LogicOO – uma metodologia para modelagem e construção de sistemas multimídia distribuídos[1]

Giancarlo Guizzardi

José Gonçalves Pereira Filho

{gguizz, zegonc}@inf.ufes.br

Universidade Federal do Espírito Santo

Departamento de Informática

Laboratório de Pesquisas em Redes e Multimídia

Caixa Postal 019011 - Vitória – ES – CEP: 29060-970

Telefone: (027) 335-2131 – Fax: (027) 335-2850

Palavras-chave: engenharia de requisitos, análise de domínio, engenharia de ontologias, vídeo sob demanda, modelagem multimídia, orientação a objetos, Java, JMF.

Resumo: Este trabalho descreve a implementação de um sistema de vídeo sob demanda seguindo o *framework* para desenvolvimento de sistemas orientados a objetos "LogicOO". Este *framework* utiliza o conceito de *ontologias* no seu nível mais alto de abstração, o que permite o reuso do conhecimento sobre o domínio analisado. O protótipo construído, denominado Hipervisão, disponibiliza aos seus usuários o acesso a vídeos digitalizados e atualmente está em testes de uso no Departamento de Informática da UFES.

1. Introdução

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 destas metodologias se apresenta como um grande desafio, principalmente pela necessidade de agrupar em um mesmo processo de desenvolvimento uma flexibilidade de modelagem que permita o emprego de diversos níveis de formalismo.

Tradicionalmente, os modelos de processo propostos pela comunidade de sistemas multimídia distribuídos, abordam estes problemas através do uso de refinamentos sucessivos de especificações construídas com técnicas de descrição formal como LOTOS, ESTELLE, SDL, entre outras, altamente difundidas em áreas como engenharia de protocolos e sistemas de tempo real. Estas técnicas endereçam apropriadamente requisitos como controle de sincronização e de tempo real, possibilitando sua simulação e verificação com alto

[1] Este trabalho foi desenvolvido no contexto do projeto multi-institucional DAMD – Design de Aplicações Multimídia Distribuídas, e contou com o apoio financeiro do CNPq/PROTEM-CC, Fase 3.

formalismo e rigor matemático. A semântica formal destas linguagens, porém, 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 em um ciclo tradicional de engenharia de software. É indesejável que a construção de um modelo do problema esteja limitado a fortes características semânticas de uma linguagem específica, assumindo compromissos ontológicos que não existem no mundo real.

Por outro lado, as metodologias de engenharia de software, em sua grande maioria, possuem um modelo de processos bem sedimentado e possibilitam a especificação de uma aplicação em níveis progressivos de abstração, variando de um nível puramente conceitual na fase de análise a um nível formal na fase de implementação. No entanto, estas metodologias não possuem, nas fases de análise e projeto, o rigor matemático necessário para garantir a satisfação de alguns dos tipos de requisitos já citados.

Este artigo apresenta uma metodologia para desenvolvimento de sistemas de software que aborda esta importante questão utilizando de maneira complementar conceitos oriundos de áreas como métodos formais, engenharia de software orientada a objetos e sistemas baseados em conhecimento. A metodologia proposta, denominada ***LogicOO - Logical Object-Oriented Model***, cria um modelo abstrato de estruturação formal, baseado no trabalho desenvolvido em [1], e realiza sua integração com um modelo de construção de software orientado a objetos tradicional [4][5].

Abordando uma outra característica importante dos sistemas multimídia distribuídos - seus domínios de aplicação usualmente complexos - o *LogicOO* propõe uma nova abordagem para a fase de *análise e especificação de domínio*, na qual o conhecimento sobre o domínio do problema é descrito formalmente e, portanto, de maneira não ambígua. Esta abordagem tem como principal objetivo maximizar o reuso do conhecimento acerca de um domínio, ao invés da simples reutilização de partes do sistema, como encontrado nos cenários mais tradicionais da engenharia de software. Para atingir tal objetivo, o conhecimento do *domínio do problema*, ao invés do conhecimento da *aplicação*, é formalizado no seu mais alto grau de abstração através do uso do conceito de *ontologias* [6]. Devido ao emprego de uma sintaxe e semântica formais para a construção de ontologias, e ao seu poder de expressão para versar sobre os conceitos e relações deste domínio, a abordagem proposta permite a formalização de requisitos complexos, que fazem com que as fases seguintes do processo de construção de software herdem uma especificação consistente e sem ambigüidades.

O artigo também ilustra um exemplo de aplicação da metodologia apresentando a implementação de um sistema multimídia distribuído concebido a partir de uma ontologia do domínio de vídeo sob demanda. O sistema desenvolvido, denominado ***Hipervisão***, encontra-se atualmente em fase de testes de uso. O Hipervisão cria uma arquitetura lógica visando conectar vários departamentos acadêmicos, pertencentes ou não a uma mesma instituição, a fim de oferecer aos seus usuários finais (alunos e professores) o acesso a aulas digitalizadas e vídeos educacionais.

Apesar da metodologia ter sido concebida 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 ela oferecidos não são limitados a esta classe de sistemas. Principalmente no que diz respeito a nova abordagem na construção de infra-estruturas formais e reutilizáveis no nível de análise de domínio, são identificadas melhorias que se mostram úteis a qualquer classe de sistemas de requisitos suficientemente complexos.

As próximas seções estão organizadas da seguinte maneira: a Seções 2 apresenta uma visão geral do *LogicOO*; a Seção 3 introduz o paradigma das ontologias; a Seção 4 mostra a arquitetura lógica construída a partir do domínio ontológico; a Seção 5 apresenta a arquitetura conceitual concebida a partir da ontologia de vídeo sob demanda; a Seção 6 conclui o artigo apresentando as considerações finais e os novos trabalhos em andamento.

2. O Modelo LogicOO

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. Nesta fase, o modelo de requisitos é desenvolvido usando paradigmas informais (linguagem textual) e semi-formais (modelo de objetos do domínio do problema, modelo de interfaces), 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.

Apesar desta abordagem ser adequada para a maioria dos casos, um problema que acontece em sistemas complexos (por exemplo, alguns sistemas multimídia distribuídos), é que a presença de inúmeras restrições envolvendo os objetos existentes faz com que a especificação de requisitos - que em um sistema desse tipo é extensa e custosa - não possa ser facilmente reaproveitada e validada. O problema da validação ocorre devido à falta de formalidade na especificação dos requisitos, isto é, a riqueza dos detalhes do domínio é expressa de maneira informal e, muitas vezes, ambígua. Devido a esta natureza informal, densa e ambígua das representações textuais, um modelo de requisitos complexo pode ser obtido, contendo muitas contradições e inconsistências envolvendo as restrições.

O segundo problema é mais complexo e existe devido à representação imposta ao domínio de conhecimento. Para explicar este problema, um exemplo será usado. Suponha que uma equipe de desenvolvimento queira construir um sistema de gerência de empresas virtuais. Existe um grande e complexo conjunto de tópicos envolvendo este assunto. Portanto, após estudar o domínio da empresa virtual, e conhecer seus *conceitos* e *relações* [3] a equipe escolhe uma direção/aplicação específica (subconjunto do domínio do conhecimento) a ser atacada pelo modelo de requisitos do sistema. Uma vez que esta tenha sido escolhida, o conhecimento do domínio complementar é perdido e, portanto, não pode ser reutilizado por outras equipes desejosas de construir uma aplicação com uma outra direção, mas dentro do mesmo domínio. Em outras palavras, todo o domínio da empresa virtual tem que ser estudado por novas equipes de desenvolvimento.

Para tentar resolver este problema alguns propostas surgiram, introduzindo uma fase de *análise de domínio* visando tratar a reutilização no seu mais alto nível de abstração [7]. A idéia empregada consiste em construir infra-estruturas de domínio que são posteriormente preenchidas nas demais fases do processo. Esta abordagem endereça com sucesso a questão da reutilização mas peca por sua natureza informal e por dificultar a construção de requisitos conceituais mais complexos.

O modelo de ciclo de desenvolvimento proposto aborda estes problemas definindo um novo processo de engenharia de sistemas multimídia distribuídos, denominado *LogicOO* (*Logical Object Oriented Model*), que estende o ciclo tradicional de desenvolvimento em dois pontos estratégicos (Figura 1). Em primeiro lugar, a metodologia introduz um *modelo de construção de ontologias* para fase de análise de domínio. ***Ontologias*** [6] [11] vêm sendo pela área de Inteligência Artificial no processo de construção de Sistemas Baseados em Conhecimento buscando os mesmos objetivos: *análise*, *formalização* e *representação do conhecimento do domínio* a fim de obter uma especificação consistente e passível de reutilização. Isto é alcançado pelo fato de que uma ontologia de domínio pode ser especializada em várias e diferentes direções, isto é, modelos de requisitos de sistema diferentes podem ser obtidos para diferentes propósitos. Devido ao uso de uma sintaxe e semântica formais para a construção de ontologias, e ao seu poder de expressão para versar sobre os conceitos e relações deste domínio, o modelo proposto permite a formalização de requisitos complexos que farão com que as fases seguintes herdem uma especificação consistente e sem ambigüidades.

A segunda extensão proposta pelo modelo *LogicOO* é uma estratégia de seleção de casos de uso aplicada a uma "subfase" da fase de projeto, chamada de *projeto detalhado*. Geralmente, nesta fase são empregados vários tipos de diagramas de interação, seja com o objetivo de modelar atividades concorrentes (*diagramas de atividade*), modelar a mudança de estado de alguns objetos (*diagramas de transição de estado*) ou simplesmente modelar a troca de mensagens entre objetos (*diagramas de seqüência*). Desta maneira, são identificadas as funcionalidades que devem ser oferecidas, isto é, as operações a serem implementadas por cada classe. No entanto, estes diagramas são limitados por não permitirem modelagens formais e simulações de atividades mais complexas como, por exemplo, atividades que envolvam requisitos temporais e de sincronização, ou casos de uso cujo comportamento seja fundamentalmente baseado em objetos dependentes do estado[2].

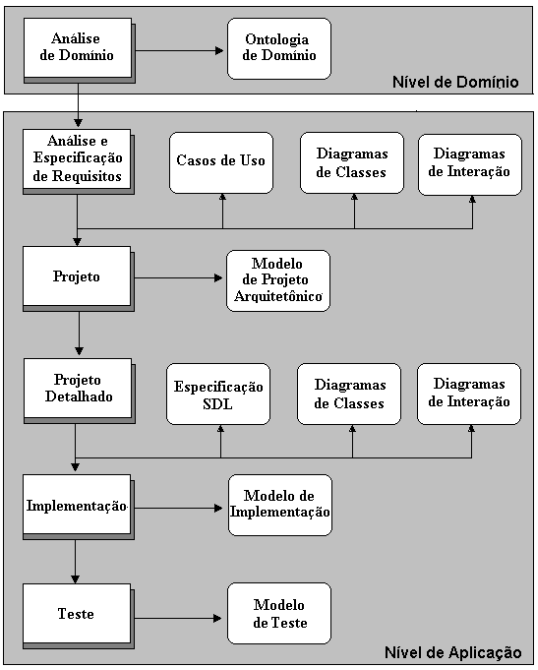

Figura 1 – O *framework LogicOO*

Os casos de uso que possuem as características citadas, são selecionados e descritos em SDL'92 [8] e, portanto, podem ser verificados

[2] Objetos dependentes do estado (*state-controlled objects*) reagem a mensagens recebidas de acordo com o seu estado interno [4].

e simulados. Além de impor um maior formalismo às relações dinâmicas envolvendo estes objetos, o uso de uma TDF baseada em um modelo de MEFEs[3] e 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 sequência) em uma só especificação. Esta técnica formal foi escolhida também por permitir a manutenção do modelo em mundo dos objetos e por sintetizar de maneira conveniente as funcionalidades de todos estes diagramas geralmente empregados. Como pode também ser observado na figura 1, a metodologia propõe uma alternativa à formalização dos casos de uso, empregando esta técnica somente naqueles casos em que ele se mostra necessária, a fim de minimizar os problemas da introdução de métodos formais em ambientes reais de desenvolvimento [3]. Apesar da importância que esta segunda extensão desempenha na metodologia, o objetivo deste artigo é se ater somente às questões relativas à nova abordagem de análise de domínio, deixando este tópico para ser abordado em trabalhos futuros.

O modelo de construção de ontologias é executado empregando duas técnicas complementares:

- uma notação gráfica (*diagrama de conceitos e relações*), que mostra os conceitos e as relações extraídas do domínio;
- uma notação textual (um formalismo baseado em Z [9], que mistura teoria dos conjuntos e lógica de primeira ordem), que é usada para construir os axiomas da ontologia versando sobre os elementos representados graficamente [3];

É importante notar que todos os elementos presentes na notação gráfica têm uma contrapartida na notação textual. Esta correspondência garante a semântica formal.

3. Representação do Conhecimento e Ontologias

Ontologia é um termo emprestado da Filosofia e significa "uma especificação de uma conceituação". Fazendo uma leitura mais pragmática, uma ontologia refere-se à descrição de conceitos e relacionamentos que podem existir para um agente ou uma comunidade de agentes. Desde tempos remotos os filósofos têm usado ontologias para descrever domínios naturais (as coisas naturais do mundo) e a existência de seres e coisas por si só. Entretanto, em um paradigma de representação do conhecimento, as ontologias tem um caráter um pouco diferente: seu propósito é firmar uma acordo com respeito ao vocabulário do domínio de interesse, a ser compartilhado por agentes que discursam sobre ele [10].

Ontologias constituem uma ferramenta poderosa para suportar a especificação e a implementação de sistemas computacionais de qualquer complexidade [6]. Ao usar esta abordagem para formar a fase de *análise e especificação do domínio* vários benefícios surgem:

[3] Máquinas de estados finitos extendidas

- *Formalização*: devido à natureza formal da notação usada, a especificação do domínio elimina contradições e inconsistências envolvendo as restrições resultando, portanto, em uma especificação não ambígua. Um outro ponto a ser destacado é que, já que uma notação formal é usada, a especificação formalizada pode ser automaticamente verificada e validada, se um provador automático de teoremas existe para aquela notação.
- *Comunicação*: ontologias são ferramentas úteis para ajudar as pessoas a se comunicarem, sob várias formas, acerca de um determinado conhecimento. Em primeiro lugar, elas podem ajudar as pessoas a raciocinar e a entender o domínio do conhecimento e, portanto, atuam como uma referência para a obtenção do consenso numa comunidade profissional sobre o vocabulário técnico a ser usado nas suas interações.
- *Representação do conhecimento e reuso*: o produto gerado pela fase de Análise e Especificação de Domínio é dito uma *ontologia de domínio*. A ontologia forma um vocabulário de consenso e representa o conhecimento do domínio no seu mais alto nível de abstração, possuindo um potencial enorme de reuso. O conhecimento formalizado na camada de domínio pode ser especializado em diferentes aplicações, servindo diferentes propósitos, por diferentes esquipes de desenvolvimento, em diferentes pontos do tempo.

À primeira vista, qualquer linguagem de representação formal do conhecimento, ou mesmo informal, poderia ser usada para representar ontologias. Na prática, entretanto, apenas poucas linguagens têm sido usadas para este propósito: *lógica de primeira ordem*, *KIF* (*Knowledge Interchange Format*), *Ontolingua*, *CML (Conceptual Modelling Language*) e *Description Logic*.

A escolha feita neste projeto foi usar uma abordagem composta, utilizando-se de duas notações: uma gráfica e uma textual. A notação gráfica é uma modificação da linguagem LINGO (Linguagem Gráfica para descrever Ontologias) [11], que mostra os conceitos extraídos do domínio (universo de discurso) e as relações entre eles. Na notação LINGO original, as relações não possuem direcionamento, o que foi modificado nesta versão. A identificação de elementos ativos e passivos no escopo de uma relação (introduzida pelo direcionamento) se faz fundamental devido aos princípios matemáticos da notação textual empregada no modelo.

Embora seja bastante similar ao modelo de Entidades-Relacionamentos (E-R) e Diagramas de Classe, o diagrama de Conceitos e Relações gerado por LINGO não carrega a semântica dos dados e funções, o que ocorre respectivamente nos modelo E-R e diagrama de classes. Esta notação gráfica não é formal, mas é essencial pois atua como elemento facilitador da comunicação entre os agentes envolvidos no processo.

A notação textual possui uma face híbrida, sendo baseada na *teoria dos conjuntos* e *lógica de primeira ordem*. Esta notação permite ao desenvolvedor formalizar, descrever, representar e raciocinar sobre o universo de discurso. Embora sendo um formalismo baseado em Z, ela produz um grande benefício ao introduzir um novo conjunto de operadores, os quais produzem uma sintaxe muito mais clara e amigável para uma pessoa sem conhecimento anterior de TDF's (Técnicas de Descrição Formal). Este é um ponto importante devido ao fato do ciclo de desenvolvimento proposto não ser complemente formal e também devido ao nível abstrato onde o formalismo é colocado. Em primeiro lugar, métodos formais representam um "choque cultural" para alguns desenvolvedores de software devido ao fato de vários deles não possuírem um bom *background* matemático. Em segundo lugar, algumas vezes esses modelos podem se tornar difíceis de servir como mecanismo de comunicação. Ambos os problemas citados são proporcionais ao aspecto não amigável da técnica.

4. Aplicação do Modelo no 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*) 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 sua 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.

A Figura 2 apresenta um diagrama que identifica os *conceitos* e *relações* básicas de um cenário de vídeo sob demanda [2], quando inicialmente pensamos no domínio do problema. Neste diagrama, os retângulos representam os conceitos enquanto que as setas ligando os retângulos representam as relações entre eles.

Numa primeira análise, cinco conceitos podem ser identificados: S(*Servidor)*, T(*Terminal)*, V(*Vídeo)*, P(*Provedor)* e C(*Central)*. Estes conceitos funcionam como classes, representando o comportamento de instâncias de conceitos numa relação classe/instância, como no paradigma de orientação a objetos. Deste modo, quando nos referimos a um servidor particular s_1, estamos nos referindo a uma instância do conceito S.

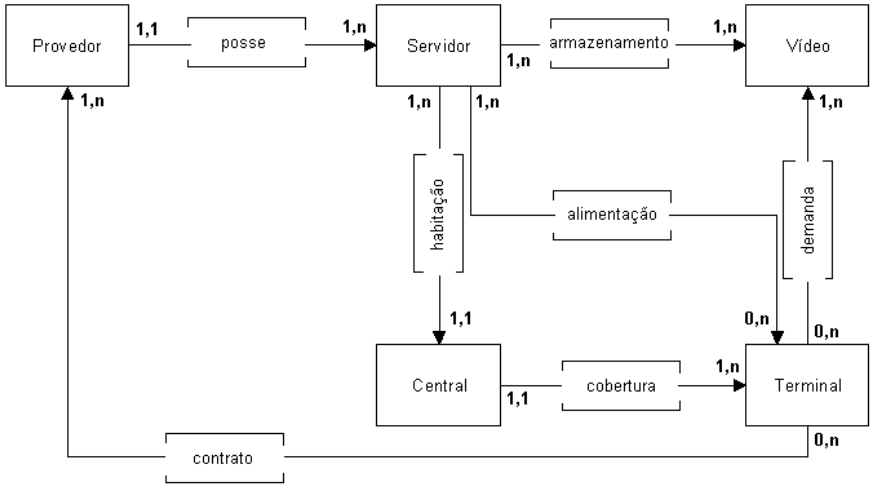

Figura 2 - Diagrama inicial de conceitos e relações do domínio de VoD[4]

[4] Ao contrário da versão original de LINGO, as cardinalidades (0,n) aqui são representadas. Isto é devido ao fato de que a maioria das restrições de relacionamento foram derivadas exatamente das relações com esta cardinalidade.

4.1 Descrição dos Conceitos

- ***Terminal*:** O terminal representa o usuário acessando o sistema. É através dele que é oferecido as principais funcionalidades de um serviço de VoD: *navegação* e *controle*. Navegação é todo processo que o usuário passa do momento de entrada no sistema até que a sessão de exibição do vídeo seja iniciada. Está dividida em sub-processos, que são: *login/logout*, seleção da programação, verificação de filmes assistidos, verificação de contas e formatação do vídeo. Neste último, o usuário deve estar habilitado a escolher o formato do vídeo a ser assistido, como som original, idioma, legendas e até mesmo a quantidade de comerciais assistidos podem ser modificados de acordo com as preferências do usuário. O controle consiste em receber, decodificar e apresentar o vídeo e enviar comandos de interação a um servidor, de acordo com as instruções do usuário. O terminal recebe o vídeo através de uma conexão entre ele e o servidor e interage com este provendo ao usuário as funcionalidades de um videocassete virtual (dar pausa, adiantar, retroceder, parar).
- ***Servidor*:** armazena os vídeos e os transmite aos terminais. Tem também a função de responder aos comandos de interação dos vários terminais simultaneamente. Fornecem instruções necessárias para a execução das funções de gerenciamento, controlando o acesso dos usuários, provendo o serviço de navegação e controlando sessões de exibição de vídeo.
- ***Vídeo*:** é provavelmente o principal ator neste domínio. Vídeos são armazenados em servidores e possuem atores principais, diretores, duração, trilha sonora, gênero, som original ou dublado, legenda, etc. Estes atributos servem para orientar o usuário na escolha de um vídeo, podendo fazer pesquisas de vídeos por diversas critérios.
- ***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*:** desempenha o papel de uma central telefônica, central de TV a cabo (CATV) ou ainda uma rede local (LAN) no cenário educacional. Esta entidade exerce o papel de delimitador do domínio de um conjunto de Terminais e Servidores..

4.2 Descrição das Relações

- Servidor ***habita*** Central: enfatiza a presença que deve existir do servidor na central que cobre os terminais que aquele servidor alimenta. O servidor pode estar localizado fisicamente ou não na central, desde que exista uma comunicação com esta central que permita alimentar os terminais.
- Central ***cobre*** Terminal: enfatiza aqui a comunicação que deve ocorrer entre terminal e central para que este possa ser alimentado por um servidor. Um servidor só pode alimentar um terminal se houver uma central habitada por este servidor que cubra aquele terminal.
- Servidor ***alimenta*** Terminal**:** Esta relação encapsula um conjunto de restrições lógicas 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 presença dela é importante para posse se restringir o conjunto de servidores que alimentam um determinado terminal.

- Terminal ***demanda*** vídeo**:** Um terminal só pode demandar um vídeo que pertença ao catálogo de vídeos que ele contrata, ou seja, que pertença a um dos provedores que ele contrata.

4.3 Caracterização do Modelo

Pode ser observado que em cada relação entre conceitos existe sempre um conceito *ativo* e um conceito *passivo* (atores e pacientes [1]). Podemos então considerar a existência de três conjuntos no universo das relações entre eles: X (ativos), Y (passivos) e R (Relações). É importante ressaltar que em toda relação os elementos ativos sairão do conjunto X, mas nem todo elemento do conjunto X precisa ser necessariamente ativo em uma relação. O mesmo ocorre para o conjunto Y de elementos passivos.

Estes conjuntos são altamente dinâmicos e o seu escopo é determinado pela relação, isto é, um objeto pode pertencer ao conjunto dos ativos em uma relação e ao conjunto dos passivos em outra. Por exemplo, a relação *contrato* existe entre terminais e provedores. Seja então r_1 esta relação. No escopo de r_1, X é o conjunto T de terminais e Y é o conjunto P de provedores. No modelo *LogicOO*, um terminal deve contratar no mínimo um provedor, podendo contratar até n deles, mas um provedor pode não ser contratado por nenhum terminal. Assim, supondo um provedor p_1 pertencente ao universo modelado, que não fosse contratado por nenhum terminal, então p1 pertenceria ao conjunto Y dos objetos passivos no escopo da relação r_1, mas não seria um elemento passivo em nenhum par ordenado (t_I,p_1) que tornasse relação r_1 verdadeira. Descrevendo logicamente esta situação temos:

$$\forall x{:}X,y{:}Y\ (\text{contrato}(x,y) \rightarrow (x \in T) \wedge (y \in P)\ \text{mas}\ \exists y_1 : Y, \forall x{:}X\ \sim\text{contrato}(x,y_1))$$

4.3.1 Operadores do Modelo

a) Universo de uma relação (U): Como já mencionado, em uma dada relação r_k temos os conjuntos X e Y dos elementos ativos e passivos. Portanto, o conjunto $X \times Y$ (X *cartesiano* Y) contém todas as possíveis combinações entre elementos ativos e passivos, para aquela relação. Porém, podem existir relações em que nem todo par ordenado (x_i, y_j) torna a relação verdadeira. É então definido o conjunto $U(r_k)$, como sendo o subconjunto de $X \times Y$, onde todos os pares ordenados (x_i, y_j) tornam a relação r_k verdadeira, isto é:

$$\forall r{:}R\ U(r) = \{ (x,y) \in X \times Y \mid r(x,y) \}$$

b) **Imagem (Im):** O operador imagem possui dois argumentos: um elemento (ou conjunto de elementos) do conjunto X e uma relação r qualquer, retornando um conjunto de elementos de Y que tornam esta relação verdadeira. É importante lembrar que os conjuntos X e Y são dependentes de cada relação. Por exemplo, para um dado servidor s_1, temos que:

$$\text{Im}(s_1,\text{armazenamento}) = \{ v_1, v_2, v_3 \}$$

ou seja, de todos os vídeos pertencentes ao universo modelado apenas os vídeos v_1,v_2 e v_3 é que são armazenados pelo servidor s_1. Uma propriedade importante deste operador é a distributiva: **$Im(\{s_1,s_2\},armazenamento) = Im(s_1,armazenazenamento) \cup^5 Im(s_2, armazenamento)$**

c) **Domínio (Dom):** O operador domínio possui como argumentos um elemento (ou conjunto de elementos) do conjunto Y e uma relação r qualquer, retornando um conjunto de elementos de X que tornam esta relação verdadeira. Por exemplo, para um dado terminal t_1, temos que:

$$Dom(t_1, cobertura) = \{ c_5 \}$$

ou seja, de todas as conexões pertencentes ao universo modelado apenas a central c_5 cobre o terminal t_1. Novamente, encontramos para este operador a propriedade distributiva:

$$Dom(\{t_1,t_2\},cobertura) = Dom(t_1, cobertura) \cup Dom(t_2, cobertura)$$

Finalmente, podemos observar que:

$$(y_1 \in Im(x_1,r_1)) \leftrightarrow (x_1 \in Dom(y_1,r_1))$$

d) **Cardinalidade (#):** A cardinalidade de um conjunto é o número de elementos pertencentes a ele. Aproveitando um dos exemplos anteriores temos :

$$\# Im(s_1,armazenamento) = 3$$

Para definir esses operadores de uma maneira mais formal, duas novas funções auxiliares e dois construtores Z são introduzidos. Essas funções são:

1. **ATV** (de ativo ou ator): esta função retorna o ator em um par. Por exemplo, seja (s_1, c_1) então ATV ((s_1,c_1)) = s_1
2. **PAS** (de passivo ou paciente): retorna o paciente em um par. Usando o mesmo exemplo, PAS((s_1,c_1)) = c_1
3. **Restrição de Domínio de uma relação com relação ao conjunto (Q ◂ R):** seja R o conjunto das relações ((x, y)). Então Q ◂ R significa o subconjunto de R que contém todos e somente os pares que possuem um elemento de Q como um ator.
4. **Restrição de Imagem de uma relação em relação ao conjunto (R ▸ W)**: seja R o conjunto de relações ((x, y)). Então R ▸ W significa o subconjunto de R que contém todos e somente os pares que contém um elemento de W como um paciente.

Com isto pode-se definir formalmente o operador **Imagem** como: **$Im(x_1,r_1) = Pas(\{x_1\} \triangleleft U(r_1))$.** Analogamente, podemos definir formalmente o operador Domínio como: **$Dom(y_1,r_1) = Atv(U(r_1) \triangleright \{y_1\})$**

Com a notação dos operadores introduzida acima, é possível fazer a descrição lógica das restrições de cardinalidade dos conceitos e relações até então presentes no modelo. A Tabela 1 mostra algumas regras acompanhada de suas interpretações.

[5] Este é o operador união de conjuntos **(∪)**; não confundir com o operador universo introduzido neste modelo **(U)**

Regra	Interpretação
∀p:**P** #**Im**(p,posse) ≥ 1	Para fazer parte do modelo um provedor deve possuir pelo menos um servidor, podendo possuir até vários deles.
∀p:**P** #**Dom**(p,contrato) ≥ 0	Um provedor pode não ser contratado por nenhum terminal, mas pode ser contratado por vários deles.
∀s:**S** #**Im**(s,alimentação) ≥ 0	Um servidor pode não alimentar nenhum terminal (um exemplo disto seria o caso do provedor que possui aquele servidor não ser contratado por nenhum terminal), podendo alimentar até vários deles.
∀t:**T** #**Im**(t,contrato) ≥ 1	Um terminal deve contratar pelo menos um provedor, podendo contratar até vários deles.
∀t:**T** #**Dom**(t,cobertura) = 1	Um terminal deve ser coberto por uma e somente uma central.

Tabela 1 - Exemplos de restrições formalizadas e respectivas interpretações

Além das restrições de cardinalidade esta notação também pode ser usada para descrever restrições de relacionamento. Por exemplo, na Tabela 1 usamos a regra **∀s:S #Im(s,alimentação) ≥ 0** afirmando que um caso onde o conjunto vazio **K (sendo K = Im(s_1,alimentação))** poderia ocorrer. Para esta regra usamos a seguinte restrição: "um servidor pode não alimentar nenhum terminal por ser posse de um provedor que não é contratado por nenhum terminal". O que ocorre neste caso é, na verdade, uma restrição de relacionamento, que pode ser formalizada da seguinte maneira:

$$\forall p{:}P, \forall s{:}S\ (\#Dom(p,contrato)=0) \wedge (s \in Im(p,posse)) \rightarrow (\#Im(s,alimentação)=0)$$

Entretanto, se um servidor alimenta um terminal isto implica que existe uma central, habitada por este servidor, que cobre este terminal. Ou seja,

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

Estes operadores podem ainda ser usados recursivamente como pode ser visto a seguir. Seja **H** o conjunto de todos os terminais cobertos pela central habitada pelo servidor s_1.

$$H = Im(Im(s,habitação),cobertura)$$

Neste caso, **H** não é o conjunto de terminais alimentados por s_1 e sim o conjunto de terminais potencialmente alimentáveis. Seja ***Q*** o conjunto de terminais alimentados por s_1, isto é: $Q = Im(s_1,alimentação)$. Pode então ser dito que:

$$\forall s{:}S\ Im(s,alimentação) \subseteq Im(Im(s,habitação),cobertura) \text{ ou } Q \subseteq H$$

ou seja, o conjunto de terminais que um servidor alimenta está contido ou é igual ao conjunto de terminais cobertos pela central habitada pelo servidor.

Além de permitir a descrição das restrições de cardinalidade e de relacionamento do domínio, o modelo empregada também pode ser útil na descrição de elementos triviais do universo modelado. Um exemplo claro disto seria a definição de um catálogo de vídeos:

- **Catálogo de vídeos de um provedor:** O catálogo de um provedor é a união dos conjuntos de vídeos armazenados por todos os servidores que ele possui, ou seja **Im(Im(p_1,posse),armazenamento)**

- **Catálogo de vídeos de um terminal:** O catálogo de um terminal é a união dos catálogos de vídeo de todos os provedores que ele contrata. Para achar este conjunto basta substituir o provedor específico $\mathbf{p_1}$ do axioma acima pelo conjunto de provedores contratados por $\mathbf{t_1}$: **Im (Im(Im(t_1,contrato),posse), armazenamento)**

  Uma melhor ilustração do que foi descrito pode ser feita através do seguinte exemplo. Seja: **P={p_1,p_2}, S = {s_1,s_2,s_3}, T = {t_1}** e **V = {v_1,v_2,v_3,v_4,v_5}**. Então:
  U (posse) = {(p_1, s_1),(p_1, s_2),(p_2, s_3)}
  U (armazenamento) = {(s_1,v_1), (s_1,v_2), (s_1,v_3), (s_2, v_3), (s_2,v_4), (s_3, v_4), (s_3,v_5)}

  Desta forma, temos que:

- **Catálogo(p_1) = Im(Im(p_1,posse), armazenamento) = Im({s_1,s_2}, armazenamento) = Im(s_1,armazenamento) ∪ Im(s_2,armazenamento) = {v_1,v_2,v_3} ∪ {v_3,v_4} = {v_1,v_2,v_3,v_4}**
- **Catálogo(p_2) = Im(Im(p_2,posse), armazenamento) = Im({s_3},armazenamento) = {v_4,v_5}**
  Se **U(contrato) = {(t_1,p_1), (t_1,p_2)}**, então:
- **Catálogo(t_1) = Im(Im(Im(t_1,contrato), posse), armazenamento) = Im(Im({p_1,p_2}, posse), armazenamento) = Im(Im(p_1, posse), armazenamento) ∪ Im(Im(p_2, posse), armazenamento) = Catálogo(p_1) ∪ Catálogo(p_2) = {v_1,v_2,v_3, v_4} ∪ {v_4,v_5} = {v_1,v_2,v_3,v_4,v_5}**

Finalmente, vale salientar que o modelo suporta também a definição de outras relações como relações de *especialização* e *todo-parte*. No entanto, a cobertura completa tanto do modelo e seus princípios matemáticos quanto do universo modelado foge da natureza introdutória deste artigo e será abordada em trabalhos futuros.

4.3.2 Os axiomas da ontologia

Uma vez que os conceitos e relações já estão definidos, e que a descrição do modelo que será usado para especificar o domínio ontológico foi introduzido, esta seção mostrará a descrição de todas as restrições de cardinalidade e relacionamento presentes nos requisitos modelados até o momento.

a) Restrições de cardinalidade

(1) ∀p:P #Im(p,posse) ≥ 1
(1)∀s:S #Im(s,armazenamento) ≥ 1
(2) ∀s:S #Im(s,habitação) = 1
(4) ∀s:S #Im(s,alimentação) ≥ 0
(5) ∀c:C #Im(c,cobertura) ≥ 1
(6) ∀t:T #Im(t,demanda) ≥ 1
(7) ∀t:T #Im(t,contrato) ≥ 1
(8) ∀p:P #Dom(p,contrato) ≥ 0
(9) ∀s:S #Dom(s,posse) = 1
(10)∀c:C #Dom(c,habitação) ≥ 1
(11)∀t:T #Dom(t,alimentação) ≥ 1
(12)∀t:T #Dom(t,cobertura) = 1
(13)∀v:V #Dom(v,demanda) ≥ 0
(14)∀v:V #Dom(v,armazena) ≥ 1

b) Restrições de Relacionamento

(15) ∀p:P, s:S (#Dom(p,contrato)=0) ∧ (s ∈Im(p,posse)) → (#Im(s,alimentação)=0)
(16) ∀s:S, ∀t:T alimentação(s,t) → ∃c:C habitação(s,c) ∧ cobertura(c,t)
(17) ∀s:S, ∀t:T alimentação(s,t) → s ∈ (Im(Im(v,contrato),posse))
(18) ∀s:S, ∀t:T alimentação(s,t) ↔ (16) ∧ (17)
(19) ∀s:S Im(s,alimentação) ⊆ Im(Im(s,habitação),cobertura)
(20) ∀s:S,∀v:V (#Im(s,alimentação)=0) ∧ (v ∈ Im(s,armazenamento)) → (#Dom(v,demanda)=0)

(21) ∀t:T, ∀v:V demanda(t,v) → v ∈ Im(Im(Im(t,contrato),posse), armazenamento)

5. Definição de uma Arquitetura Lógica a partir da Ontologia de Domínio

A Figura 3 mostra a arquitetura conceitual concebida a partir da ontologia de vídeo sob demanda desenvolvida na seção anterior. O sistema desenvolvido, denominado *Hipervisão*, emprega o paradigma descentralizado de vídeo sob demanda [1]. Particularmente, para a nossa escolha de materialização desta arquitetura (ambiente educacional), este é o único paradigma concretizável no momento.

Nesta arquitetura, os terminais do usuário são agrupados em centrais (*RDL - Redes de Domínio Local*) que, por sua vez, são conectadas umas às outras através de uma *Rede de Acesso Remoto* (*RAR*). Neste cenário, provedores podem possuir vários servidores (provedor *possui* servidor) em várias centrais (servidor *habita* central). Terminais também são conectados às centrais (central *cobre* terminal). Desta maneira, um terminal só pode estabelecer uma conexão para exibição de vídeo (servidor *alimenta* terminal) com um servidor conectado à mesma central que ele, e que pertença a um provedor por ele contratado (restrições de relacionamento 16, 17 e 18 enumeradas na seção anterior):

(16) ∀s:S, ∀t:T alimentação(s,t) → ∃c:C habitação(s,c) ∧ cobertura(c,t)

(17) ∀s:S, ∀t:T alimentação(s,t) → s ∈ (Im(Im(v,contrato),posse))

(18) ∀s:S, ∀t:T alimentação(s,t) ↔ (16) ∧ (17)

Apesar disso, cada provedor que um terminal contrata pode possuir um conjunto genérico de servidores, inclusive espalhados em várias centrais. Desta forma, o catálogo do terminal é composto por todos os vídeos armazenados em todos os servidores pertencentes aos terminais que ele contrata (restrição de relacionamento 21):

(21) (∀ t:T, ∀ v:V demanda(t,v) → v ∈ Im(Im(Im(t,contrato),posse),armazenamento))

e não somente os vídeos armazenados nos servidores que o alimentam (Im(Dom(t,alimentação),armazenar)), mesmo que este último seja o conjunto de vídeos que o usuário pode efetivamente assistir no momento em que desejar (chamado de seu catálogo *on-line*). Se o vídeo escolhido pelo terminal não está no seu catálogo *on-line*, então ele precisa ser transportado, através da RAR do servidor em que está armazenado, para um servidor conectado à mesma central deste terminal (s ∈ Dom(Dom(t,cobertura),habitação)).

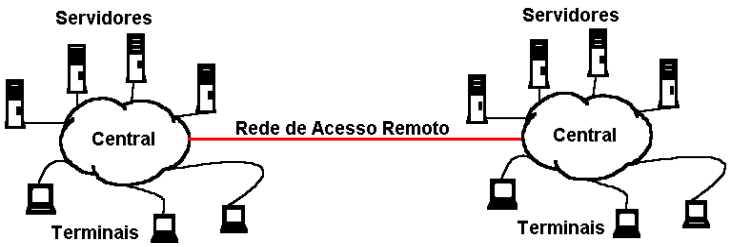

Figura 3 – arquitetura conceitual de vídeo sob demanda

5.1 Alternativas de Implementação

Após a concepção de uma arquitetura conceitual a partir de uma ontologia de domínio, várias aplicações podem ser derivadas. Duas diferentes opções de especialização são:

- **Cenário comercial:** Numa implementação comercial os terminais são materializados como televisões e *set-top boxes* (STB), a central é uma central telefônica conectando, através de enlaces ADSL, usuários dentro de um mesmo círculo de vizinhança, respeitando a 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 ou de entretenimento.
- **Cenário educacional:** Por um outro lado, em um cenário educacional os terminais são computadores ligados a uma rede local departamental. Provedores podem ser colegiados de cursos acadêmicos, departamentos ou centros. A RAR pode ser uma *backbone* institucional conectando os vários departamentos de uma mesma instituição ou até mesmo a Internet conectando centros educacionais ao redor do mundo.

No sistema *Hipervisão*, por se tratar de um projeto acadêmico, o último cenário foi o escolhido para ser implementado. Uma discussão mais detalhada sobre as funcionalidades do sistema e sobre a materialização dos elementos conceituais apontados é descrito em [15].

5.2 Considerações de Projeto e Implementação

O primeiro passo na evolução do *Hipervisão* do nível de domínio para o nível de sistema, seguindo o esquema proposto no *LogicOO*, é estruturar os conceitos, relacionamentos e requisitos (componentes de uma ontologia) em termos de casos de uso e diagramas de classes, atividades que compõem a fase de análise e especificação de requisitos em um processo de engenharia de software orientado a objetos. Posteriormente alguns casos de uso selecionados (por exemplo, aqueles que apresentam fortes restrições temporais) são especificados em SDL.

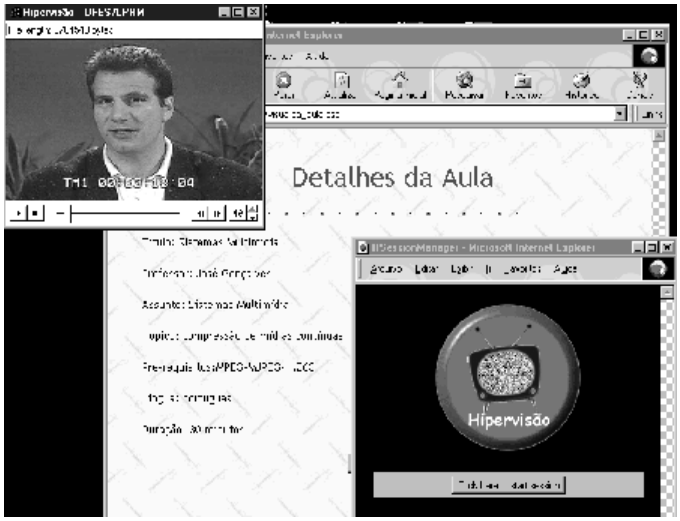

Figura 4 – Tela de exibição e controle de vídeo

Em termos de implementação, as funcionalidades do caso de uso *navegação* (grupo de funcionalidades que encapsula desde o *login* até a consulta e seleção de vídeos), são acessíveis aos usuários através do serviço WWW, usando um cliente Web padrão. Esta interação foi implementada em ASP (Active Server Pages) [12]. O caso de uso *controle* (grupo de funcionalidades que encapsula o processo de gerência da sessão de exibição,

incluindo os comandos de interação do videocassete virtual) foi implementado em Java, usando o JMedia, que é a implementação Intel do JMF (Java Media Framework – um conjunto estruturado de classes e interfaces para programação multimídia) [14].

A Figura 4 apresenta as principais telas de interação do Hipervisão.

6. Conclusões

As aplicações distribuídas de vídeo digital (principalmente as de vídeo sob demanda) são atualmente um tópico em destaque para pesquisa e desenvolvimento na área de multimídia distribuída. Porém, a maioria das pesquisas atualmente em curso abordam o problema somente do ponto de vista tecnológico, tentando conseguir desempenhos ótimos dos meios de armazenamento e comunicação. Este trabalho possui uma outra abordagem, concentrando-se nos seus aspectos de modelagem abstrata. Esta questão é fundamental, principalmente quando se deseja construir aplicações mais complexas, com inúmeras restrições de domínio.

Desta forma, é proposto um novo processo de engenharia de sistemas multimídia distribuídos, denominado LogicOO, que estende o ciclo tradicional de desenvolvimento abordando os problemas relacionados à representação, consistência e validação do conhecimento relativo a um domínio de interesse, bem como os problemas relacionados à reutilização deste conhecimento. Para isso, o framework LogicOO une características dos processos de engenharia tradicionais dos sistemas multimídia distribuídos (métodos formais), da engenharia de software e dos sistemas baseados em conhecimento, utilizando ontologias para o processo de modelagem da fase de análise e especificação do domínio e a técnica de descrição formal SDL para agrupar de maneira conveniente e mais formal os diversos diagramas de interação empregados na fase de projeto detalhado. Uma vez construída (no nível de domínio), uma ontologia pode ser especializada (no nível de aplicação) em diversos tipos de sistemas.

No caso do presente artigo, foi ilustrado como uma arquitetura conceitual, concebida a partir de uma ontologia de vídeo sob demanda, pode ser especializada em um protótipo operacional de propósito específico. A aplicação desenvolvida, denominada *Hipervisão*, oferece aos seus usuários finais o acesso a aulas e vídeos digitalizados. O sistema, além de outras funcionalidades, possibilita ao usuário controlar a sessão de exibição, configurar o vídeo a ser assistido e interagir com a exibição propriamente dita através da metáfora de um videocassete virtual. O estudo realizado também mostra como a ontologia utilizando o *LogicOO*, devido ao seu caráter geral, pode ter suas restrições incorporadas tanto em um modelo de objetos quanto em um cenário puramente estruturado utilizando banco de dados relacional. Atualmente o sistema encontra-se em fase de testes de utilização e, simultanemanete, está sofrendo melhorias na implementação da sua interface.

Alguns projetos estão sendo derivados do Hipervisão, utilizando o sistema como infra-estrutura básica. Exemplos incluem: *(i)* uma proposta de arquitetura para o desenvolvimento

de sistemas tutores inteligentes na Internet; e *(ii)* uma arquitetura de vídeo sob demanda para comércio eletrônico baseado em agentes.

Futuramente a abordagem utilizada para construção de ontologias estará sendo testada em uma grande variedade de domínios e comparada de maneira mais criteriosa a outras abordagens existentes. Este processo possibilitará a verificação de sua generalidade para outras classes de sistemas (senão sistemas multimídia distribuídos), assim como definir métricas mais concretas que avaliem os bons resultados observados subjetivamente.

Referências Bibliográficas

[1] Sampson, P.; Snijders, W. M.; Rossavik, K. "*Overall system specification and architecture*", Diamond Project, Deliverable no. 2, 1997.

[2] Guizzardi G., Pereira Filho J. G., "*Uma metodologia baseada em objetos para descrição lógica de sistemas de vídeo sob demanda*", IV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia, Rio de Janeiro, 1998.

[3] Guizzardi G., Pereira Filho J. G., "*Um framework para modelagem e construção de sistemas de vídeo sob demanda*", Em: Projeto DAMD – *Design* de Aplicações Multimídia Distribuídas - Ed. Wanderley Lopes de Souza, 1999 (livro em fase de impressão).

[4] Jacobson I. *"Object-Oriented Software Engineering"*, Addison-Wesley, 1992.

[5] Fowler M.; Scott K. *"UML Distilled"*, Addison-Wesley, 1997.

[6] Falbo, R.A., Menezes, C.S., Rocha, A.R.C., *"Using ontologies to improve knowledge integration in software engineering environments"*, Proceedings of the World Multiconference on Systemic, Cybernetics and Informatics / 4th International Conference on Information Systems Analysis and Synthesis SCI'98/ISAS'98, Orlando, USA, 1998.

[7] Arango, G. *"Domain analysis - From art to engineering discipline"*, Communications of the ACM, 1989.

[8] CTITT, COM X-R 17-E; *"Recommendation Z.100 - CTITT Specification and Description Language (SDL) and Annex A to the Recommendation"*, Geneve, 1992.

[9] Spivey J. M.; *"Understanding Z: A specification language and its formal semantics"*, Cambridge University Press, 1988.

[10] Gruber, "*Ontolingua: A mechanism to support portable ontologies, version 3.0"*. Technical Report, Knowledge Systems Laboratory, Stanford University, California, 1992.

[11] Falbo, R.A., Menezes, C.S., Rocha, A.R.C., , *"A Systematic approach for building ontologies"*, Proceedings of the IBERAMIA'98, Lisboa, Portugal, October, 1998.

[12] Francis, F. et al. *"Active Server Pages 2.0"*, Wrox Press, 1998.

[13] Horstmann C. S.; Cornell G. "Core Java: Fundamentals", 2nd Edition, Volume I e II, Sun Press,1997.

[14] Sullivan S.C. et al. "Programming with the Java Media Framework", Wiley Books, New York, 1998.

[15] Guizzardi G., Pereira Filho J. G., "*Hipervisão: de uma ontologia de domínio a uma aplicação de vídeo sob demanda*", Em: Projeto DAMD – *Design* de Aplicações Multimídia Distribuídas - Ed. Wanderley Lopes de Souza, 1999 (livro em fase de impressão).

[16] Pressman, R.S. *"Software Engineering: A practitioner's approach"*, 4th Edition, McGraw- Hill, 1997.