1 deve ser capaz de conhecer todos os usuários que o contratam ($\text{Im}(p_1, \text{contrato})$), bem como seus respectivos dependentes.

c) **Usuário Final (EUTUser):**

- **Contratar provedor:** Um usuário final e_1 deve ser capaz de verificar seus provedores contratados, além de verificar outros provedores existentes com os quais não possui um contrato no momento ($\sim \text{Im}(\text{terminal}(e_1), \text{contrato})$)⁹.
- **Cadastrar Dependente:** O usuário responsável pelo terminal, deve ser capaz de cadastrar seus dependentes, assim como impor restrições de gênero a cada um deles. Por exemplo, um responsável pode querer que um de seus dependentes não assista filmes cujo gênero seja erótico ou guerra.
- **Selecionar Vídeo:** No caso de uma distribuição centralizada, o conjunto resultante da relação definida no axioma **(A03)** é igual ao conjunto abaixo:

$$(\forall t:T, \forall v:V \text{ demanda}(\text{terminal}(e_1), v) \leftrightarrow v \in \text{Im}(\text{Im}(\text{Im}(\text{terminal}(e_1), \text{contrato}), \text{posse}), \text{armazenamento}))$$

ou seja, todos os vídeos armazenados em todos os servidores pertencentes aos provedores que um usuário contrata. Por outro lado, em uma configuração de distribuição descentralizada - como é o caso dessa aplicação - é importante que essa distinção seja ressaltada, visto que o primeiro conjunto (A03) pode ser um subconjunto do segundo. Nesse caso, cada provedor que um terminal contrata pode possuir servidores espalhados em várias centrais. Desta forma, o catálogo do terminal não é composto somente pelos vídeos armazenados nos servidores que o alimentam $\text{Im}(\text{Im}(\text{terminal}(e_1), \text{alimentação}), \text{armazenamento})$, mesmo que este último seja o conjunto de vídeos que o usuário pode efetivamente assistir no momento em que desejar, ou como poderia ser chamado, o seu catálogo *on-line*. Se o vídeo escolhido pelo terminal não está em seu catálogo *on-line*, então ele precisa ser transportado do servidor em que está armazenado para um servidor conectado à mesma central desse terminal ($s \in \text{Im}(\text{Im}(\text{terminal}(e_1), \text{cobertura}), \text{habitação})$).

O usuário final deve, portanto, ser capaz de visualizar e selecionar para exibição vídeos pertencentes em seus catálogos *on-line* e *off-line*. Quando um vídeo é selecionado, a fronteira entre os níveis de gerência e sistema é transposta, tendo início o caso de uso *Assistir Vídeo*.

Por fim, o catálogo de um usuários do tipo *ChildUser* é o subconjunto do respectivo catálogo de seu responsável, em que nenhum dos vídeos é classificado em um dos gêneros restringidos para aquele usuário.

A partir da requisitos levantados na atividade de análise dessa aplicação específica, é identificada a adequação do reuso do *framework* de gerência de VoD desenvolvido na seção 5.4. Porém, para que os requisitos específicos da aplicação (documentados nos casos de uso) possam ser satisfeitos, é necessário que serviços adicionais sejam derivados e que novas classes seja incorporadas ao *framework* original. Além disso, esta fase pode gerar novas questões de competência específicas da aplicação, agregando novos axiomas, e conseqüentemente impulsionando tanto a evolução do framework original quanto a criação de uma ontologia de aplicação. O *framework* derivado nesse processo se comporta como

⁹ O símbolo \sim nesse caso representa a operação de complemento de um conjunto e não a negação lógica

um corpo de conhecimento capaz de responder às questões de competência tanto do domínio quanto da aplicação.

A seção 5.5.1 discute os aspectos mais importantes desse processo de especialização. As seções 5.5.2 e 5.5.3 discutem, respectivamente, aspectos relativos ao projeto e implementação dos casos de uso dos níveis de gerência e sistema.

5.5.1 - Especialização do framework de gerência de VoD

Nesta atividade, subclasses específicas da aplicação são derivadas a partir das classes originais do framework através do uso de herança. A fim de eliminar os efeitos colaterais decorrentes da má utilização dessa técnica - como quebra de encapsulamento e sobrescrita inadequada de métodos - as interfaces contratuais das superclasses são declaradas de forma a serem impassíveis de serem sobrescritas, se comportando assim como um verdadeiro *framework* caixa-preta (*black-box framework*).

No caso geral, por uma convenção aqui adotada, as classes derivadas possuem o mesmo nome de suas superclasses precedido pelo sinal `_`, sendo a exceção os casos em que um nome específico adiciona legibilidade e clareza ao modelo.

A seguir cada uma das partes do modelo derivado é discutida em maiores detalhes.

a) Central e Provedor

Como pode ser observado, as classes derivadas adicionam métodos, atributos e relacionamentos com objetos próprios da aplicação. Na figura 5.13, por exemplo, as classes derivadas de *Central* e *Provedor* se relacionam com um objeto *Adm*, que representa seus administradores.



Figura 5.13 - Especialização das classes *Provedor* e *Central*

b) Servidor e Vídeo

Como descrito no caso de uso Cadastrar Vídeo, as mídias contínuas áudio e vídeo devem ser modeladas e implementadas como objetos separados. Na figura 5.14, é apresentada a solução adotada. A classe derivada *_Video* se relaciona com um objeto *MidiaPrimaria*, que representa o material de vídeo digitalizado em algum formato suportado (MPEG, AVI, H.263, entre outros), e pode, opcionalmente, se relacionar com um ou mais objetos do tipo *MidiaSecundaria* (áudio ou legenda). Dessa forma, um mesmo vídeo, por exemplo, *A Lista de Schindler*, pode possuir várias opções de áudio (ex. inglês e espanhol) e de legenda (ex. português e holandês).

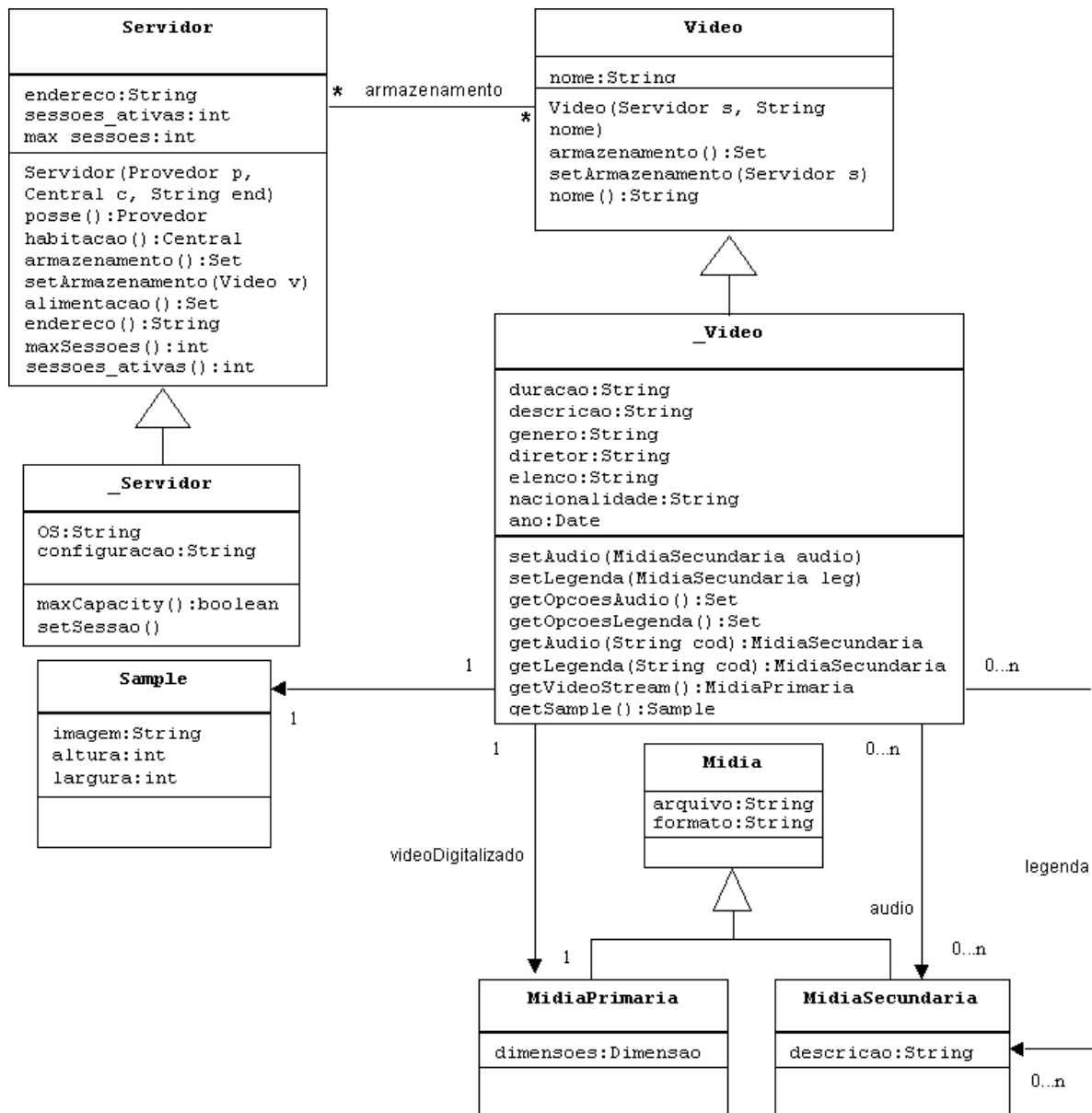


Figura 5.14 - Especialização das classes *Servidor* e *Vídeo*

Como um mesmo vídeo pode ser armazenado em vários servidores, a aplicação faz uso das funcionalidades de controle de sessões de exibição, implementadas pela classe *_Servidor*, com o objetivo de promover o balanceamento de carga entre eles.

c) Terminal

À classe derivada *_Terminal*, podem estar indiretamente associados vários usuários finais (EUTUser). Um único usuário ParentUser é responsável pelo terminal. Esse usuário, no entanto, pode possuir vários dependentes, aos quais pode impor restrições de gênero. Desta forma, o conjunto de vídeos que um ChildUser pode assistir pode ser um subconjunto daquele definido pela relação *demand* existente entre Terminal e Video. Isso vale tanto para os catálogos *on-line* quanto *off-line*.

Deste modo, a principal questão de competência da ontologia dá origem a uma outra no nível de aplicação: *Dado um EUTUser, qual são seus catálogos on-line e off-line?* Para responder a essa questão são derivados os seguintes axiomas:

W: _Terminal
X:ParenteUser
Y:ChildUser

- (A04)** $w:W, \forall v:V \text{ catalogo}(w,v) \leftrightarrow v \in (\text{Im}(\text{Im}(w,\text{alimentação}) / \text{Im}(\text{Im}(\text{Im}(w,\text{contrato}),\text{posse}),\text{armazenamento}))$
- (A05)** $x:X, \forall v:V \text{ demanda}(x,v) \leftrightarrow v \in \text{Im}(\text{terminal}(x),\text{demanda})$
- (A06)** $x:X, \forall v:V \text{ catalogo}(x,v) \leftrightarrow v \in \text{Im}(\text{terminal}(x),\text{catalogo})$
- (A07)** $\forall y:Y, \forall v:V \text{ demanda}(x,v) \rightarrow v \in _Video$
- (A08)** $\forall y:Y, \forall v:V \text{ demanda}(y,v) \leftrightarrow (v \in \text{Im}(\text{responsavel}(x),\text{demanda})) \wedge (\text{genero}(v) \notin \text{restricoes}(y))$
- (A09)** $\forall y:Y, \forall v:V \text{ catalogo}(y,v) \leftrightarrow (v \in \text{Im}(\text{responsavel}(x),\text{catalogo})) \wedge (\text{genero}(v) \notin \text{restricoes}(y))$

Como pode ser observado, o formalismo definido no Capítulo 4 e usado para descrever o conhecimento do domínio capturado pela ontologia, também pode ser aplicado para formalizar axiomas de derivação e consolidação pertencentes ao nível de aplicação. Da mesma forma, as regras de transição podem ser aplicadas para derivar as invariantes e pré-condições que devem estar presentes no diagrama de classes da aplicação derivado a partir do framework. A figura 5.15 mostra o diagrama de classes correspondente para os conceitos envolvidos nos axiomas acima descritos.

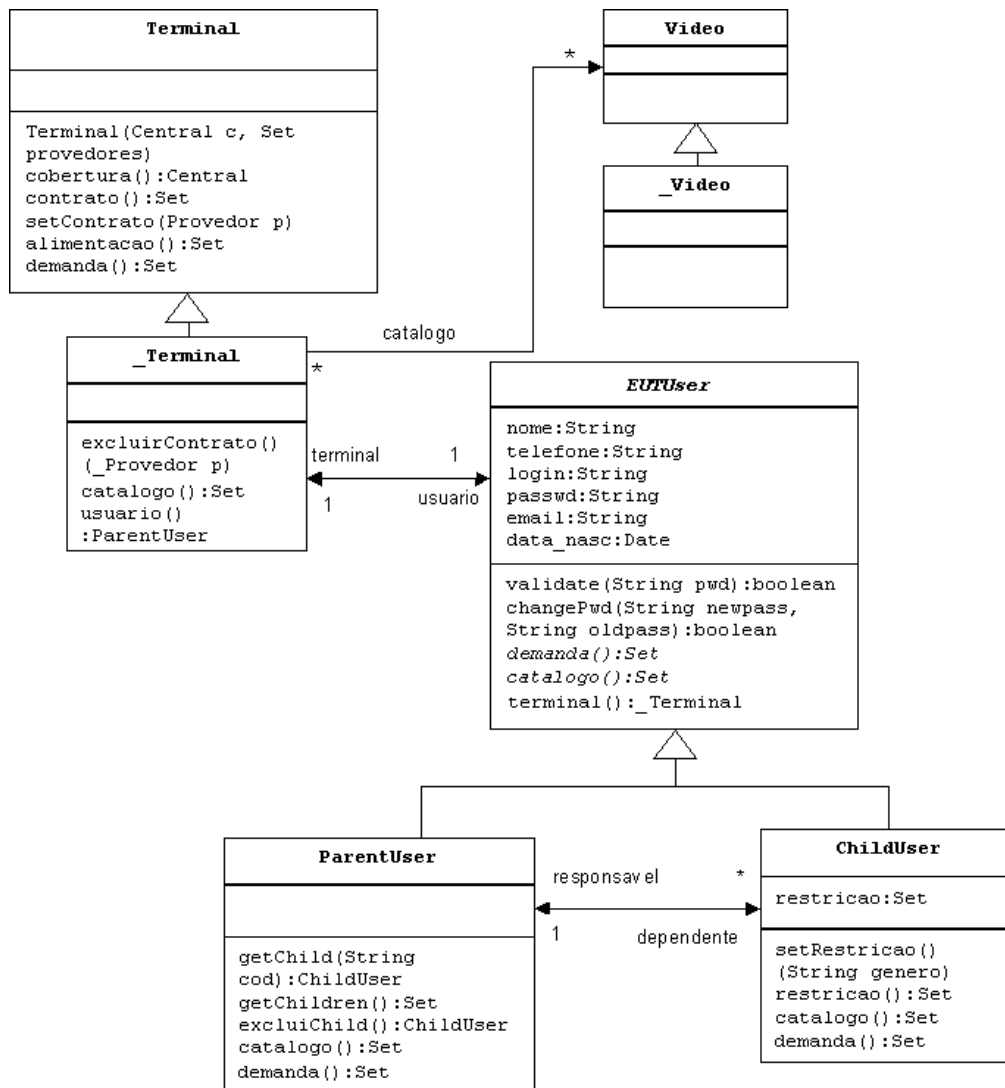


Figura 5.15 - Especialização da classe *Terminal*

5.5.2 - Projeto e Implementação do nível de gerência

Todas as decisões de projeto e implementação tomadas - como a escolha da linguagem de programação e dos modelos de interface e protocolos, bem como a definição da arquitetura de software - têm como objetivo atingir os requisitos não-funcionais de escalabilidade, flexibilidade e adaptatividade que vêm sendo discutidos ao longo desse trabalho. A figura 5.16 mostra as soluções adotadas para implementação das funções do nível de gerência nas camadas de interface, lógica do negócio e persistência.

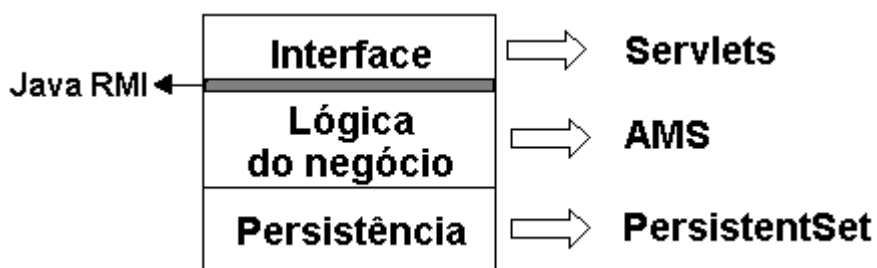


Figura 5.16 - Desenvolvimento em camadas do nível de gerência no *Hipervisão*

Nesse trabalho, como é ilustrado pela figura 5.11, a camada de interface é implementada através de diversos servidores de interface (IU) distribuídos. Os usuários têm acesso ao sistema se conectando através de um *browser* a um desses servidores. A escolha de fornecer uma interface via WWW deve-se ao fato de ser uma alternativa padronizada, aberta, bem difundida e que permite a utilização de estações clientes com baixa capacidade de memória e baixo poder de processamento. Desse modo, a estação cliente cuida apenas dos aspectos relativos à apresentação do conteúdo e interação com o usuário final, enquanto que todo o processamento da interface - como controle de sessão, geração dinâmica do conteúdo e interação com as camadas de lógica do negócio e persistência - é feito pelos servidores (IU). Além disso, é extremamente interessante que tanto a navegação do usuário, quanto a escolha de como a interface é apresentada a ele, sejam controladas pelos AMS, possibilitando um maior grau de personalização do serviço.

Para a implementação das funções de processamento da interface, foi utilizada a tecnologia de *Servlets*. Essa tecnologia especificada pela SUN (Voss, 2000, Lubling & Razorfish, 2000, McPherson, 2000) tem como intuito oferecer uma nova abordagem para programação de servidores HTTP¹⁰. Seu objetivo é oferecer a esses servidores, o mesmo papel desempenhado pelas *Applets* para o lado cliente (*browsers*), ou seja, propiciar um mecanismo independente de plataforma para a extensão dinâmica de funcionalidades - carregamento dinâmico de *bytecodes*¹¹ Java. Além da independência de plataforma, este mecanismo oferece algumas outras vantagens se comparado a tecnologias alternativas como CGI (Common Gateway Interface) (Ben-Natan, 2000), ASP (Active Server Pages) (Francis et al., 1998) e PHP (Professional Home Page) (Professional..., 2000):

- *Rapidez de resposta*: Cada vez que um CGI é requisitado, um novo processo é criado pelo sistema operacional do servidor WWW. Ao invés disso, um processo só é criado para uma servlet a primeira vez que ela é invocada (a menos que o seu código seja alterado). A partir de então, todas as requisições subsequentes são feitas ao processo residente.
- *Gerência de Sessão*: Pelo fato do protocolo HTTP ser independente de estado (*stateless*), o conceito de sessão de trabalho é inexistente, ou seja, requisições subsequentes de um mesmo usuário a um mesmo servidor HTTP são tratadas de forma completamente independente. Em (Ben-Natan, 2000), é apresentada uma tecnologia desenvolvida pela Netscape chamada *cookies* que permite endereçar essa questão, ainda que de maneira rudimentar. Além dessa opção, as servlets

¹⁰ Hyper Text Transfer Protocol - protocolo usada para transferência de documentos HTML na World Wide Web (WWW)

¹¹ Código intermediário gerado para a Máquina Virtual Java (JVM)

permitem a criação de uma verdadeira sessão de trabalho, na qual objetos podem ser armazenados e posteriormente recuperados, permitindo assim a comunicação entre servlets. Dessa maneira, é possível não apenas reconhecer quem é o usuário que vem interagindo com a aplicação, como construir modelos complexos de comportamento.

- *Capacidade de desenvolvimento*: Ao contrário das linguagens usadas em ASP e PHP (que são meras linguagens de script), servlets utilizam uma verdadeira linguagem de programação (Java) e, com exceção de algumas restrições de segurança, podem fazer qualquer coisa que um objeto Java ordinário é capaz. Exemplos dessas habilidades são o atendimento paralelo e a sincronização de múltiplas requisições simultâneas e a capacidade de delegação de uma requisição a outros servidores (ou outras servlets), a fim de realizar funções como balanceamento de carga entre servidores redundantes.

Portanto, em cada um dos servidores IU, existe um conjunto de servlets com objetivo de permitir (e controlar) o acesso a funções específicas de gerência, implementadas pelo AMS. A estratégia de alocação de funcionalidades às servlets é a seguinte: (i) primeiro, para cada caso de uso do nível de gerência é feita uma decomposição funcional; (ii) uma vez feito isso, para cada uma das funções terminais encontradas, é, então, associada uma servlet a ser implementada. No final desta subseção, este processo é exemplificado, usando os casos de uso *Login* e *Selecionar Vídeo*.

O conjunto de todas as funcionalidades associadas às servlets - resultantes do processo de decomposição funcional aplicado a todos os casos de uso - define também a interface contratual do AMS, ou seja, todos os serviços que ele deve oferecer. O framework especializado na seção 5.4, se apresenta como uma base de conhecimento capaz de responder todas as questões de competência tanto do domínio quanto da aplicação, o acesso a esse conhecimento (ou a parte dele) é provido de maneira organizada pelo AMS

O AMS desempenha, também, o papel de interface com a base de dados persistentes. Como os frameworks são desenvolvidos utilizando o *package Set*, a persistência é feita através do tipo `PersistentSet`. Um objeto `PersistentSet` é um conjunto capaz de fazer sua persistência (e de todos os elementos que o compõem) de maneira transparente ao usuário do tipo. Elementos que devem ser persistentes devem implementar a interface `SetElement`. A base de dados é definida, portanto, como uma família, ou seja, um conjunto de conjuntos. Cada conjunto membro de uma família deve implementar a interface `MemberSet`. As classes `Set` e `PersistentSet`, bem como as interfaces `SetElement` e `MemberSet` foram definidas na seção 4.3.1. Ao ser construído, o AMS recupera cada conjunto membro da base de dados, passando a referenciá-los diretamente.

A figura 5.17 mostra o uso dessa abordagem para o caso do sistema Hipervisão.

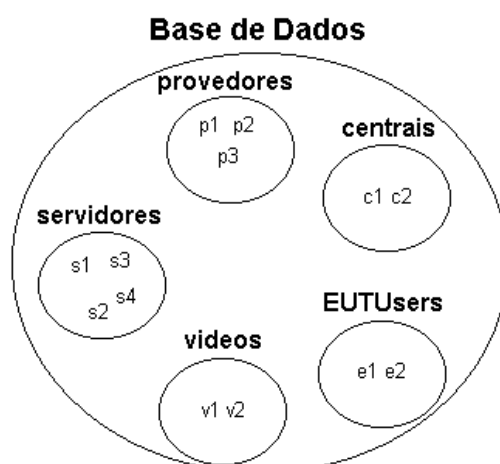


Figura 5.17 - Base de Dados como uma família de conjuntos

Como os servidores de interface (IU) e o AMS estarão no caso geral remotamente distantes, a sua comunicação é feita através de um ORB (*Object Request Broker*) que gerencia a comunicação entre os objetos distribuídos. Este componente permite tanto a invocação remota de métodos quanto a troca de objetos passados como parâmetro através da rede. Esta tecnologia que teve origem com o advento das RPCs (Remote Procedure Call) conta atualmente com algumas alternativas de implementação, sendo as principais delas:

- **CORBA (Common ORB Architecture) - OMG:** Provê um suporte completo a formas complexas de comunicação remota de objetos, implementados em diversas linguagens de programação e executando em diversas plataformas (Ben-Natan, 2000).
- **DCOM (Distributed Common Object Model) - Microsoft:** Suporta a comunicação de objetos implementados em diversas linguagens diferentes, executando na plataforma Windows (Ben-Natan, 2000).
- **Java RMI (Remote Method Invocation) - SUN:** Suporta a comunicação de objetos Java em diversas plataformas (Horstmann & Cornell, 1997, McPherson, 2000, McCluskey, 2000).

Um importante requisito deste trabalho é manter a independência de plataforma, eliminando a possibilidade do uso de DCOM. Sem dúvida o padrão CORBA é a mais completa das três alternativas, mas como nesse trabalho todos os objetos são implementados na mesma linguagem (JAVA), para efeito de simplificação a opção foi feita pelo uso de Java RMI.

Assim como nas RPCs, a comunicação remota em RMI é intermediada por objetos *proxy* chamados *stubs* e *skeletons*. Esses objetos são gerados a partir de uma interface e, portanto, possuem os mesmos métodos declarados por ela. Desse modo, um objeto remoto deve implementar uma interface que encapsule todos seus métodos passíveis de serem invocados remotamente. Além disso, é necessário que ele implemente o mecanismo necessário à transferência de parâmetros através da rede (*marshalling*) ou estenda uma classe que implemente este mecanismo. A classe `UnicastRemoteObject`, provida pelo pacote `java.rmi.server` da SUN, é um exemplo de classe deste tipo.

Finalmente, para que possa ser encontrado pelos clientes, o objeto remoto deve ser registrado em um serviço de nomeação. Um serviço deste tipo é provido pela classe Naming do pacote `java.rmi`.

A figura 5.18 apresenta o diagrama de classes do AMS. Nesse diagrama, o objeto remoto AMS implementa a interface `RI_AMS`, na qual são declarados todos os seus métodos acessíveis a clientes remotos. O objeto AMS é instanciado, registrado no mecanismo de nomeação e mantido continuamente em execução pelo `AMSServer`. Para garantir que somente uma instância de AMS é gerada, este objeto utiliza o padrão *Singleton* apresentado em (Gamma et al., 1995).



Figura 5.18 - Diagrama de Classes do *Application Management System* (AMS)

5.5.2.1 - Decomposição funcional de casos de uso com delegação de responsabilidade a *servlets* de controle

Como citado anteriormente, a interface com usuário nesse projeto é implementada através da interação de um cliente HTTP com os servidores de interface (IU). Os IUs são compostos por um servidor HTTP padrão (JavaWebServer) e um conjunto de *servlets*, cujo objetivo é gerar conteúdo HTML dinâmico a partir da interação com as outras camadas da arquitetura de software - lógica do negócio e persistência. Por utilizar o protocolo HTTP para a troca de dados, as *servlets* estão amarradas ao modelo *request/response* do protocolo, ou seja, a cada interação do usuário é feita uma requisição ao servidor, que entrega a uma *servlet* específica a tarefa de respondê-la com uma nova página HTML de interface gerada dinamicamente. A figura 5.19 exemplifica este processo através do caso de uso *Login*.

UserLogin.htm

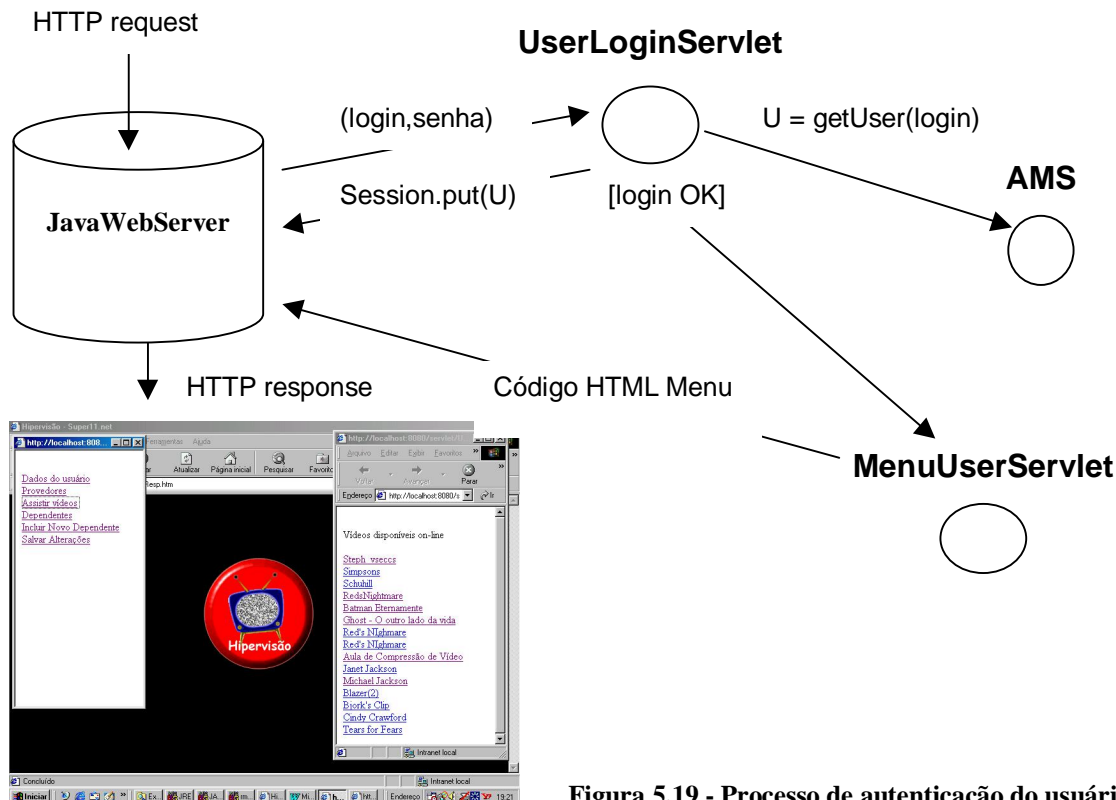
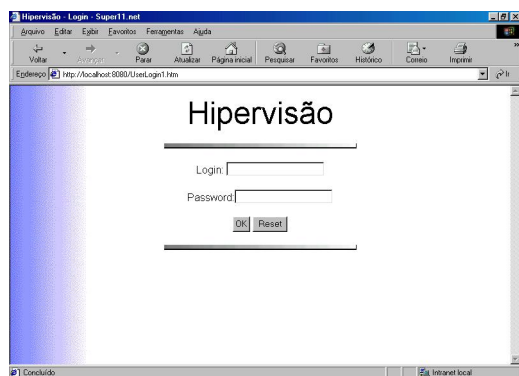


Figura 5.19 - Processo de autenticação do usuário

Nesse caso, o processo de autenticação é decomposto em duas funções: a verificação da senha e a geração do menu do usuário. A realização de cada uma dessas funções é delegada a uma *servlet* - respectivamente *UserLoginServlet* e *MenuUserServlet*. Cada *servlet* é, portanto, um objeto de controle responsável pela gerência de uma das funções que compõe o caso de uso. Nesse caso, excepcionalmente, as duas funções são conduzidas dentro de um mesmo ciclo *request/response*. Porém, isso não acontece na maioria dos casos de uso deste projeto, sendo geralmente necessários vários desses ciclos, cada um deles comprometido com a realização de uma tarefa específica. Dessa forma, o emprego de uma técnica para decomposição funcional dos casos de uso mostrou-se bastante apropriada.

A técnica escolhida foi a definida pelo padrão ITU-T Z.100 (ITU SDL, 1994) pela sua clareza e simplicidade. A figura 5.20 mostra um resumo da notação proposta por ela.

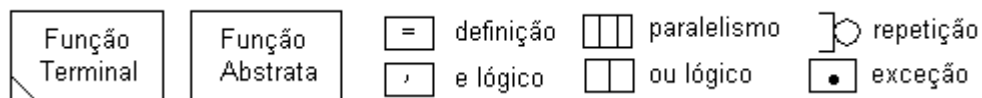


Figura 5.20 - Resumo da notação proposta na norma Z.100 para decomposição funcional

A seguir, é mostrada a aplicação dessa técnica para o caso de uso *Selecionar Vídeo* (figura 5.21). A tabela 5.3 apresenta cada função terminal, associando-as às *servlets* correspondentes. São também descritas as tarefas realizadas e o resultado gerado por cada uma das *servlets*.



Figura 5.21 - Decomposição funcional do caso de uso *Selecionar Vídeo*

Função	Servlet	Página HTML retornada
Mostrar opção de seleção	UserVideos.java	Quando o usuário acessa no menu a opção de <i>Selecionar Vídeo</i> , uma lista de <i>hiperlinks</i> é gerada contendo todos os vídeos pertencentes aos catálogos on-line e off-line do usuário.
Mostrar detalhes vídeo	VideoDetails.java	Para um dado vídeo, são mostradas todas as suas informações, como nome, diretor, gênero, nacionalidade, imagem de exemplo (<i>sample</i>), entre outras. Além disso, é oferecido ao usuário um mecanismo de escolha de uma das opções de áudio em questão (vide o exemplo da figura 5.22.)
Configurar Sessão	SessionConfig.java	Quando o usuário seleciona um vídeo para assistir, então é configurada uma sessão de exibição entre o terminal do usuário e um dos servidores que armazena o vídeo em questão ($ln(v_1, \text{armazenamento})$). A escolha do servidor é feita com base no número de sessões de exibição simultâneas que cada um deles está envolvido no momento, a fim de promover um mecanismo simples de balanceamento de carga entre os servidores.
Criar Sessão	Session.java	Este é objeto é, na verdade, uma <i>applet</i> e não uma <i>servlet</i> , já que todo o seu processamento é feito no terminal do usuário. Essa <i>applet</i> marca a fronteira entre os níveis de gerência e sistema, que somente é transposta ao comando de início de sessão dado pelo usuário (figura 5.23). Quando isso acontece, um objeto gerente de exibição é criado (JPlayer). Esse objeto é discutido detalhadamente na seção seguinte.

Tabela 5.2 - Funções terminais do caso de uso assistir vídeo e suas respectivas *servlets* associadas

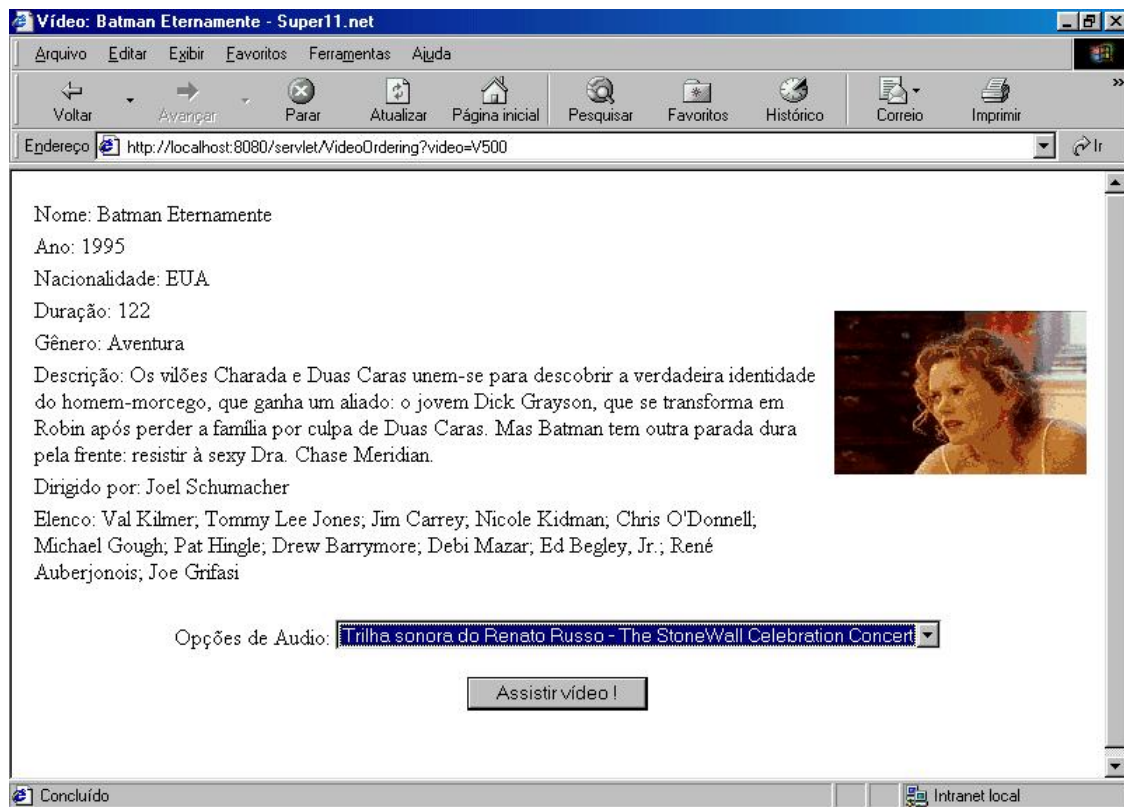


Figura 5.22 - Página de detalhes de um vídeo gerada pela *servlet VideoDetails*

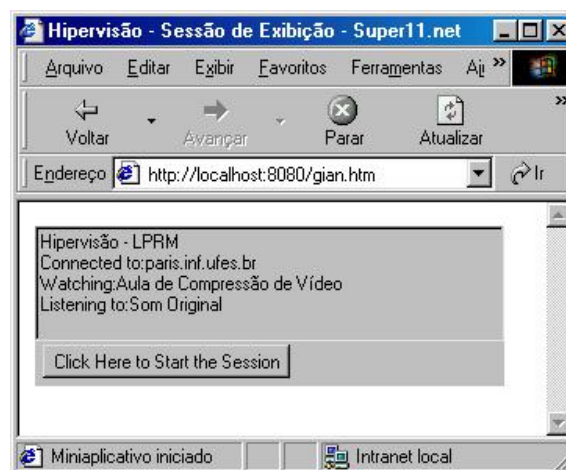


Figura 5.23 - *Applet* de configuração da sessão de exibição gerada pela *servlet SessionConfig*

5.5.3 - Projeto e Implementação do nível de sistema

O nível de sistema é percebido pelo usuário unicamente através do caso de uso *Assistir Vídeo*. Esse caso de uso é composto por duas funções distintas, mas altamente interrelacionadas:

- (i) Aquisição dos dados referentes às mídias que serão exibidas;
- (ii) Oferecer ao usuário os comandos de interação do videocassete virtual (exibir, retroceder, adiantar, dar pausa, controle de volume, entre outras) e cuidar para que a infra-estrutura usada possa dar suporte a essa interação;
- (iii) Cuidar do inter-relacionamento entre estas duas funções. Um exemplo do impacto que um comando de interação pode causar na infra-estrutura é o caso onde o usuário adianta o vídeo (*FF*), referenciando uma posição cujos dados (ex. vídeo, áudio) ainda não foram adquiridos. Nesta situação, os objetos deste caso de uso devem cuidar de questões como: informar ao usuário da situação ocorrida, gerenciar o transporte e a re-sincronização das mídias envolvidas, entre outras tarefas.

Assim como no projeto e implementação do nível de gerência, as decisões aqui tomadas visam empregar soluções abertas, padronizadas e arquitetonicamente neutras, a fim de atingir os requisitos de escalabilidade, flexibilidade e adaptatividade pretendidos. Além disso, o desenvolvimento é mais uma vez conduzido com um foco direto na prática de reutilização. A figura 5.24 ilustra, através de uma arquitetura em camadas, as decisões de linguagem, protocolos e *framework* adotadas.

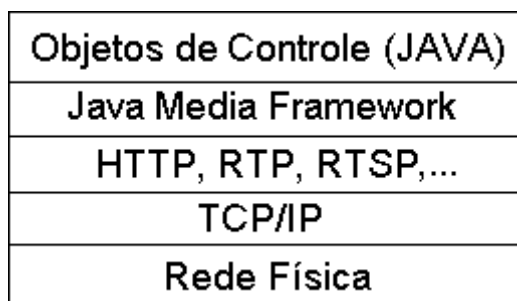


Figura 5.24 - Desenvolvimento em camadas do nível de sistema do *Hipervisão*

Como citado anteriormente, este caso de uso tem um forte relacionamento com a função abstrata *configurar exibição* descrita anteriormente e é exatamente na interseção entre eles que é marcada a fronteira entre os níveis de gerência e sistema. Quando o usuário pressiona o botão "Assistir Vídeo", é iniciado o processo de configuração da sessão de exibição. Esse processo comandado pela servlet *SessionConfig* consiste na configuração da infra-estrutura de sistema necessária para que o usuário possa assistir e interagir com vídeo escolhido. A gerência dessa infra-estrutura é delegada a um conjunto de objetos gerentes implementados em Java, cada um responsável por uma função específica (camada superior da figura 5.24). Primeiro, é criado um gerente de sessão (*Session*) que se conecta ao AMS, recuperando as instâncias apropriadas de *Video* e *Servidor*, além da opção de áudio escolhida para aquele vídeo. De posse desses objetos, o gerente de sessão cria um gerente de exibição (*JPlayer*)

que configura as dimensões da tela de apresentação, cria um painel de interação (*InteractionPanel*), contendo a janela de exibição de vídeo e um painel de controle. Por fim, é também criado um gerente de apresentação (*PlaybackManager*), responsável por criar, configurar e associar *Players*, que guiarão a aquisição de um mínimo de dados para dar início à apresentação, bem como a sincronização e apresentação destas mídias. A figura 5.25 mostra as relações envolvendo essas classes.

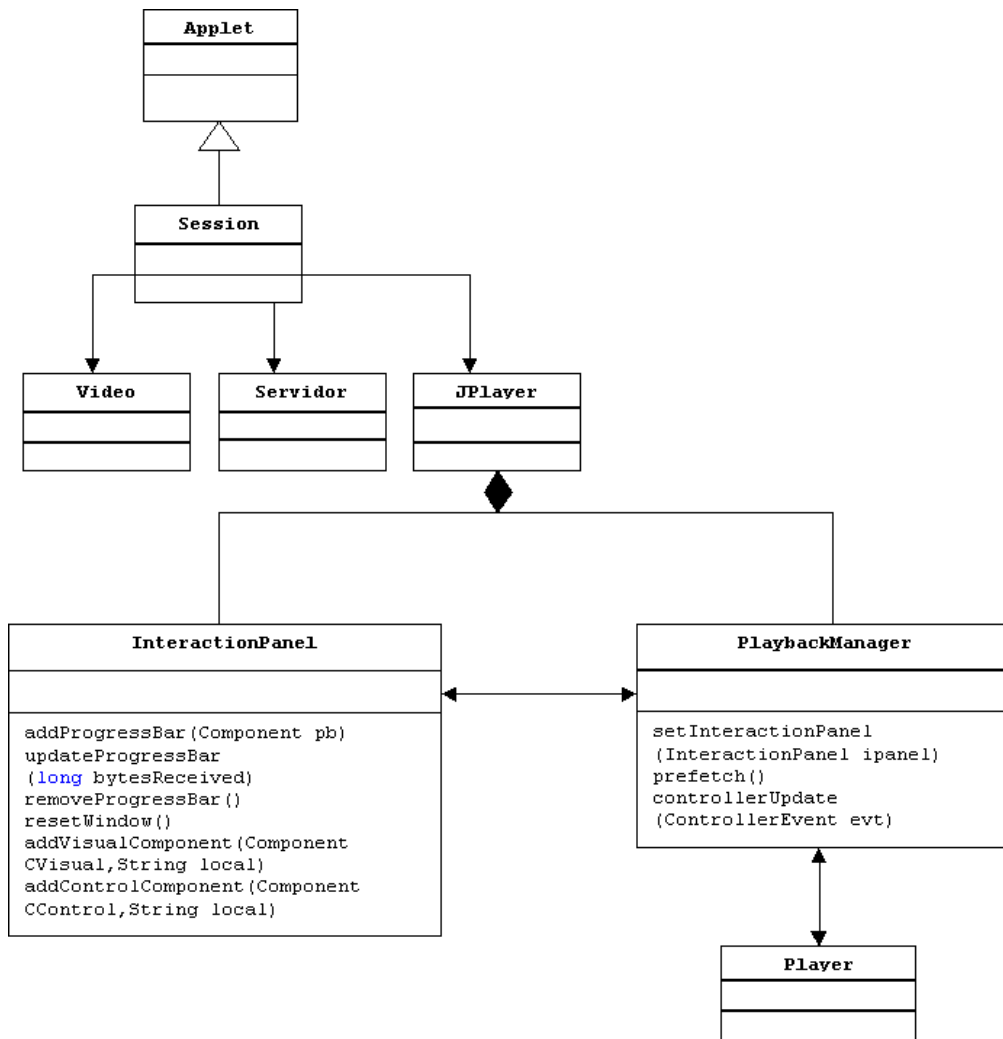


Figura 5.25 - Objetos envolvidos em uma sessão de exibição

Apesar de suas inúmeras vantagens, as primeiras versões da linguagem JAVA apresentavam um suporte extremamente limitado ao desenvolvimento de aplicações multimídia. A API básica suportava áudio de forma rudimentar (arquivos .au de 8 bits) e o suporte para vídeo era inexistente. Para suprir esta lacuna, um consórcio liderado pela SUN e contendo empresas como IBM, SGI e INTEL se comprometeu a desenvolver um extensa arquitetura chamada *Java Media and Communications API*. Esta arquitetura compreende um completo conjunto de API's para dar suporte à apresentação (*playback*) e criação de formatos multimídia avançados, como por exemplo:

- Síntese de MIDI e áudio de alta qualidade (Java Sound)
- Reconhecimento de voz e síntese de texto-palavra (Java Speech)
- Aplicações de telefonia (Java Telephony)
- Animações e Gráficos avançados (Java 2D e 3D)
- *Capture* e *playback* de mídias contínuas, ferramentas de conferência multimídia e trabalho cooperativo (Java Media Framework)

Como um dos objetivos deste trabalho é a implementação de um sistema de VoD, o foco da discussão é apenas no subconjunto do Java Media Framework (Java Media Player API) responsável pelas funcionalidades de *playback* (figura 5.26)¹².

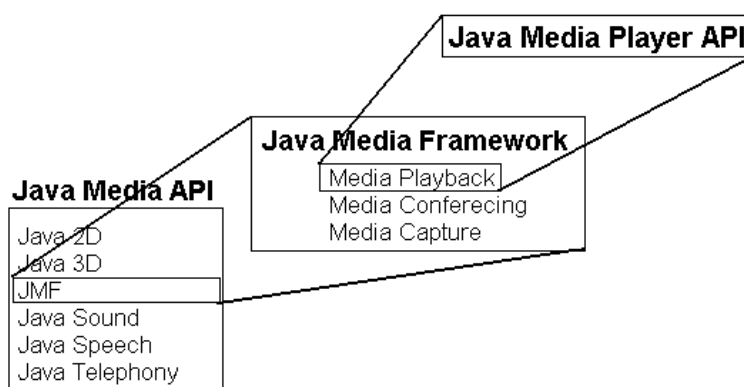


Figura 5.26 - O Java Media Player no contexto do Java Media and Communications API

O Java Media Player API (Sullivan et al., 1998) é um conjunto estruturado de classes e interfaces que encapsula estratégias de resolução dos principais problemas encontrados no desenvolvimento de aplicações envolvidas no transporte, sincronização, controle e apresentação de mídias contínuas.

Como benefício adicional, por ser feito em Java, o *framework* é independente de plataforma. Além disso, são utilizados mecanismos difundidos para o desenvolvimento de conteúdo multimídia para o ambiente WWW, sem demandar a presença de *plug-ins* adicionais na estação cliente. Além disso, suporta os principais formatos padronizados de codificação (MPEG-1, MPEG-2, H.261/H.263 -Quicktime, MIDI, AVI, WAV, AIFF, AU) e protocolos (HTTP-Streaming, RTP¹³, RTSP¹⁴, MediaBase).

A peça chave do framework é a interface *Player*. Um objeto implementando esta interface é capaz de ler e processar fluxos de dados multimídia lidos a partir de uma determinada fonte, e exibi-los em um determinado intervalo temporal. Um *Player* agrupa as funcionalidades definidas nos seguintes elementos:

- (a) **Clock:** Relógio de controle de tempo, que provê as funcionalidades básicas de sincronização, usadas para controlar a exibição dos dados;

¹² Informações detalhadas sobre as outras API's podem ser encontradas em <http://java.sun.com/products/java-media/jmf/index.html>.

¹³ Real Time Protocol

¹⁴ Real Time Stream Protocol: Artigos, listas de discussão e ferramentas sobre RTP e RTSP podem ser encontradas em <http://www.cs.columbia.edu/~hgs/rtp/>

- (b) **Controller:** Estende as funcionalidades do Clock para prover mecanismos de aquisição de recursos e de notificação de eventos;
- (c) **Duration:** Provê métodos para determinar a duração da mídia que está sendo apresentada;
- (d) **Media Handler e DataSource:** Funcionam, respectivamente, como abstrações de um decodificador de mídia e um protocolo de transporte e controle;

Como pode ser observado, a interface *Player* especializa as interfaces *MediaHandler* e **Controller** para encapsular um mecanismo padronizado de apresentação e interação, independente de conteúdo codificado e de protocolo.

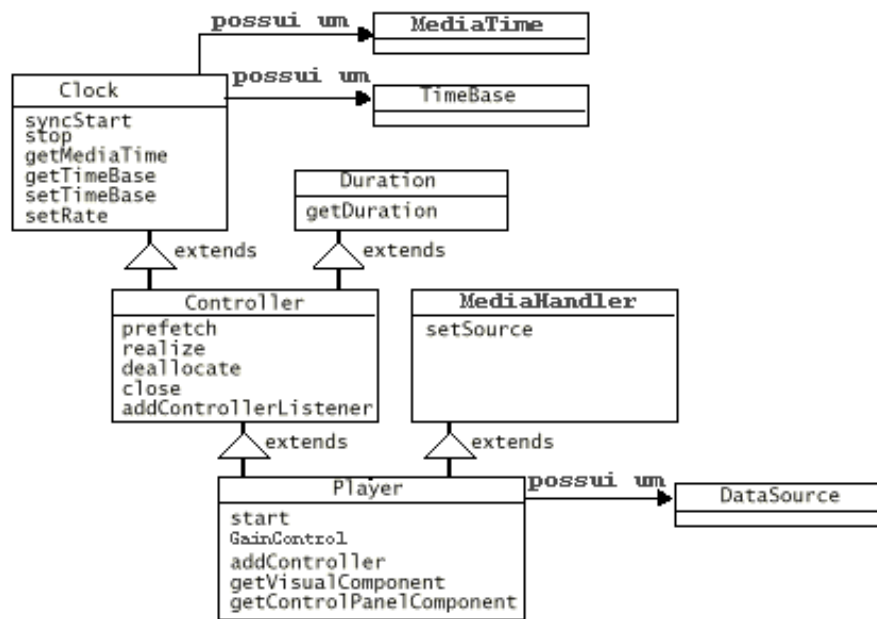


Figura 5.27 - Hierarquia das interfaces associadas ao *Player*

Antes que um *Player* possa exibir os dados recebidos, algumas operações devem ser realizadas. Pelo fato de muitas delas serem assíncronas e por ocorrerem em intervalos indeterminísticos de tempo, o *framework* faz com que seja possível controlá-las, definindo um conjunto de estados operacionais para um *Player* e provendo um mecanismo para movê-lo por estes estados (métodos *start*, *prefetch*, *realize*, *stop*).

A interface *Clock* define os primeiros dois estados: *started* e *stopped*. Em seguida, o estado *stopped* é dividido em três outros - *unrealized*, *realized* e *prefetched* - pelo *Controller*. Por fim, como as transições para *realized* e *prefetched* são assíncronas, são definidos dois estados intermediários: *realizing* e *prefetching*. A seguir, uma descrição sucinta desses estados é apresentada:

1. **Unrealized:** quando um *controller* é criado, ele se encontra neste estado. Ele não sabe o tipo de dado que irá processar nem os recursos que serão necessários;
2. **Realized:** neste estado ele determina os recursos necessários e adquire aqueles recursos que não requerem exclusividade. Por possuir informações suficientes sobre a mídia a ser exibida, após esse estado, um *Player* pode disponibilizar à aplicação os

componentes de interface adequados, como por exemplo, uma janela visual na qual um vídeo pode ser apresentado e/ou um painel de interação;

3. **Prefetched:** na transição entre *realized* e *prefetched*, o *controller* adquire os recursos exclusivos mínimos para que a exibição da mídia possa ter início. Quando o *Player* atinge este estado ele está pronto para iniciar a exibição em um tempo de latência inicial conhecido;
4. **Started:** mídia em execução. Como a exibição de uma mídia é geralmente iniciada sem que todo seu conteúdo tenha sido recebido, um *Player* em execução pode voltar ao estado de *prefetching*, após um evento que tenha gerado uma inanição de dados (*DataStarvation*). Exemplos de situações capazes de gerar eventos deste tipo são: (i) comando de reposicionamento do usuário, acessando uma posição da mídia que ainda não tenha sido recebida; (ii) taxa de transferência de dados na rede é menor que a taxa de exibição;
5. **Stopped:** execução interrompida, não iniciada ou finalizada.

O processo de transição entre estes estados possui restrições de natureza complexa, como por exemplo a opcionalidade de algumas transições e a existência de métodos que só podem ser invocados em determinados estados, sob a pena de ocasionar erros e exceções. Desta forma, o *framework* utiliza um protocolo essencial de notificação que envia para os objetos interessados mensagens de exceções, erros e de transições de estados (ex. *RealizeComplete*, *PrefetchComplete*, entre outros), além de mensagens que dizem respeito ao estado da mídia exibida (ex. *endOfMedia* e *DataStarved*), como mostra a figura 5.28. Este mecanismo faz com que a aplicação possa sempre garantir que um *Player* se encontra em um estado apropriado antes de invocar um serviço.

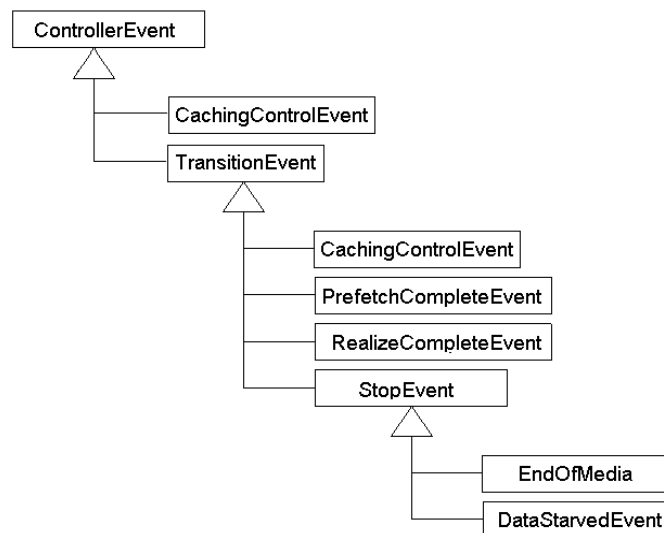


Figura 5.28 - Eventos gerados por um Controller (Player)

A figura 5.29 ilustra este mecanismo de notificação. Um objeto interessado em receber as notificações deve implementar a interface *ControllerListener* e, em seguida, ser adicionado à fila de interessados do *Player*. A partir de então, sempre que um evento

ocorrer, o *Player* notifica cada um dos objetos interessados, enviando a mensagem *ControllerUpdate*.

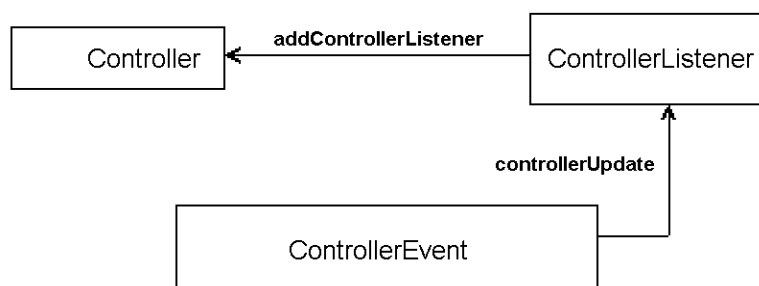


Figura 5.29 - Mecanismo de notificação de um Controller (Player)

Com a utilização desse *framework*, grande parte dos problemas de design de sistemas multimídia distribuídos discutidos na seção 5.2 é automaticamente endereçada, simplificando substancialmente o trabalho de formalização necessário para sua resolução. No entanto, é fundamental que os objetos que se relacionam com o *Player* tenham sempre conhecimento do estado em que ele se encontra, e organizem de forma ordenada o acesso a seus métodos, a fim de eliminar a possibilidade de condições de corrida (*race conditions*). Por este motivo, a interação entre as classes *PlaybackManager* e *Player* é devidamente formalizada e simulada, conforme definido pelo processo de desenvolvimento instanciado na seção 5.2.2. Para a tradução de modelos de projeto orientados a objetos em diagramas SDL, foi utilizado o processo proposto em (Verschaeve et al., 1996). Dessa forma, foram gerados os diagramas no nível de sistema¹⁵ (figura 5.30) para o *JPlayer*, e em nível de bloco para o *PlaybackManager* (figura 5.31) e *Player* (figura 5.32). Além disso, foram especificados os canais de comunicação contendo as mensagens válidas a serem trocadas por esses objetos em cada um das direções, bem como os diagramas em nível de processo contendo as máquinas de estados dos objetos *PlaybackManager* (figura 5.33) e *Player* (figura 5.34). A natureza formal da linguagem permite a execução de simulações, nas quais podem ser observadas as trocas de mensagens entre os processos ativos e como cada máquina de estados reage aos estímulos recebidos. O resultado de uma simulação realizado é mostrado na forma de um MSC na figura 5.35. Nessa figura, a entidade **env** (*environment*) representa o ambiente externo do sistema sendo simulado.

¹⁵ O nível de sistema citado aqui diz respeito à divisão entre níveis de sistema, bloco e processo feita pela linguagem SDL e não possuem nenhuma relação com os níveis de gerência e sistema citados anteriormente.

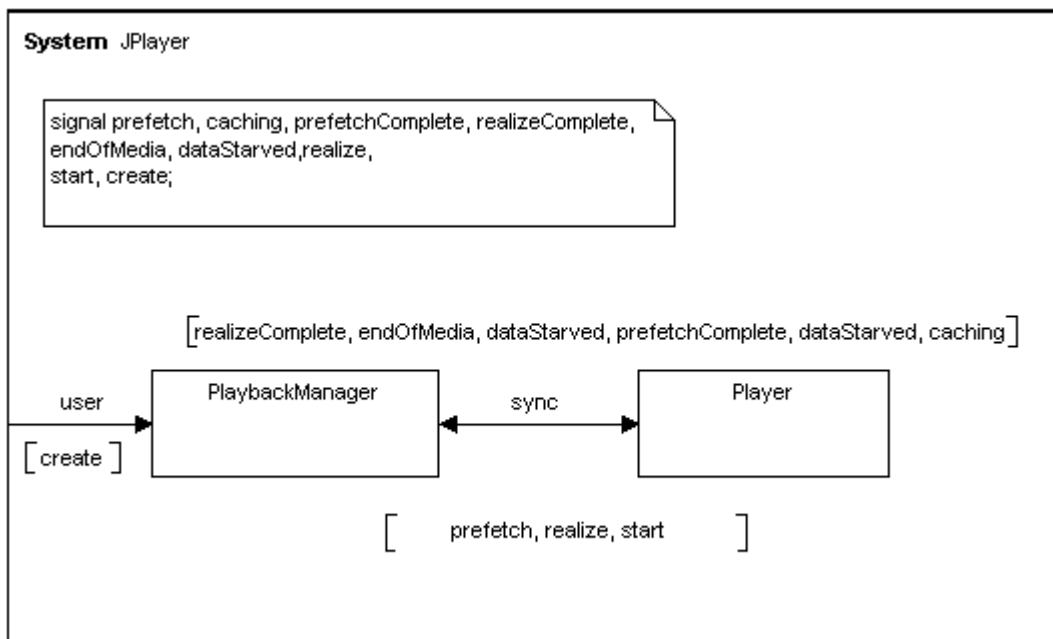


Figura 5.30 - Diagrama de sistema do objeto *JPlayer*

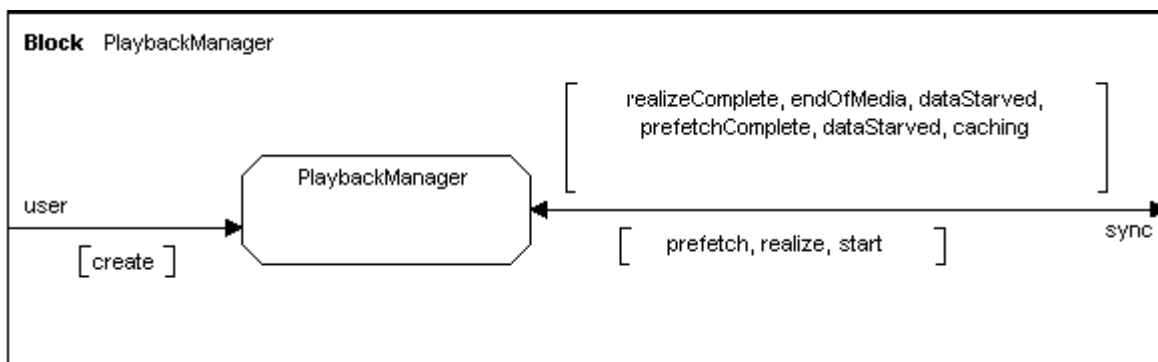


Figura 5.31 - Diagrama de bloco do objeto *PlaybackManager*

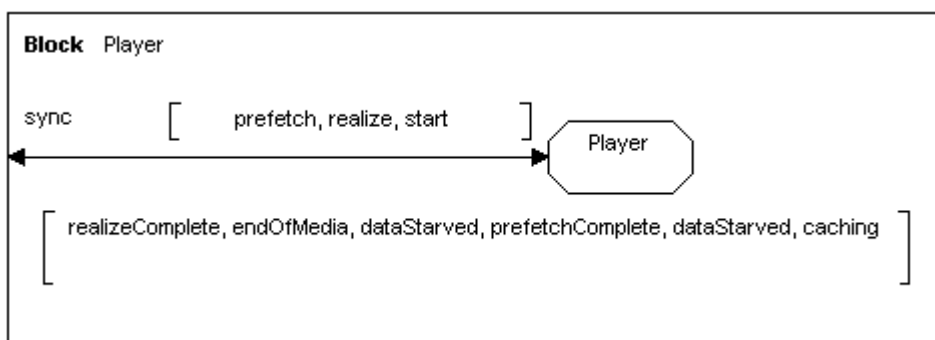


Figura 5.32 - Diagrama de bloco do objeto *Player*

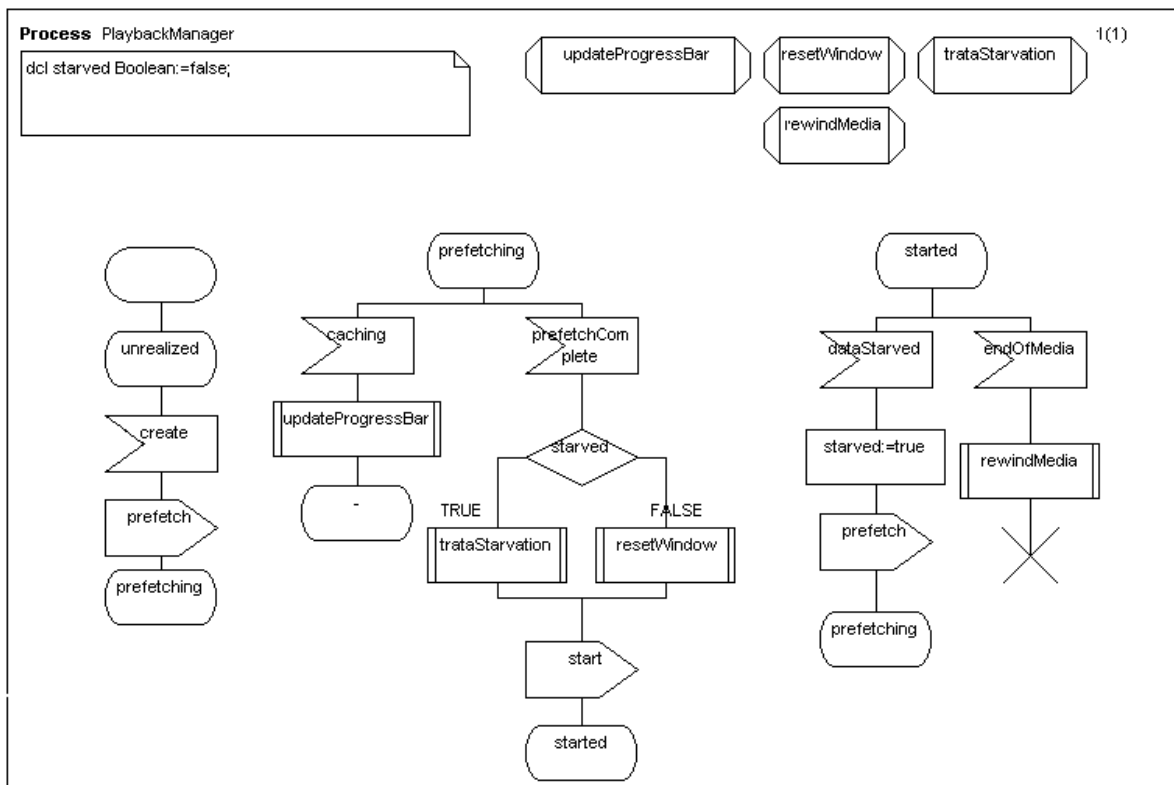


Figura 5.33 - Diagrama de processo do objeto *PlaybackManager*

Como citado anteriormente, o processo de aquisição dos dados necessários para dar início à apresentação (*prefetch*) é assíncrono e não-determinístico. Dessa forma, uma vez dado o comando de início de sessão, a apresentação das mídias pode demorar um tempo maior que o esperado pelo usuário, principalmente se o vídeo escolhido pertencer ao seu catálogo *off-line*. Para abordar esse problema, o objeto *Player* notifica o *PlaybackManager* o progresso da operação de aquisição dessas mídias, através de mensagens do tipo *caching* (figura 5.33). O *PlaybackManager*, então, delega ao *InteractionPanel* - através do procedimento *updateProgressBar* - a tarefa de reportar ao usuário o estado da operação (figura 5.36).

É importante salientar que o *PlaybackManager* atinge o estado de *prefetch* assim que o conteúdo adquirido for suficiente para começar a apresentação, ou seja, o usuário não precisa esperar que as mídias sejam totalmente transportadas para seu terminal para que a apresentação tenha início. Assim que isso acontece, o procedimento *resetWindow* é executado, requisitando ao *InteractionPanel* a transformação da janela de interação, a fim de prover os comandos necessários para que o usuário possa assistir e interagir com o vídeo (figura 5.37).

Finalmente, assim que a apresentação termina, as mídias que compõem o vídeo são automaticamente retornadas ao início pelo procedimento *rewindMedia* (figura 5.33) para que o usuário possa reiniciar a apresentação.

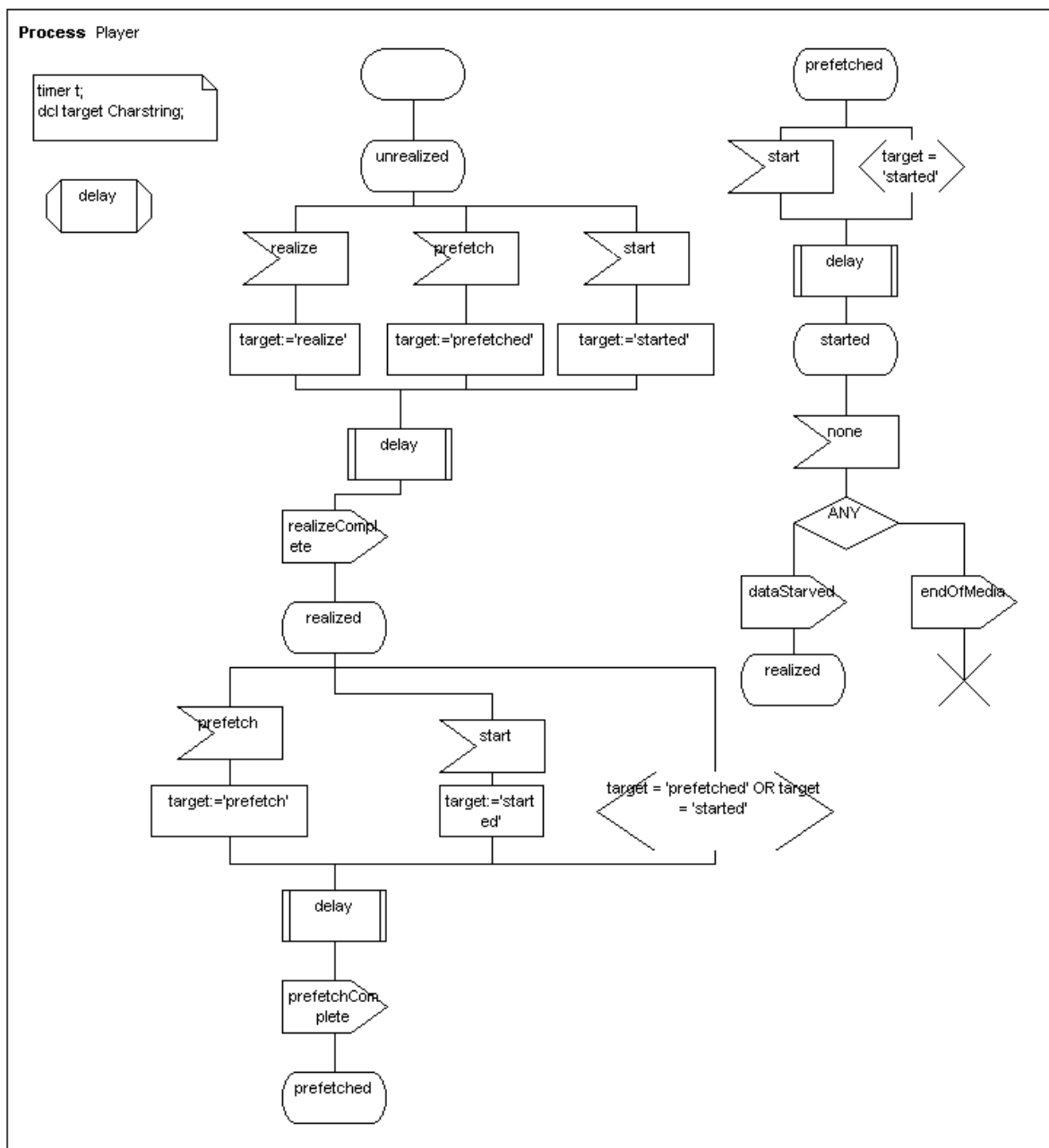


Figura 5.34 - Diagrama de processo do objeto *Player*

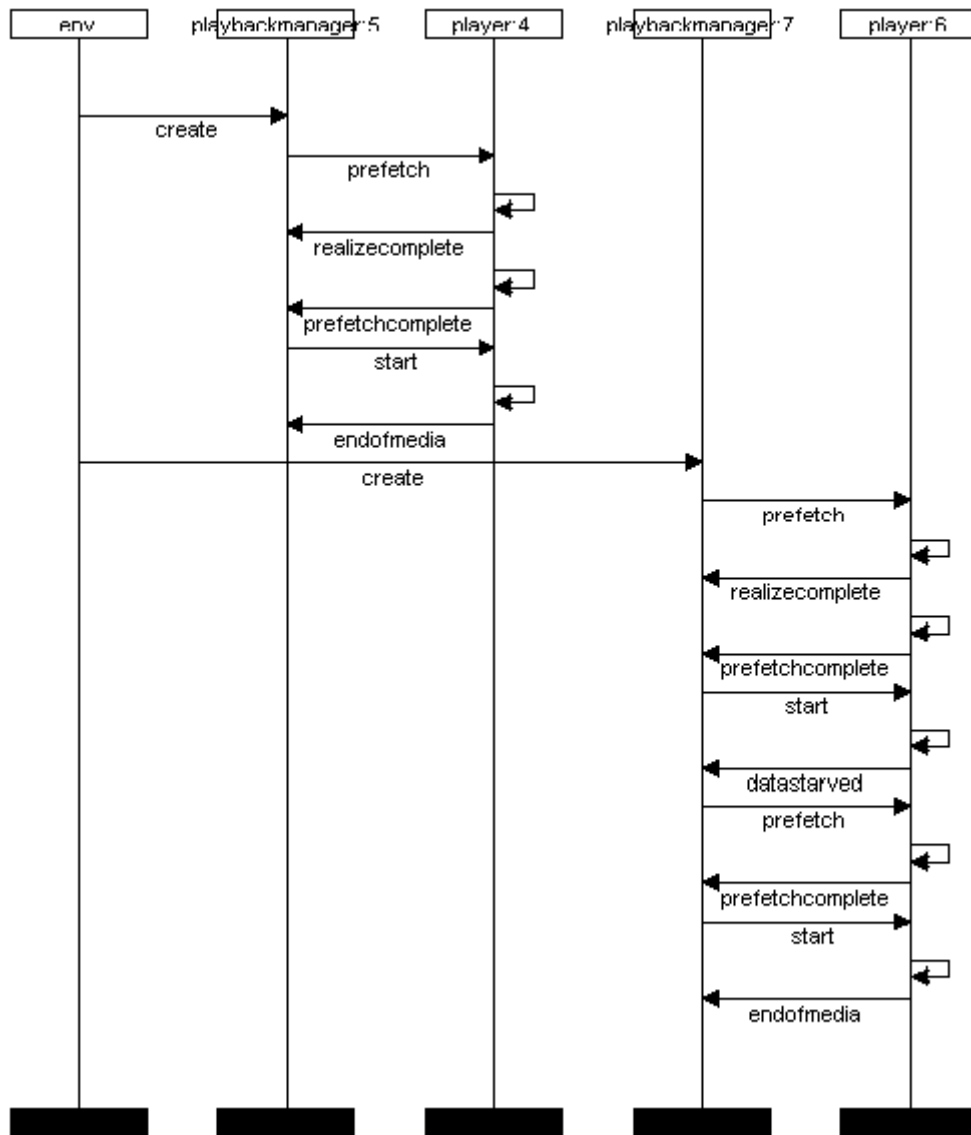


Figura 5.35 - Diagrama de MSC ilustrando a simulação de interações entre os objetos *PlaybackManager* e *Player*

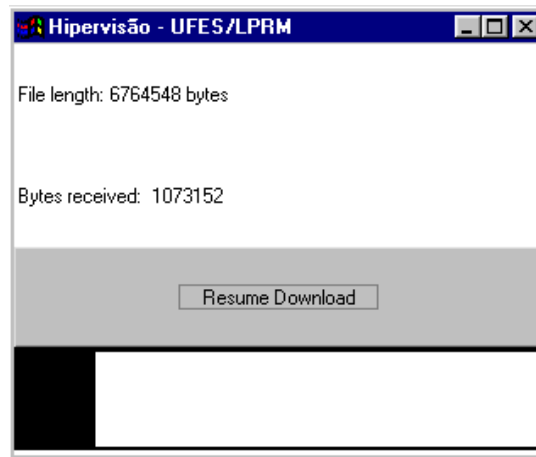


Figura 5.36 – Janela de informação e controle do processo de aquisição de dados

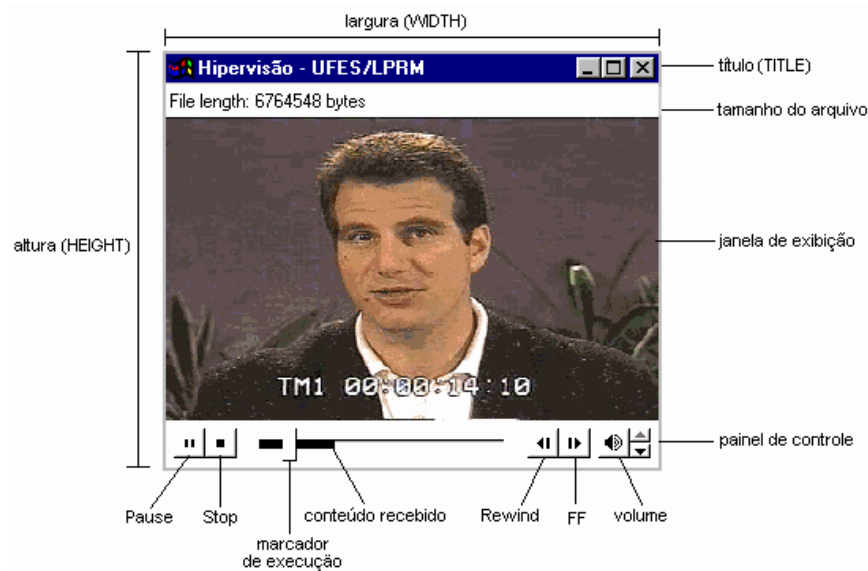


Figura 5.37 – Janela de apresentação e interação

5.6 - Conclusões

Este capítulo apresentou um estudo de caso para a metodologia de desenvolvimento para/com reuso proposta neste trabalho, agrupando experimentações individuais da aplicação de cada uma de suas partes. Primeiramente, o modelo de processos proposto foi instanciado para o desenvolvimento de uma aplicação de Vídeo sob Demanda. O modelo instanciado associa, de forma complementar, atividades, métodos e técnicas das áreas de inteligência artificial, métodos formais e engenharia de software orientada a objetos, formando um processo de desenvolvimento com múltiplos formalismos, capaz de suprir

algumas demandas não atendidas pelos métodos atualmente existentes para essa classe de aplicações.

Em seguida, como experimentação da abordagem de Engenharia de Domínio proposta, uma ontologia de gerência de Vídeo sob Demanda foi construída e o *framework* correspondente foi derivado. A metodologia se mostrou eficaz, capturando o conhecimento do domínio, sem impor muitos compromissos ontológicos, e gerando uma infra-estrutura de objetos, capazes de responder às questões de competência levantadas.

Finalmente, a análise de uma aplicação específica de Vídeo sob Demanda foi feita, identificando-se atores e casos de uso, além de restrições e questões de competência específicas do nível de aplicação. Com essa atividade, foi identificada a adequação de dois componentes existentes, passíveis de serem reutilizados: o *framework* de gerência, desenvolvido na seção 5.4, e um outro componente chamado *Java Media Framework*.

O desenvolvimento desse sistema constitui tanto um abrangente estudo de caso de desenvolvimento com reuso, quanto um exemplo de integração de Técnicas de Descrição Formal em um processo de desenvolvimento orientado a objetos. Isso se deve a três fatores: a especialização do *framework* de gerência (um *framework* caixa branca); a utilização do *JMF* (um *framework* caixa preta) e a realização das atividades propostas no processo instanciado

Por fim, é importante ressaltar a adequação do *framework* *JMF* como componente de reuso, capaz de encapsular, de forma flexível, soluções para grande parte dos problemas de projeto existentes nos sistemas multimídia distribuídos. Neste trabalho, o reuso do componente *JMF* facilitou substancialmente o processo de formalização necessário ao desenvolvimento da aplicação. Esse processo também foi facilitado pela escolha das TDFs adotadas (SDL/MSD), principalmente pelo fato de utilizarem conceitos comuns à orientação a objetos.