

Um Modelo Conceitual Hipermídia com Nós de Composição e Controle de Versões

Luiz Fernando G. Soares

Departamento de Informática, PUC-Rio
R. Marquês de São Vicente 225
22453 - Rio de Janeiro, RJ - Brasil
E-mail: lfgs@inf.puc-rio.br

Marco Antonio Casanova

Centro Científico Rio, IBM Brasil
Caixa Postal 4624
20001 - Rio de Janeiro, RJ - Brasil
E-mail: casanova@vnet.ibm.com

Noemi de La Rocque Rodriguez

NIC - RNP/CNPq
PUC-Rio
R. Marquês de São Vicente 225
22453 - Rio de Janeiro, RJ - Brasil
E-mail: noemi@inf.puc-rio.br

Resumo

O artigo descreve um modelo para a construção de documentos hipermídia que incorpora soluções para os problemas de organização do espaço de nós e de tratamento de versões. Este modelo está sendo usado como base para a implementação de uma ferramenta que poderá ser usada tanto para a construção de sistemas hipermídia monolíticos, distribuídos ou não, como para a introdução de características hipermídia em uma aplicação qualquer. O trabalho discute também, resumidamente, a arquitetura desta ferramenta, cuja estrutura em camadas possibilita o acesso a vários níveis de interface, além da obediência à proposta de padrão MHEG para intercâmbio de objetos multimídia e hipermídia.

1 - Introdução

Nos anos recentes, serviços multimídia têm sido incorporados em várias áreas, como sistemas de educação e treinamento, sistemas para automação de escritório, sistemas de informação e pontos de vendas, etc. Neste contexto, muitas aplicações multimídia serão construídas para plataformas heterogêneas, e muitas aplicações oferecerão seus serviços através da interconexão de plataformas diversas. Estes serviços usarão uma grande quantidade de objetos estruturados residentes em estações, armazenados em meios digitais, ou recuperados de fontes remotas através de redes. Estes objetos representarão um investimento significativo, tornando-se imperativo que esta informação permaneça disponível e não seja perdida pela incompatibilidade de estruturas de dados usadas nas diversas aplicações e, desta forma, que possam ser reusados.

Os sistemas hipermídia existentes foram desenvolvidos como uma aplicação única e auto-contida, fazendo com que sejam especializados e difíceis de serem reutilizados em outras aplicações. Muitos dos esforços empregados na construção de sistemas hipermídia seguiram esta direção, com exceções dignas de menção, como o NEPTUNE [DeSc86] e o Hypertext Abstract Machine (HAM) [CaGo88]. O sistema descrito neste artigo também se constitui em uma exceção, provendo não apenas um modelo conceitual de dados hipermídia, como uma arquitetura aberta, cujo modelo de interface separa os componentes de dados e de exibição dos objetos, o que permite a construção de interfaces independentes da plataforma final de exibição. Estes dois componentes combinados vão fornecer uma ferramenta capaz não só de possibilitar a construção de sistemas hipermídia monolíticos, como também de introduzir características hipermídia a uma aplicação qualquer. Eles vão permitir ainda que os mecanismos de armazenamento possam ser adaptados aos requisitos de eficiência e tamanho impostos por uma aplicação particular. A obediência à proposta de padrão MHEG permitirá ao sistema intercambiar seus objetos pelas mais diversas aplicações multimídia e hipermídia que obedeçam ao padrão além, é claro, de permitir a reutilização de objetos gerados por estas aplicações.

O presente artigo se concentra na apresentação do modelo conceitual de dados do sistema, chamado de Modelo de Contextos Aninhados (MCA), abordando apenas resumidamente os aspectos da arquitetura do sistema, que são objetos de outro artigo [SoCa93]. A próxima seção descreve o modelo básico, deixando para a seção 3 a descrição do modelo estendido com a incorporação dos mecanismos de controle de versões. A seção 4 apresenta resumidamente a arquitetura do sistema. A seção 5 é reservada às conclusões.

2 - O Modelo de Contextos Aninhados

A possibilidade de representar referências arbitrárias entre partes quaisquer de um documento representa, para muitos, a característica marcante de um sistema hipertexto. No entanto é necessário combinar esta flexibilidade com mecanismos de organização do documento, a fim de evitar o problema conhecido como "lost in hyperspace" [Hala88]; mecanismos para definição de diferentes visões de um mesmo documento e para organização hierárquica (linear ou não) de

documentos. Os modelos com nós de composição provêm suporte a tais mecanismos, em especial modelos que permitem o aninhamento de nós de composição.

A definição de documentos hipermídia no MCA é baseado nos conceitos usuais de nós e elos. *Nós* são fragmentos de informação e *elos* são usados para a interconexão de nós que mantêm alguma relação.

O modelo distingue ainda duas classes básicas de nós, chamados de nós terminais e nós de contexto, sendo este último o ponto central do modelo. Intuitivamente, um *nó terminal* contém dados cuja estrutura interna é dependente da aplicação e não é parte do modelo. A classe de nós terminais pode ser especializada em outras classes (texto, gráfico, áudio, vídeo, etc.) conforme requerido pelas aplicações.

Um *nó de contexto* é usado para agrupar conjunto de nós terminais ou de contextos, recursivamente. Nós de contexto podem ser aninhados em qualquer profundidade e assim permitem organizar, hierarquicamente ou não, conjunto de nós. Por exemplo, para organizar hierarquicamente um livro texto, pode-se primeiro definir um nó de contexto *B*, que contém um conjunto de nós representando capítulos do livro e um conjunto de elos indicando a organização dos capítulos, que não é necessariamente em sequência linear. De forma análoga, para o *i*-ésimo capítulo, pode-se definir o nó de contexto *C_i* que contém um conjunto de nós representando as seções do capítulo e um conjunto de elos indicando a organização das seções, e assim sucessivamente.

O conceito de nós contexto generaliza o conceito homônimo introduzido no sistema Neptune [DeSc86 e DeSc87], que por sua vez se baseou em algumas idéias do PIE [GoBo87]. Ele também generaliza as webs do sistema Intermedia [Meyr86] e as fileboxes do Notecard [Hala88].

Um elo conecta dois nós. Como o conteúdo de um nó possui uma estrutura interna, que pode ser bastante complexa, os elos também indicam, para cada extremidade, a região onde o elo está ancorado. Por exemplo, para nós texto, a região pode ser simplesmente uma cadeia de caracteres dentro do texto, para nós de imagem bidimensional, pode ser um retângulo determinado por um par de coordenadas, e para nós de contexto, pode especificar um deslocamento, conforme será definido nos próximos parágrafos.

A classe nó possui dois componentes especiais, CONTENTS e MASK. Dado um nó *N*, o valor de CONTENTS é chamado o conteúdo de *N* e depende da classe de *N* (se contexto, texto, áudio, vídeo, etc.). O valor de MASK é uma lista de regiões dentro do conteúdo de *N*, também dependente da classe de *N*. A lista de regiões de um nó funciona como uma interface externa do nó, no sentido que outros nós não se referirão diretamente a pontos dentro do conteúdo do nó, mas apenas indiretamente através de um índice da lista. Assim, qualquer mudança no conteúdo de um nó não se refletirá nos elos e, portanto, nos nós de contexto que contêm estes elos, se a lista de um nó for sempre atualizada adequadamente.

Pode-se definir agora, mais precisamente, âncora, elo e nó de contexto.

Uma âncora é um par (N_0, s_0) , onde N_0 é um nó terminal ou de contexto, e s é um deslocamento definido como:

- o valor nulo λ ; ou
- um identificador de uma entrada na lista MASK de N_0 ; ou
- um par (N_1, s_1) , onde N_0 é um nó de contexto que contém N_1 e s_1 é um deslocamento.

N_0 é chamada de base da âncora e s_0 seu deslocamento.

Um elo inclui um par (s, d) de âncoras, onde s é a âncora de origem e d a âncora de destino. As extremidades do elo são as bases de s e d .

A cada elo são associadas condições e ações, em conformidade com a proposta de padrão MHEG [MHEG92]. Generalizando, uma vez que o modelo de contextos aninhados engloba todos os objetos em conformidade com a proposta de padrão MHEG, ele também vai englobar todas as características e definições destes objetos, como por exemplo os mecanismos de sincronização definidos nos elos, etc. Como extensão ao padrão MHEG, a cada entrada da lista MASK também são associadas condições e ações [CCRS93].

O conteúdo de um nó de contexto N é um par da forma (S, L) , onde S é um conjunto de nós e L é um conjunto de elos, tais que suas extremidades estão contidas em S . Diz-se que N contém cada nó em S e cada elo em L . Cada entrada do componente MASK de um nó de contexto N é um conjunto de nós pertencentes a S , ou uma âncora cuja base está contida em N .

Por exemplo, seja A um nó de contexto que contém um outro nó de contexto B que por sua vez contém dois nós, C e D . Como B contém C e D , um elo conectando estes nós pode, em princípio, ser definido em B como $((C, i), (D, j))$, onde i e j são deslocamentos válidos para C e D , respectivamente. Se se quer criar um elo em A conectando C e D , ele deve ser definido como $((B, m), (B, n))$. Neste caso, m e n identificam entradas na lista MASK de B , cujos conteúdos são iguais a (C, i) e (D, j) ; ou, então, especificam diretamente os deslocamentos (C, i) e (D, j) , respectivamente.

Continuando as definições, uma hiperbase é qualquer conjunto H de nós, tais que, para qualquer nó $N \in H$, se N é um nó de contexto, então todos os nós contidos em N também pertencem à hiperbase H .

Uma vez que o modelo permite o mesmo nó esteja presente em diferentes contextos, e que contextos sejam aninhados em qualquer profundidade, é necessário que exista uma forma de identificar sob que sequência de contextos aninhados um dado nó está sendo pesquisado e que elos de fato tocam este nó a partir do aninhamento. Estas noções são capturadas pela noção de perspectiva de um nó e a noção de elos visíveis por um nó a partir de uma perspectiva.

A perspectiva de um nó N é uma sequência $P = (N_1, \dots, N_m)$, $m \geq 1$, tal que $N_1 = N$, N_{i+1} é um contexto e N_i está contido em N_{i+1} , para $i \in [1, m)$. Como N é implicitamente dado por P , P é referido simplesmente como perspectiva. Um nó M está presente em uma perspectiva $P =$

(N_1, \dots, N_m) se e somente se $M = N_i$ para algum $i \in [1, m]$. Note assim que podem existir várias perspectivas diferentes para um mesmo nó N , se este nó estiver presente em mais de um contexto. Chama-se perspectiva corrente a perspectiva usada para alcançar o nó sobre o qual se está trabalhando em determinado momento.

Seja N_1 um nó e $P = (N_1, \dots, N_m)$ uma perspectiva de N_1 . Diz-se que um elo l é visível a partir de P por N_1 com deslocamento i_1 , se e somente se:

- l está contido em N_2 e (N_1, i_1) é uma âncora de l ; ou
- l é visível a partir de P por N_2 com deslocamento i_2 e o deslocamento i_2 é (N_1, i_1)

Um elo é visível a partir de P por N , se e somente se ele é visível a partir de P por N para algum deslocamento.

O modelo de contextos aninhados, tal como definido acima, provê suporte imediato para o tratamento de algumas das noções de versão; deixa-se estas considerações, no entanto, para a próxima seção.

3 - Tratamento de Versões no Modelo de Contextos Aninhados

Esta seção propõe um conjunto de mecanismos para tratamento de versões em sistemas hipermídia. Em [Hala88 e Hala91], a falta de suportes ao tratamento de versões é colocada por Halasz como uma falha dos sistemas de hipertexto propostos até então. Podemos usar estes textos como base para estabelecermos o que se entende por, ou talvez os requisitos de, um mecanismo de versão. Vários significados diferentes podem ser identificados naquele trabalho; enumeramos abaixo estes significados, transcrevendo para cada um a frase original onde ele é colocado.

1. exploração de alternativas --- *"A good versioning mechanism will also allow users to simultaneously explore several alternate configurations for a single network."*
2. gerência de configurações --- *"In a software engineering context it should be possible to search for either the version that implements Feature X or the set of changes that implement Feature X."*
3. manutenção do histórico de um documento --- *"A good versioning mechanism will allow users to maintain and manipulate a history of changes to their network."*
4. atualização automática de referências --- *"In particular, a reference to an entity may refer to a specific version of that entity, to the newest version of that entity, to the newest version of that entity along a specific branch of the version graph, or to the (latest) version of the entity that matches some particular description (query)."*
5. manipulação de um conjunto único de versões --- *"Although maintaining a version thread for each individual entity is necessary, it is not a complete versioning mechanism. In general, users will make coordinated changes to a number of entities in the network at one time. The*

developer may then want to collect the resultant individual versions into a single version set for future reference."

Além dos pontos abordados acima, discutiremos neste trabalho ainda dois outros, que julgamos pertinentes na definição de um sistema distribuído para tratamento de documentos multimídia. Em primeiro lugar, é importante que um tal sistema tenha como captar a noção de que diferentes representações (em caso particular, diferentes mídias) podem ser usadas para descrever uma mesma informação (como, por exemplo, a versão escrita e falada de uma palestra). Representações diferentes de uma mesma informação devem ser tratadas como diferentes versões desta informação.

O conceito de versão também é útil para modelar o uso simultâneo de uma informação por vários indivíduos. Cada uma das cópias da informação em uso em determinado instante é considerada uma versão desta informação, dando mais um sentido ao termo versão. O acoplamento deste mecanismo de versão com o mecanismo de notificação vai proporcionar um bom suporte a trabalhos cooperativos. Na verdade este caso é uma generalização do caso anterior, onde não só tem-se representações diferentes da mesma informação, mas a informação deve ser compartilhada para uso simultâneo.

Assim, completando a lista acima, temos ainda:

6. gerência de representações de uma mesma informação.
7. utilizações concorrentes de uma mesma informação.

O Sistema para Tratamento de Versões que aqui propomos é adequado a qualquer sistema hipermídia com nós de composição com aninhamento, em particular àqueles baseados na proposta de padrão MHEG, caso do MCA. O modelo de contextos aninhados, tal como definido na seção 2, provê suporte imediato para o tratamento dos requisitos 1 e 2, conforme apresentado em 3.1. Os requisitos 3 e 4 colocados por Halasz necessitam de um tratamento mais complexo, que será descrito na seção 3.2; esta seção descreve uma extensão ao MCA, introduzindo o conceito de contexto de versões, útil também no suporte à noção de versões como representações de uma mesma informação, requisito 6. O requisito 5 é abordado na seção 3.5, enquanto 3.3 e 3.4 introduzem a noção de estado de uma versão e hiperbase pública, necessário ao tratamento do problema levantado pelo requisito 7.

3.1 - Exploração de Alternativas e Gerência de Configurações

A solução para o problema de exploração de alternativas é trivial em qualquer modelo com nós de composição aninhados, como o MCA, bastando criar contextos distintos que correspondam às diferentes alternativas estudadas. Os nós de contexto vão possibilitar, assim, um mecanismo para a definição de diferentes visões de um mesmo documento, sintonizadas em diferentes aplicações ou classes de usuários.

Voltando ao exemplo da organização de um livro em capítulos e seções apresentado em parágrafo anterior, o autor pode definir organizações diferentes para os capítulos, para classes distintas de leitores, definindo outros nós de contexto contendo os mesmos nós de B (os mesmos capítulos), mas com um conjunto diferente de elos indicando as novas organizações. Cada um destes novos contextos podem ser encarados como uma visão diferente do mesmo livro. Note que ambos compartilham uma única cópia dos nós C_i . Para representar o fato destes diferentes contextos serem alternativas do mesmo documento, eles podem ser incluídos em um contexto mais externo, *livro texto*.

Uma solução similar pode ser adotada para o problema de gerência de configurações de software. Cada "release" de um sistema pode ser representado através de um contexto. Este contexto conterá as conexões entre as versões de componentes usadas neste release. Estas conexões podem representar relações como dependência entre objeto e fonte, inclusão, importação, arquivos teste utilizados, documentação relevante, etc. Simultaneamente, as várias versões de uma mesma componente podem ser agrupadas em um *contexto de versões*, conceito que será discutido mais adiante. Estes contextos de versões podem, por sua vez, ser agrupados em contextos, por exemplo, *bibliotecas de entrada e saída*. A identificação entre configuração e contexto é bastante feliz, pois, assim como um contexto, uma configuração compartilha com outras seus componentes, mas as ligações entre estes componentes é particular a cada configuração.

3.2 - Navegação pelo Histórico de um Documento e Atualização Automática

A manutenção do histórico de um documento é capturada no MCA introduzindo uma nova classe de contextos chamados contextos de versões. Um contexto de versões, assim como um contexto comum, agrupa um conjunto de nós e elos, mas este agrupamento tem, no caso de contextos de versões, uma semântica bem definida.

Os nós contidos em um contexto de versões não precisam ser da mesma classe e são chamados nós correlatos. Por exemplo, um usuário pode criar em um mesmo contexto de versões dois nós, T e S , que descrevem o mesmo pedaço de texto, exceto que T é uma versão escrita e S uma versão falada (o que será bastante útil no atendimento ao requisito 6). Em casos como este, o usuário é quem deve fazer a inclusão de um nó em um contexto de versões explicitamente, pois o sistema não tem como deduzir que S e T são versões de uma mesma informação.

Um elo $((v_1, i_1), (v_2, i_2))$ no contexto de versões V indica que o conteúdo do nó v_2 foi criado a partir do conteúdo do nó v_1 . Diz-se neste caso que v_1 é o ascendente de v_2 e que v_2 é descendente de v_1 . Os deslocamentos, neste caso, simplesmente tornam mais precisa a informação sobre qual parte do conteúdo de v_1 gerou que parte de v_2 . Quando o usuário cria um novo nó a partir de um nó N já existente, o sistema pode deduzir que se trata de uma nova versão de N e inserir este novo nó no contexto de versões de N , assim como um elo entre N e o novo nó.

O conteúdo de uma versão pode ser construído a partir do conteúdo de várias outras versões, mas não é permitida a derivação cíclica.

Com esta extensão, pode-se navegar através do histórico de um documento usando-se exatamente os mesmos mecanismos de navegação oferecidos para contextos comuns [SCCL92], permitindo a criação de uma interface homogênea com o usuário no caso de um contexto qualquer e de um contexto de versões. A diferença é que, no caso de contextos de versões, o agrupamento tem uma semântica, não existente em um contexto qualquer. O fato de dois nós estarem em um mesmo contexto de versões indica que eles representam a mesma informação, em algum nível de abstração, sem implicar que um seja direta ou indiretamente derivado do outro. Se existe uma ligação entre os dois nós, indica-se adicionalmente que o conteúdo de uma deles foi derivado a partir do outro.

Para utilizar o mecanismo de versões e a facilidade de atualização automática de referências, os contextos que formam um documento podem, ao invés de incluírem diretamente os nós desejados, incluir contextos de versões deste nós. Supõe-se que existe sempre, dentro de um contexto de versões, uma versão privilegiada, denominada versão corrente. Pode-se assim definir, por exemplo, um procedimento de exibição padrão de um contexto de versões, que dispare a exibição da versão corrente. A definição do que vem a ser a versão corrente pode variar de aplicação para aplicação: pode ser a última criada, pode ser a última criada por um determinado usuário, pode ser a última criada a partir de uma dada versão, etc. Esta noção pode ser capturada pelo componente MASK do nó de contexto de versões, conforme definido na seção 2. Neste caso MASK funciona como uma interface externa do nó, no sentido que outros nós não se referirão diretamente a pontos dentro do conteúdo do contexto de versão, mas apenas indiretamente através de um índice da lista, que por sua vez conterá, em cada posição, uma definição de versão corrente.

Uma aplicação pode então usar a noção de contextos de versões para manter o histórico de um documento d (requisito 3), como também para atualização automática de referências (requisito 4), por exemplo, como a seguir. Suponha que d tenha um componente c cujas versões o sistema esteja interessado. Seja C o contexto de versões contendo os nós C_1, \dots, C_n que representam as versões de c . A aplicação deve então incluir C , e não diretamente qualquer das versões C_i 's, no nó de contexto D usado para modelar d . O deslocamento de todos os elos em D que tocam em C vão apontar para entradas da lista MASK de C , que por sua vez serão mantidas pela aplicação apontando para os nós C_i considerados como a versão corrente. Se a aplicação quiser recuperar versões anteriores de d com respeito a c , ela pode simplesmente navegar dentro de C , usando os mesmos mecanismos de navegação presentes em qualquer nó de contexto [Casa91, Soar93]).

A lista MASK de um contexto de versão deve ser atualizada a cada introdução de uma modificação no contexto de versão, ou então, preferencialmente, conter estruturas virtuais.

O modelo de contextos aninhados permite extensões para acomodar a noção de estruturas virtuais (nós, elos e regiões), isto é, estruturas resultantes da avaliação de alguma expressão. Um nó virtual tem o seu conteúdo calculado no momento em que ele é selecionado, durante o processo de navegação em um documento. Como exemplo [Hala88], um usuário poderia definir um contexto virtual para, a cada consulta sua a um documento, apresentar todos os nós adicionados a este documento desde sua última consulta.

Um elo virtual teria a sua extremidade destino calculada no instante em que o usuário o ativasse. Para tal, tanto as ações associadas a um elo poderiam estar associadas a avaliação de uma expressão, quanto uma entrada da lista MASK de um nó poderia conter uma expressão que seria avaliada no momento de sua ativação. Esta última facilidade é exatamente aquela que se precisa para a definição da versão corrente em um contexto de versão, avaliada de acordo com alguma expressão, por exemplo, o último nó criado por fulano.

A noção de estrutura virtual só é possível se o sistema tem um suporte de um bom mecanismo de consulta sobre a rede hipermídia. A definição dos componentes em uma estrutura virtual são de fato *queries*. Por ainda não dispor de um mecanismo eficiente de consulta, estruturas virtuais ainda não foram incluídas na implementação corrente do MCA, e nem discutidas como seriam representadas dentro da estrutura MHEG, fazendo parte de nossa intenção futura. Desta forma, na implementação corrente, a lista MASK de um contexto de versão que especifica as várias definições das versões corrente é atualizada a cada modificação do contexto de versão. Assim, o suporte à atualização automática de referências requisitado por Halasz é satisfeito.

Uma outra possibilidade interessante, mas ainda não muito estudada, é a definição de tipos especiais de elos que incluam informação sobre o processo de derivação de uma versão, permitindo que o histórico contenha mais informação além do grafo de derivação de versões.

3.3 - Consistência

Uma vez que se considera que qualquer acesso a uma informação cria uma versão desta informação, é necessário ter cuidado com a consistência de dados e interconexões entre estes. Para controlar este aspecto, define-se a noção de estado de um nó.

Um nó pode estar em um dos seguintes estados: temporário, permanente e obsoleto. Um nó ao ser criado assume o estado temporário e permanece neste estado enquanto sofre modificações. Quando se torna estável, um nó pode ser promovido ao estado permanente, quer por um pedido explícito do usuário, quer implicitamente por certas operações oferecidas pelo modelo [SCCL92]. Um nó no estado permanente não pode ser destruído diretamente, mas o usuário pode torná-lo obsoleto, dando a oportunidade para que outros nós que façam referências a ele ou dele derivados possam ser notificados.

Um nó no estado permanente, chamado nó permanente, tem as seguintes características:

- não pode ser modificado;
- se for um nó contexto, só pode conter nós permanentes ou obsoletos;
- pode ser usado na derivação de novos nós (versões);
- não pode ser destruído diretamente;
- pode ser tornado obsoleto, mas não temporário.

Um nó no estado temporário, chamado nó temporário, tem as seguintes características:

- pode ser modificado;
- pode conter nós em qualquer estado, se for um nó de contexto;

- não pode ser usado para derivar novos nós (versões);
- pode ser destruído diretamente;
- pode se tornar permanente, mas não obsoleto.

Um nó no estado obsoleto, chamado nó obsoleto, tem as seguintes características:

- não pode ser modificado;
- se for um nó contexto, só pode conter nós permanentes ou obsoletos;
- não pode ser usado na derivação de novos nós (versões);
- é automaticamente destruído pelo sistema, por algum mecanismo de coleta de lixo, quando não é mais necessário;
- não pode mudar de estado.

Como consequência das definições, os ascendentes de um nó no grafo de versões são nós permanentes ou obsoletos. É também digno de menção que estas restrições garantem que um nó de contexto permanente ou obsoleto contém apenas nós permanentes ou obsoletos, o que por sua vez implica que todos os seus elos têm como base apenas nós permanentes ou obsoletos. Porém, estas propriedades não se aplicam a nós de contextos temporários.

3.4 - Cooperação e Intercâmbio

Em um ambiente onde se deseja que vários usuários trabalhem de forma cooperativa, é necessário que estes possam compartilhar informação. Por outro lado, também é importante permitir que cada usuário tenha informações não compartilhadas, não só como proteção aos seus dados como para não onerar todos os usuários com a navegação em um espaço grande demais de informações. Trabalho cooperativo é entendido aqui como o processo de criação ou modificação de uma hiperbase, ou um subconjunto de uma hiperbase, por um grupo de usuários.

Define-se, para organizar o espaço de nós, uma partição deste espaço, onde cada subconjunto é denominado uma base de nós. Um e apenas um entre estes subconjuntos é designado por hiperbase pública e, como toda hiperbase, todos os nós que estão presentes em algum contexto neste subconjunto também devem pertencer a ele. Outra restrição da hiperbase pública é que todos os seus nós estejam no estado permanente ou obsoletos. Todas as demais bases de nós são chamadas bases privadas. É importante salientar que na partição do espaço de nós, os nós de contexto de versões são deixados a parte e não pertencem nem à hiperbase pública nem a qualquer base privada.

Um nó na hiperbase pública não pode migrar desta hiperbase para uma base privada, embora possam ser criadas versões destes nós nas diversas bases privadas. A recíproca no entanto é verdadeira; nós podem ser movidos de uma base privada qualquer para a hiperbase pública, desde que sejam tomados antes permanente.

Se após a criação de uma versão V de um nó N da hiperbase pública em uma base privada esta nova versão não sofrer qualquer modificação, ao se passar a versão criada para a hiperbase

pública, ela é simplesmente destruída, pois não há a necessidade de replicação de nós, e os contextos das bases privadas que contém V são atualizados para conterem agora N, ao invés de V.

Resta ainda discutir o que acontece quando se cria uma nova versão de um contexto da hiperbase pública em uma base privada. Neste caso só são criadas versões dos nós presentes neste contexto quando estes nós são visitados. Como exemplo, considere que determinado documento na hiperbase pública contém um contexto C e que este contexto contém nós I , H e M . Se, durante uma sessão de trabalho sobre este documento, em uma base privada qualquer, o contexto C é selecionado, automaticamente será criada uma versão temporária C_1 deste contexto na base privada, mas os nós presentes nele serão, a princípio, os mesmos I , H e M , e não novas versões destes nós. Se o nó H for por sua vez selecionado, neste instante então será criada uma nova versão H_1 , na mesma base privada, que substituirá H em C_1 , conforme discussão acima. Se o nó M não for manipulado durante esta sessão de trabalho, a versão deste nó presente em C_1 permanecerá a original.

3.5 - Propagação de Versões

Quando se cria uma nova versão de um nó pode-se supor a criação de novas versões de todos os contextos onde este nó está presente. Por outro lado, quando se cria uma nova versão de um contexto, também pode-se pensar na criação de novas versões de todos os nós nele presente. A criação automática de novas versões em situações como estas é chamada de propagação de versões.

Apesar de útil em alguns casos, a propagação ilimitada de versões implica em uma proliferação de nós muitas vezes indesejada. A figura 1 mostra um exemplo de proliferação de versões como resultado de propagação automática exaustiva. Na figura 1(a), $D0$ é um nó pertencente aos contextos $E0$, $B0$ e $F0$. $E0$ e $B0$ por sua vez estão contidos nos contextos $C0$ e $A0$, respectivamente. A criação de uma versão $D1$ de $D0$, na figura 1(b), gera por propagação as demais versões da figura.

No MCA, segundo o comando do usuário, a propagação de versão ou não é realizada, ou então é realizada ao longo apenas da perspectiva corrente, ou seja, a perspectiva usada para alcançar o nó no momento de criação de sua versão. A figura 2 mostra o mesmo exemplo da figura 1, agora utilizando-se a propagação apenas ao longo da perspectiva corrente ($D0$, $B0$, $D0$)

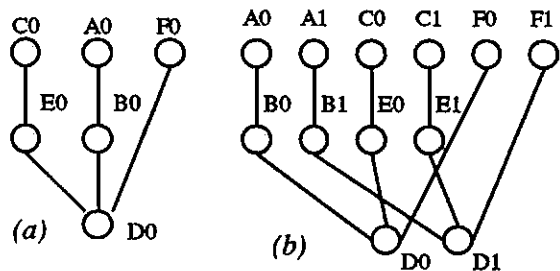


Figura 1 - Proliferação de versões ao longo de todas as perspectivas

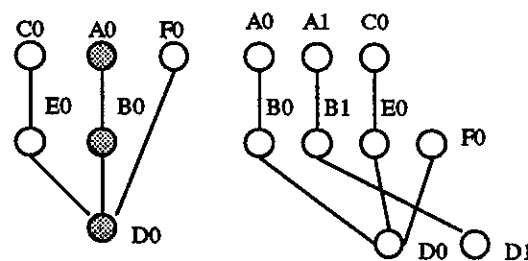


Figura 2- Propagação ao longo da perspectiva corrente.

Ainda que se limite a propagação à perspectiva corrente, a proliferação de novas versões ainda representa um problema grave, como discutido em [WoKL86] e em [Hala88]. É necessário se poder manipular um conjunto de versões, atendendo ao requisito 5 colocado na seção 1. Este conceito, presente em [GoBo87], é incorporado ao MCA. A estratégia adotada leva em consideração o estado das versões presentes na perspectiva corrente. Definiu-se na seção 3.2 que, enquanto determinado nó representar uma versão temporária, ele não pode ser usado para derivar uma nova versão. Assim, coerente com a definição anterior, se um nó N na perspectiva corrente estiver no estado temporário, a propagação de versão ao invés de criar uma nova versão, simplesmente altera o conteúdo de N . A figura 3 ilustra bem o problema da propagação de versão na perspectiva corrente quando da alteração do conteúdo de dois nós, $C0$ e depois $D0$.

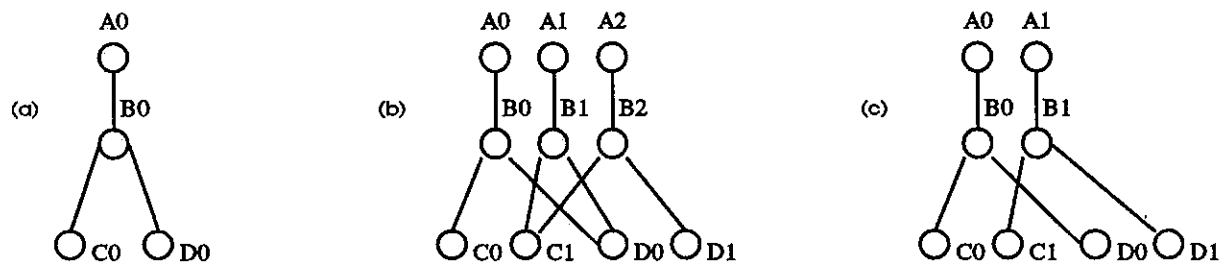


Figura 3 - Propagação de versão após a alteração, em (a), de $C0$ e depois $D0$. (b) Propagação sem levar em conta o estado das versões. (c) Propagação levando-se em conta que apenas os estados de $A0$, $B0$, $C0$ e $D0$ são permanentes.

Estabelecendo-se como regra que a criação de uma nova versão não é propagada quando o contexto onde a versão original está presente ainda é uma versão temporária, mantém-se compatibilidade com o tratamento de estado definido anteriormente, e ganha-se uma ferramenta para o agrupamento de modificações, atendendo o requisito 5.

O trabalho em [Maio93] discute a granularidade com que versões devem ser criadas. Argumenta-se que a carga cognitiva relativa à criação explícita de versões pelo usuário é grande demais, e que após cada sessão de trabalho em um documento deve automaticamente ser criada uma nova versão deste. É interessante observar que a solução descrita acima, onde a propagação de versões depende do estado dos contextos na perspectiva corrente, oferece a noção de versão de documento (ao invés de simplesmente versão de um nó). No MCA, um documento é simplesmente um contexto. Assim, uma sessão de trabalho em um documento tipicamente consiste em criar uma nova versão, temporária, do contexto correspondente, e transformá-la em permanente ao se considerar que o documento alcançou um estado consistente. O conjunto de alterações feitas durante uma sessão de trabalho em um contexto corresponde à criação de uma nova versão deste contexto, além de implicar na criação de novas versões dos nós alterados, cujo histórico poderá ser examinado a qualquer instante, como visto anteriormente.

4 - Arquitetura Hiperfídia em Camadas

Nesta seão   apresentada uma arquitetura em camadas para sistemas hiperfídia e   discutido como esta arquitetura se relaciona com os conceitos do modelo de contextos aninhados apresentados nas seões anteriores.

Diversos requisitos nortearam o desenvolvimento do modelo e da estrutura hiperfídia em camadas:

- O sistema deve operar independente da plataforma: tipo de m quina, sistema operacional, dispositivos de exibição, ferramentas e redes.
- O sistema deve ser independente das aplicaões e poder ser estendido para utilizaão por diversas aplicaões.
- O sistema deve prover uma arquitetura aberta, permitindo  s aplicaões diversos n veis de interface.
- O n mero t pico de usu rios   muito grande.
- Os dados s o objetos longos espalhados em v rias m quinas.
- O sistema n o tem conhecimento do conte do dos dados armazenados, exceto de seu tipo.
- Os objetos intercambi veis devem seguir um padr o internacional, possibilitando n o s  a utilizaão de seus objetos em outros sistemas, mas a utilizaão de objetos gerados por outros sistemas.
- O sistema deve suportar interatividade em tempo real, incluindo a aquisião de dados multim dia.

A implementaão de sistemas hiperfídia multi-usu rios baseados em uma  nica estaão de trabalho dificilmente vai permitir a apresentaão de  udio e v deo em tempo real, devido ao tamanho dos objetos multim dia, que os obrigam a ficarem residentes em dispositivos de armazenamento com pequena faixa passante de entrada e sa da. O requisito de tempo real vai exigir que a arquitetura hiperfídia possa ser implementada em um sistema distribu do.

A exig ncia de uma implementaão distribu da vem n o apenas do fato mencionado acima, como t m do fato que muitas aplicaões ser o desenvolvidas para estaões heterog neas que ser o interconectadas para oferecerem servios multim dia, como por exemplo, trabalho cooperativo, sistemas de mensagem multim dia, videofonia, publicaões e livros eletr nicos, sistemas telem ticos audiovisuais para treinamento e educaão, jogos e simulaão, pontos de vendas,  m de novas classes de aplicaão multim dia.

A Arquitetura hiperfídia possui tr s camadas e quatro interfaces [SoCC93], como pode ser visto na figura 4.

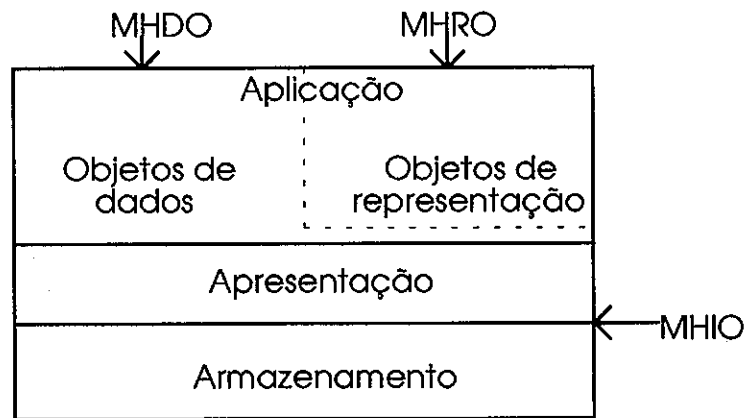


Figura 4 - Estrutura Hipermédia em Camadas

A Camada de Armazenamento oferece o armazenamento persistente para objetos multimídia, que contém um identificador único, um tipo específico, além de outros atributos, e podem ser compartilhados por várias aplicações. Esta camada pode ser parcialmente implementada usando qualquer sistema de banco de dados comercialmente disponível, ou um servidor de banco de dados em uma implementação distribuída [SoCa93]. Ela oferece uma interface para o intercâmbio de dados hipermédia chamada *multimedia hypermedia interchangeable objects interface* (MHIO Interface).

A Interface MHIO (Multimedia Hypermedia Interchangeable Objects) é a chave para prover a compatibilidade entre aplicações e equipamentos, uma vez que ela estabelece dois pontos aos quais as camadas de apresentação e armazenamento devem obedecer: (1) a representação codificada para os objetos multimídia intercambiáveis, que corresponde à proposta de padrão ISO MHEG [MHEG92]; e (2) as mensagens, requisições, confirmações, etc., usado por estas camadas. Estes dois pontos são objeto da série de recomendações T.170 da CCITT, que deverá incluir o padrão MHEG como sua recomendação T.171.

Os componentes do modelo de dados da Camada de Aplicação são os objetos usuais de um sistema hipermédia, isto é, nós e elos. Esta camada introduz duas novas idéias na arquitetura de aplicações interativas, parecidas com as encontradas em [PuGu91]. A primeira idéia é a definição de objeto de dados, criados como um novo objeto, ou como uma instância local de objetos de armazenamento, uma versão destes objetos, com novos atributos, dependentes da aplicação. Ao contrário dos objetos de armazenamento, que contêm apenas atributos, os objetos de dados contêm métodos para manipular os novos atributos, além de métodos para manipulação da informação originalmente pertencente ao objeto de armazenamento, se é o caso. A segunda idéia diz respeito à representação dos objetos de dados. Ela consta de uma abstração de nível ainda mais alto do objeto multimídia, criando um objeto de representação, cuja classe é uma especialização da classe objeto de dados que incorpora novos métodos para exibição do atributo conteúdo de dados multimídia, além de converter o atributo conteúdo de dados para um formato mais apropriado, conforme definido pelos métodos que o manipulam. Um objeto de representação

é, desta forma, uma nova versão de um objeto de armazenamento, derivada de um objeto de dados, como definido no requisito 6 da seção 1.

As aplicações que desejam ter incorporadas características hipermídia, ou visam a construção de sistemas hipermídia monolíticos, podem fazer uso das interfaces MHRO (Multimedia Hypermedia Representation Object) ou MHDO (Multimedia Hypermedia Data Object), além da já mencionada MHIO.

Cabe à Camada de Apresentação a conversão do formato de dados dos objetos usados por uma aplicação e plataforma particular para o formato padrão definido pela interface MHIO, e vice versa. Deve-se ressaltar que esta camada não implementa qualquer método associado com os objetos de dados.

A Interface de Apresentação define a codificação dos objetos trocados pelas camadas de apresentação e de aplicação. A princípio, os objetos desta interface não são intercambiáveis entre aplicações diferentes e, embora não haja uma restrição explícita, não é esperado que uma aplicação faça uso dos objetos definidos por esta interface.

Resumidamente, para ganhar acesso a um dado multimídia, uma aplicação deve, de forma simplificada, proceder como se segue. Através de facilidades navegacionais do sistema hipermídia, ela identifica indiretamente o objeto persistente que contém o dado desejado; a partir de então ela requisita a criação de um objeto de representação correspondente.

Do ponto de vista da arquitetura em camadas, a noção de hiperbase pública e bases privadas deve ser entendida da seguinte forma. Em geral, todos os nós manipulados pelas diversas camadas correspondem a nós em bases diferentes. A camada de armazenamento é responsável pelos nós da hiperbase pública, e pelos nós contexto de versões, ao passo que a camada de apresentação cria nós nas bases privadas da camada de aplicação e move dados das bases privadas para a hiperbase pública.

5 - Conclusão

O modelo de contextos aninhados com suporte a versões é a base conceitual de uma máquina de hipermídia em desenvolvimento conjunto pela IBM Brasil e Pontifícia Universidade Católica do Rio de Janeiro. O objetivo deste projeto é fornecer uma ferramenta para construção de aplicações específicas de tratamento de documentos. Julgou-se, assim, que a melhor forma de oferecer esta ferramenta seria sob a forma de uma biblioteca de tipos e classes em uma linguagem orientada por objetos. A linguagem escolhida para esta implementação foi C++ por sua disponibilidade em um grande número de plataformas.

Um ponto do modelo cuja implementação é bastante simplificada em uma LOO é a provisão de nós e elos virtuais. Pela falta de uma linguagem de consulta adequada, a implementação corrente não possui estruturas virtuais. De fato, sem um bom mecanismo de consulta é impossível um bom mecanismo de estruturas virtuais, e sem estes mecanismos um sistema de tratamento de versões

não é completo. Estes são os principais pontos que pretendemos atacar na próxima extensão do modelo.

Suporte a trabalho cooperativo sempre foi um dos requisitos do projeto, que começou com o tratamento de documentos em um sistema de teleconferência. Suporte a trabalho cooperativo implica entre outras coisas no controle de notificação. Mecanismos para controle de notificação devem também fazer parte da próxima extensão do modelo.

Resta mencionar ao fim deste artigo que os mecanismos para tratamento de versões aqui apresentados, embora tenham tomado como exemplo o modelo de contexto aninhado, podem ser adaptados, sem muitos problemas, a qualquer modelo que contenha nós de composição que possam ser aninhados.

O desenvolvimento do modelo de contextos aninhados foi baseado na idéia que as aplicações hipermídia são uma especialização de um conjunto maior de aplicações. O modelo fornece uma ferramenta capaz não só de possibilitar a construção de sistemas hipermídia monolíticos, como também de introduzir características hipermídia a uma aplicação qualquer. A estruturação da ferramenta em camadas possibilita também a flexibilidade adicional de se oferecer diferentes níveis de interface, utilizadas pelas aplicações conforme suas necessidades. A definição de uma estrutura em camadas permite ainda que os mecanismos de armazenamento possam ser adaptados aos requisitos de eficiência e tamanho impostos por uma aplicação particular. A obediência à proposta de padrão MHEG faz com que todos os objetos gerados possam ser intercambiados pelas mais diversas aplicações multimídia e hipermídia que obedeçam ao padrão e, mais ainda, permite a utilização de objetos gerados por estas aplicações.

Agradecimentos: Os autores gostariam de agradecer a Sérgio Colcher, Paulo Jucá, Thaís Batista, Guido Sousa e Maria Júlia Lima, que muito contribuíram na discussão das idéias aqui apresentadas. O trabalho árduo de implementação realizado por estas pessoas e por outros membros do grupo, aos quais também gostaríamos de agradecer, tem tornado possível o refinamento constante do modelo apresentado.

Referências Bibliográficas

- [CaGo88] Campbell, B.; Goodman, J.M. "HAM: A General Purpose Hypertext Abstract Machine". *Communications of the ACM*, Vol. 31, No. 7. Julho de 1988.
- [Casa91] Casanova, M.A.; Tucherman, L.; Lima, M.J.; Rangel Netto, J.L. Rodriguez, N.L.R.; Soares, L.F.G. "The Nested Context Model for Hyperdocuments". *Proceedings of Hypertext '91*. Texas. Dezembro de 1991.
- [CCRS93] Casanova, M.A.; Colcher, S.; Rodriguez, N.L.R.; Soares, L.F.G. "Ancoragem, Estruturas Virtuais e Consulta em Objetos MHEG". *Relatório Técnico CCR -Rio, IBM Brasil*. Em elaboração.
- [DeSc86] Delisle, N.; Schwartz, M. "Neptune: A Hypertext System for CAD Applications". *Proceedings of ACM SIGMOD '86*. Washington, D.C. Maio de 1986.

- [DeSc87] Delisle, N.; Schwartz, M. "Context - A Partitioning Concept for Hypertext". *Proceedings of Computer Supporteed Cooperative Work*. Dezembro de 1986
- [GoBo87] Goldstein, I.; Bobrow, D. "A Layered Approach to Software Design". *Interactive Programming Environments*. McGraw Hill, pag. 387-413. Nova York. 1987.
- [Hala88] Halasz, F.G. "Reflexions on Notecards: Seven Issues for the Next Generation of Hypermedia Systems". *Communications of ACM*, Vol.31, No. 7. Julho de 1988.
- [Hala91] Halasz, F.G. "Seven Issues Revisited". *Final Keynote Talk at the 3rd ACM Conference on Hypertext*. San Antonio, Texas. Dezembro de 1991.
- [Maio93] Maioli, C; Sola, S.; Vitali, F. "Versioning issues in a Collaborative Distributed Hypertext System". *Technical Report UBLCS-93-6*, Laboratory for Computer Science, Universidade de Bolonha, Itália, 1993.
- [Meyr86] Meyrowitz, N. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework". *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*. Portland, Oregon. Setembro de 1986.
- [MHEG92] MHEG. "Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects - Part1: Base Notation. *Working Document S.7. ISO/IEC JTC1/SC29/WG12*. Novembro de 1992.
- [PuGu91] Puttress, J.J.; Guimarães, N.M. "The Toolkit Approach to Hypermedia". 1991.
- [SCCL92] Soares, L.F.G.; Casanova, M.A.; Cavalcanti, M.R.; Lima, M.J.D. "Versions in the Nested Context Hypermedia Model. *Relatório Técnico Departamento de Informática, PUC-Rio*. Agosto de 1992.
- [SoCa93] Soares, L.F.G.; Casanova. "Modelo de Contextos ANinhados com Intercâmbio de Objetos MHEG em Arquiteturas Distribuídas". *Anais do XI Simpósio Brasileiro de Redes de Computadores*. Campinas, São Paulo. Maio de 1993.
- [Soar93] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.L.R. "Tratamento de Versões em um Modelo Conceitual Hipermídia com Nós de Composição". *Relatório Técnico PUC-Rio - Departamento de Informática*. Rio de Janeiro. Maio de 1993.
- [SoCC93] Soares, L.F.G.; Casanova, M.A.; Colcher, S. "An Architecture for Hypermedia Systems Using MHEG Standard Objects Interchange". *Proceedings of the Workshop on Hypermedia and Hypertext Standards*. Amsterdam, The Netherlands. Abril de 1993.
- [WoKL86] Woelk D.; Kim W.; Luther W. "An Object-Oriented Approach to Multimedia Databases". *Proceedings of the ACM SIGMOD Conference on Management of Data*. Washington D.C. May 1986, 311-325.