Propagação e Proliferação de Versões em Modelos Conceituais Hipermídia com Composições

Luiz Fernando G. Soares* Guido L. de Souza Filho+ Rogério F. Rodrigues* Débora Muchaluat*
lfgs@inf.puc-rio.br guido@dimap.ufrn.br rogerio@telemidia.puc-rio.br debora@telemidia.puc-rio.br

*Depto. de Informática, PUC-Rio
R. Marquês de São Vicente 225
22453-900 - Rio de Janeiro, Brasil

+DIMAp, UFRN
Campus Universitário, Lagoa Nova
59072-970 - Natal-RN, Brasil

Resumo

Este artigo discute o problema da proliferação de versões de nós e elos em modelos conceituais hipermídia com composições aninhadas. A possibilidade de um mesmo nó estar contido em diferentes composições aninhadas, assim como poder ser exibido (editado) com representações diferentes, pode gerar uma proliferação desnecessária de versões, se não houver um controle bem definido da propagação de versões de composição. A resolução repetitiva de estruturas virtuais (nós, elos e âncoras) durante o processo de navegação no hiperdocumento, bem como a exigência de tratar todos os atributos de uma entidade conjuntamente como versionáveis ou não, também podem gerar uma proliferação indesejada de versões. O artigo aborda esses assuntos comentando soluções parciais apresentadas em vários modelos hipermídia e apresentando a proposta integrada definida no modelo NCM do sistema HyperProp para o controle da proliferação e propagação de versões.

Abstract

This paper discusses the links and node versions proliferation in hypermedia models with nested composite nodes. The possibility of a node being contained in different nested compositions, as well as being exhibited (edited) with different representations, can generate a proliferation of useless versions, if we do not have a well-defined version propagation control. The repetitive resolution of virtual hypermedia structures (nodes, links and anchors) during hyperdocument navigation, as well as the requirement of defining all attributes of an entity, as a whole, as versionable or not, can also create an undesirable proliferation of versions. The paper discusses these issues, relating partial solutions presented in several hypermedia models and presenting the proposal defined in the Nested Context Model of the HyperProp System for version proliferation and propagation control.

1 - Introdução

A possibilidade de definir referências arbitrárias entre partes quaisquer de um documento representa, para muitos, a característica marcante de um sistema hipermídia. No entanto, é necessário combinar esta flexibilidade com mecanismos que permitam a definição estruturada de documentos. A estruturação deve, entre outras coisas, facilitar o reuso de dados, como por exemplo, permitir que um mesmo componente do documento pertença a diferentes partes da estrutura sem a duplicação de seu conteúdo. Isto vai exigir que a definição de referências entre componentes do documento (a partir de agora denominados nós) seja feita independente do conteúdo de seus dados, permitindo o reuso dos dados sem a herança das relações. Em sistemas hipermídia não estruturados, como atualmente o WWW, essa herança é obrigatória.

A estruturação lógica do documento é realizada pela introdução do conceito de composição. Em alguns modelos, os nós de composição são definidos como um agrupamento lógico dos nós de um documento, e podem ser aninhados (ou seja, uma composição pode conter outra

composição recursivamente) [1, 2], ou não [3]. Nesses modelos, as referências são armazenadas em repositórios independentes, sendo inclusive, em alguns casos, um repositório único [2, 3]. Já em outros modelos, mais gerais, as composições contêm não somente nós (que podem ser outras composições), mas também as referências (a partir de agora denominadas elos), o que oferece como principais benefícios a contextualização dos elos [4]. Em todos os modelos, como os elos ancoram em nós recursivamente contidos em composições e como um mesmo nó pode pertencer a várias composições diferentes, os elos devem não só definir os nós de ancoragem fonte e destino, mas também em que aninhamento de composições estão estes nós.

O conceito de composição é necessário não apenas por prover mecanismos para estruturação de documentos, mas também, como conseqüência, por possibilitar o trabalho cooperativo e mecanismos de controle de versões eficientes [5]. No entanto, a complexidade do controle de versões torna-se ainda maior em modelos com composições, quando deve-se levar em conta todos os problemas da proliferação de versões de nós e elos, e propagação de versões nas composições.

O problema da propagação agrava-se ainda mais quando são permitidas representações diferentes de um mesmo nó (diferentes exibições de um mesmo dado ou diferentes cópias da mesma informação em uso em determinado instante) como versões deste nó. O mecanismo de controle de versões, apresentado em [5, 6] salienta a importância desse fato, no entanto, embora mencione o problema de replicação de elos na criação de versões e defina de forma rigorosa o escopo da replicação, o problema acarretado na propagação de versões é apenas ligeiramente mencionado. O trabalho presente complementa o trabalho anterior [6].

Controle de versões de elos não é considerado no trabalho, por motivos que serão discutidos no texto. No entanto, uma discussão sobre os problemas que acarretaria em modelos com composições também é realizada.

O artigo analisa em profundidade o problema da proliferação de versões em modelos com composições, em especial nos modelos com composições aninhadas. Para tanto, é necessário a definição um pouco mais formal de tais modelos, o que é realizado de forma resumida na Seção 2, mas suficiente para dar base ao trabalho. A Seção 3 apresenta o problema da proliferação de versões desnecessárias, deixando para a Seção 4 as várias soluções propostas no controle de versões do sistema HyperProp. A Seção 5 é reservada a conclusões.

2 - Versões em Documentos Multimídia com Composições Aninhadas

Em modelos com composições, documentos são modelados como um tipo especial de nó,

chamado nó de composição. O conteúdo de dados do hiperdocumento está contido em outra classe de nós, os nós usuais hipertextos, chamados de *nós de conteúdo*, que podem ser especializados em outras classes (texto, áudio, imagem, vídeo, etc.). Um *nó de composição* pode ser definido como uma coleção de elos, nós de conteúdo e nós de composição, recursivamente.

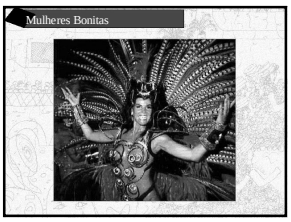
De uma forma geral, um *elo* (uma relação n:m) pode ser definido por um conjunto de pontos terminais de origem, um conjunto de pontos terminais de destino e o tipo da relação e seus atributos. Como um nó pode pertencer a mais de uma composição, os pontos terminais de origem e destino são identificados por pares da forma $<(N_k,...,N_1),A>$, tal que N_1 é um nó, N_{i+1} é um nó de composição e N_i está contido em N_{i+1}, para todo $i \in [1,k)$, com $k > 0$, e A é uma âncora (um conjunto de informações marcadas) de N_1, que é denominado *nó âncora* do ponto terminal. N_k deve ser um componente do nó de composição que contém o elo, ou o próprio nó de composição.

Como modelos com composições aninhadas usualmente permitem que composições diferentes possam conter o mesmo nó, é necessário introduzir o conceito de perspectiva do nó, para a compreensão dos mecanismos de controle de proliferação de versões propostos. Intuitivamente, a perspectiva de um nó identifica através de que seqüência de nós de composição aninhados uma dada instância de um nó N está sendo observada. Formalmente, a *perspectiva* de um nó N é uma seqüência $P = (N_m,....,N_1)$, com $m \geq 1$, tal que $N_1 = N$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para $i \in [1,m)$ e N_m não está contido em qualquer nó. Note que pode haver várias perspectivas diferentes para um mesmo nó N, se este nó estiver contido em mais de uma composição. A *perspectiva corrente* de um nó é aquela percorrida pela última navegação ao nó na sessão de trabalho do usuário. Dada a perspectiva $P = (N_m,....,N_1)$, o nó N_1 é chamado *nó base da perspectiva*.

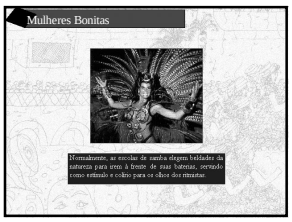
A definição do conteúdo de um componente do documento bem como a definição de sua granularidade, isto é, que conjunto de suas unidades de informação pode ser marcado para definição de âncoras, estão contidas no que chamaremos de *objetos de dados*, cuja definição detalhada pode ser encontrada em [7]. Usualmente, a criação de tais objetos é realizada por editores de uma mídia específica.

Adicionalmente, é necessário definir como cada componente deverá se comportar quando exibido. Para cada apresentação de um conjunto de unidades de informação de um objeto de dados, deve ser possível especificar com que ferramenta o objeto deve ser apresentado. Essas definições devem, preferencialmente, ser especificadas independente do objeto de dados associado, de forma a permitir seu reuso com diferentes apresentações. No NCM (Nested

Context Model), o modelo conceitual do sistema HyperProp, tais objetos são chamados de *descritores*. Na Figura 1a e 1b podemos ver o mesmo objeto de dados contendo a imagem da mulher sendo exibido de duas formas diferentes, devido a dois descritores distintos. Note que o uso de descritores permite ao mesmo objeto de dados ser exibido em mídias diversas, por exemplo, um segmento de texto pode ser exibido como um texto usando um descritor D_1, ou pode ser sintetizado como um áudio, usando um outro descritor D_2.



a) Parte do documento sobre o carnaval



b) Apresentação da imagem da Figura 1.a com outro descritor

Figura 1 - Duas composições, parte de um documento sobre o carnaval brasileiro

A agregação de um objeto de dados a um descritor, para sua apresentação, gera o que chamaremos de *objeto de representação*, e será considerado uma *versão de representação* do objeto de dados.

Objetos de dados podem ser criados como objetos totalmente novos, como versões locais de outros objetos já criados e persistentes, denominados de *objetos de armazenamento*, ou como versões de outros objetos de dados. Os objetos de dados gerados a partir de outros objetos são considerados *versões de dados*. Assim, objetos de armazenamento, de dados e de representação estão intimamente relacionados por operações de versionamento.

Todas as vezes que um objeto de dados for apresentado ele deve ser associado a um descritor, que pode ser passado como atributo do elo usado para navegação até esse nó, ou como atributo do nó de composição que contém o nó a ser apresentado, ou ainda pode ser um descritor default do tipo do nó, caso nenhum descritor seja especificado pelo elo ou pela composição. A Figura 2 exemplifica a associação de nós de dados a descritores. Como será representado em todo texto, nós são desenhados como círculos, elos como linhas cheias e nós de composição como círculos que contêm outros círculos. A associação entre nós e descritores é ilustrada na figura por linhas pontilhadas conectando objetos do plano intermediário com objetos de representação desenhados no plano superior. Em particular, deve ser notada a associação dos descritores D_1, D_2 e D_3 com o mesmo nó *A'*, criando os objetos de representação A_1", A_2", e A_3". Na figura, *A'* tem três representações diferentes, por exemplo, uma para cada forma diferente de navegação até ele. Pode-se navegar para *A'* em profundidade (através do nó de composição *C'*) usando o descritor D_1, ou pode-se navegar por um dos elos

que tocam *A'*, usando o descritor D_2, ou pelo outro elo, usando o descritor D_3.

Note que, na criação de versões de representação, elos são duplicados (observe o elo que partia do nó *A'*). Quando permite-se a um objeto de dados gerar vários objetos de representação, a definição rigorosa dessa replicação torna-se de grande importância [6].

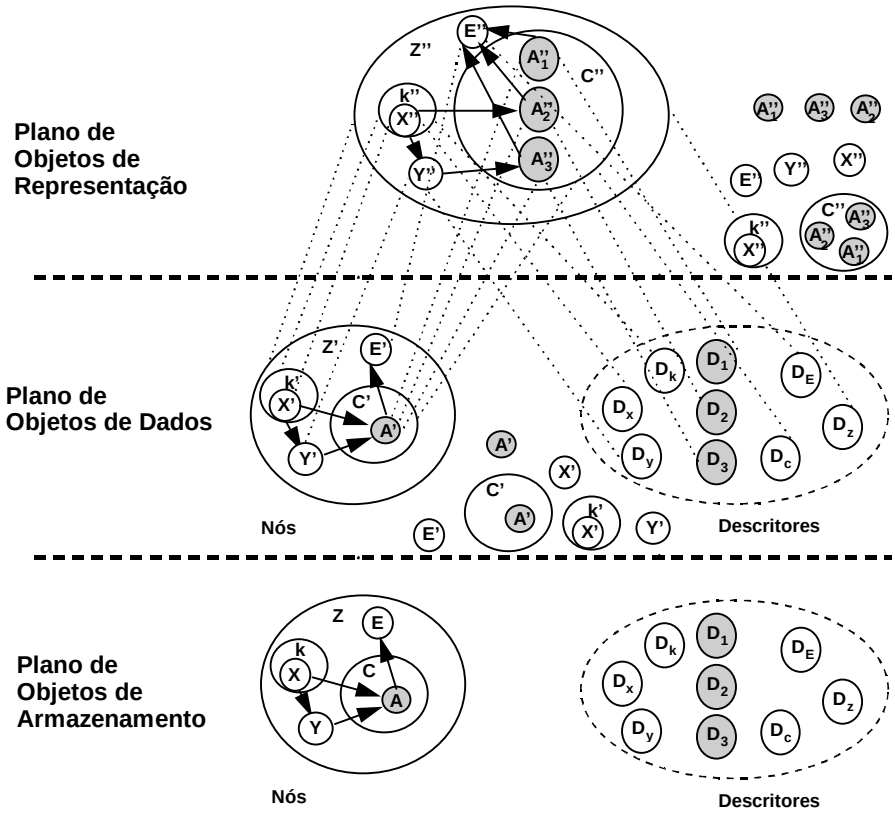


Figura 2 - Planos de Objetos

Modelos de versões [8, 9, 10, 5] usualmente incluem a noção de estado de um nó para facilitar o controle de consistência entre nós, o suporte a trabalho cooperativo e a criação automática de versões. Continuando com o NCM como exemplo, um nó *N*, seja ele de conteúdo ou de composição, pode estar em um dos seguintes estados: *permanente*, *temporário* e *obsoleto*. Um nó é sempre criado no estado temporário e permanece neste estado enquanto está sendo modificado. Quando se torna estável, o nó pode ser promovido ao estado permanente, explicitamente através de uma operação do usuário, ou implicitamente como efeito colateral de certas operações do modelo. Um nó permanente não pode ser diretamente removido ou alterado, mas o usuário pode torná-lo obsoleto, permitindo que outros nós que o referenciem, ou dele derivados, sejam notificados. Nós obsoletos são mantidos apenas enquanto são referenciados por um elo ou são contidos em outro nó.

Um nó *N* do plano (base) de objetos de armazenamento (POA) não pode migrar para o plano (base) de objetos de dados (POD), embora possam ser criadas versões de dados de *N*.

Analogamente, um nó em um POD não pode migrar para um plano de objetos de representação (POR), embora possam ser criadas versões de representação deste nó. Em geral os modelos apresentam primitivas do tipo *Check-in* que permitem a criação de versões de um nó *N* em uma dada perspectiva [9, 10, 5].

Um usuário pode mover um nó de conteúdo ou de composição de um POR para um POD e de um POD para o POA através da primitiva *check-out*, desde que o nó esteja no estado permanente. Em geral todos os modelos requerem que se um nó de composição permanente *C* for movido, então todos os nós contidos em *C* devem também ser movidos para a mesma base. Note que a movimentação de um nó de um plano para outro só precisa ser realizada quando ocorrer alguma modificação no conteúdo original. Por exemplo, se após a criação de uma versão de dados *D* de um nó objeto de armazenamento *A*, *D* não sofrer qualquer modificação, *D* é simplesmente destruída (ou seja, tornada obsoleta e depois destruída) ao ser movida para o POA, pois não há a necessidade de replicação de nós.

Outras operações podem ser definidas para manipulação das versões de dados e de representação. Uma discussão detalhada sobre operações para controle de versões pode ser encontrada em [5, 6], onde uma comparação de diversos modelos também é realizada.

3 - O Problema da Proliferação de Versões

Nessa seção identificamos algumas questões relativas à proliferação desnecessária de versões, para então apresentarmos as soluções propostas na Seção 4.

Criação de versões a partir de qualquer alteração em um objeto - Considerar um objeto sempre como um todo indivisível nos mecanismos de versionamento pode gerar uma grande quantidade de versões desnecessárias. Isto fará com que qualquer mudança em qualquer atributo de um objeto acarrete em nova versão.

Criação de versões de um mesmo nó em diferentes perspectivas ou com um mesmo descritor - A primitiva *check-in* discutida na seção anterior, quando aplicada ao mesmo nó em diferentes perspectivas gera diferentes versões, uma para cada perspectiva. Muitas vezes esta propriedade é desejada, porém pode ser oportuno que alterações em um nó sejam válidas para qualquer perspectiva do nó. A proliferação de versões então não se justifica. Por sua vez, a aplicação seguida da primitiva *check-in* em um mesmo objeto de dados para a geração de um objeto de representação, sempre com o mesmo descritor, pode também gerar um grande número de versões indesejáveis.

Criação automática de versões como efeito colateral de operações - Alguns sistemas evitam o problema cognitivo de versionamento através da criação automática de versões, ou a intervalos

regulares (exemplificados a cada *autosave* de um editor) ou como efeito colateral dos comandos (exemplificados a cada *save* de um editor). Estas duas opções podem gerar grande quantidade de versões que não representam necessariamente um passo consistente na evolução de um trabalho.

Criação de versões a partir de resolução de estruturas virtuais - Freqüentemente a navegação em um sistema hipermídia é baseada em uma consulta. A possibilidade de se ter estruturas virtuais (conteúdos de nós, âncoras e elos), ou seja, estruturas que resultam da avaliação de alguma expressão é uma característica de muitos sistemas hipertexto. Um nó virtual pode ter seu conteúdo e âncoras computados quando de sua seleção durante um processo de navegação. Em particular, uma âncora pode ser computada quando um elo que a referencia for selecionado. De forma similar, um elo pode ter seus pontos terminais computados quando selecionado. A importância destes conceitos nos mecanismos de controle de versões são aprofundados em [5]. Se cada vez que uma consulta for resolvida, por exemplo em uma navegação em profundidade, uma nova versão for criada, a sessão de trabalho ficará rapidamente repleta de versões, muitas vezes desnecessárias.

Propagação automática de versões - Em qualquer sistema com nós de composição, uma questão que se coloca é o que acontece com um nó de composição quando uma nova versão de um dos seus componentes é criada. Um sistema oferece *propagação automática de versões* quando novas versões de um nó de composição que contém um nó *N* são criadas cada vez que uma nova versão de *N* for criada.

Como um nó pode estar contido em diferentes nós de composição, propagação automática de versões poderá causar a criação de um grande número de versões indesejáveis, conforme ilustrado na Figura 3. A base inicial, mostrada na Figura 3(a), possui um nó *D0* que pertence aos nós de composição *E0*, *B0* e *F0*; *E0* por sua vez está contido em *C0* e *B0* em *A0*. A criação da versão *D1* de *D0* gera cinco novos nós de composição, conforme mostrado na Figura 3 (b), se a propagação de versões for aplicada exaustivamente.

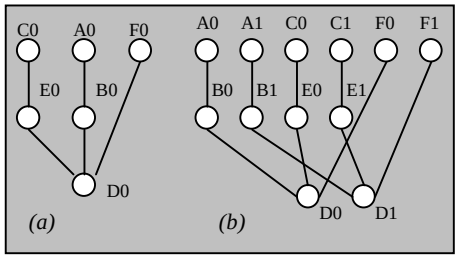


Figura 3 - Proliferação de versões (Note que as arestas não indicam elos, mas a relação de pertinência, ao contrário da convenção adotada em todas as outras figuras)

4 - Soluções Propostas no Sistema HyperProp e Trabalhos Relacionados

Controle de versões tem sido investigado especialmente em projetos de banco de dados e em engenharia de software. Vários modelos têm sido propostos na literatura para descrever a organização e manipulação de documentos em ambientes de trabalho cooperativo, em especial em áreas como CAD e automação de escritório. As referências [8, 9, 10, 11, 12, 13], usadas na comparação com o NCM, endereçam o suporte a versões em sistemas hipermídia.

O NCM fornece mecanismos para evitar a proliferação desnecessária de versões baseados nos conceitos de base privada, estados de nós, atributos versionáveis e não versionáveis de nós, definição de primitivas diferentes para criação de versões e mecanismos de propagação de versões, discutidos no restante desta seção, em referência aos pontos levantados na seção anterior.

4.1 - Criação de versões a partir de qualquer alteração em um objeto

O NCM, como o HyperPro, permite que mudanças em alguns atributos de um nó não criem versões. HyperPro provê essa facilidade associando o aspecto de imutabilidade ao tipo de nó. Para cada tipo deve ser especificado que atributos podem ser modificados em um nó no estado permanente (frozen, no jargão HyperPro). HyperPro também especifica que tipos de elos podem ser ligados a um nó permanente, sem a geração de uma nova versão.

No NCM, de uma forma mais genérica, a especificação de imutabilidade de um tipo é apenas uma definição *default* que pode ser substituída em uma instância particular de um nó. Nós acreditamos que o conteúdo de um nó dê mais informações a respeito de sua imutabulidade do que seu tipo. Por exemplo, qual é a diferença entre um nó tipo áudio e um nó tipo texto quando derivados de um mesmo objeto de dados? Nós também acreditamos que a definição de que tipos de elos podem ser ligados a um nó deve ser responsabilidade do nó de composição que contém o elo. Se alguma entidade vai ser modificada pela alteração ou inclusão de um elo esta entidade é o nó de composição. Assim, por consistência, cabe à composição definir a imutabilidade de seus elos.

No NCM cada atributo (incluindo o conteúdo) de um nó pode ser especificado como versionável ou não. Um atributo não versionável pode ter seu valor modificado sem a necessidade da criação de uma nova versão. Ao contrário, modificações no valor de um atributo versionável devem ser realizadas em uma nova versão do objeto, se este já estiver no estado permanente. Atributos versionáveis e não versionáveis contribuem para a não proliferação desnecessária de versões. Obviamente, algum tipo de mecanismo de notificação será necessário, como extensão ao mecanismo de suporte a versões, especialmente no caso de atualizações concorrentes de atributos não versionáveis.

4.2 - Criação de versões de um mesmo nó em diferentes perspectivas ou com um mesmo descritor

No NCM existem duas primitivas para criação de versões de representação e duas para criação de versões de dados.

As primitivas *Check-in (P, D)* e *Check-in-one (P, D)* permitem a criação de uma nova versão de representação de um objeto de dados *N*, onde *N* é o nó base da perspectiva *P*. A versão é criada pela associação do descritor *D* ao nó *N*. As primitivas diferem no caso de já existir uma versão de representação de *N* no plano de objetos de representação, derivada do mesmo descritor *D*. Neste caso, *Check-in (P, D)* cria uma nova versão, ao passo que a primitiva *Check-in-one (P, D)* não cria nova versão. Em qualquer caso, sendo *R* a versão de representação do objeto de dados *N* pela agregação do descritor *D*, então:

i) se não existirem outras referências ao nó *N*, especificadas nos vários nós que compõem a perspectiva *P*, com outros descritores (referências por elo ou composição), *R* é incluído no plano de objetos de representação substituindo *N* na perspectiva *P*. Todos os elos visíveis em *P* por *N* (isto é, todos os elos definidos nos nós da perspectiva *P* que contêm *N* na lista de nós de um ponto terminal do elo), se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *R*, em substituição a *N*.

ii) se existirem outras referências ao nó *N*, especificadas nos vários nós que compõem a perspectiva *P*, com outros descritores (referências por elo ou composição), *N* não é substituído, mas *R* é incluído na mesma composição que contém *N*, e todos os elos visíveis em *P* por *N*, que têm *N* como âncora de destino e *D* como descritor, devem agora ser apropriadamente direcionados para *R*. Todos os elos visíveis em *P* por *N* que têm *N* como âncora de destino e um descritor diferente de *D* devem permanecer inalterados. Todos os demais elos visíveis por *N* em *P* devem ser duplicados e ter, na cópia, *N* substituído por *R* em seus pontos terminais.

As primitivas *open* e *check-in* permitem a criação de uma nova versão de dados temporária de um nó *N* (objeto de dados ou de armazenamento) no plano de objeto de dados. Estas primitivas diferem no tratamento de nós de composição, como descrito a seguir.

A primitiva *Check-in (P)* permite a criação de uma nova versão de dados de *N*, onde *N* é o nó base da perspectiva *P*. A versão temporária *T* criada é exatamente idêntica a *N*, ou seja, se *N* for um nó de composição, *T* conterá os mesmos nós e elos de *N*. *T* deverá substituir *N* na perspectiva *P*, se *N* for um nó objeto de armazenamento. Neste caso, todos os elos visíveis em *P* por *N*, se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *T*, em substituição a *N*. Caso *N* seja um objeto de dados, *T* é

simplesmente incluído na perspectiva *P* na mesma composição que contém *N*.

Já a primitiva *Open(P)* cria no plano objetos de dados uma versão de dados temporária *T* de *N* (onde *N* é o nó base da perspectiva *P*) e, recursivamente, de cada componente de *N*, se *N* é um nó de composição. *T* conterá, então, as novas versões de dados dos componentes de *N*, e seus elos serão criados de forma a refletir apropriadamente os elos de *N*. Se um componente recursivamente contido em *N* pertencer a mais de uma composição, apenas uma versão de dados temporária será criada para este nó. *T* deverá substituir *N* na perspectiva *P*, se *N* for um nó objeto de armazenamento. Neste caso, todos os elos visíveis em *P* por *N*, se existirem, são alterados de forma a refletir apropriadamente, nos seus pontos terminais, o novo nó *T* e todas as versões de dados criadas, recursivamente contidas em *T*, se *T* é um nó de composição, em substituição aos nós dos quais as versões de dados foram derivadas. Caso *N* seja um objeto de dados, *T* é simplesmente incluído na perspectiva *P* na mesma composição que contém *N*.

Maiores detalhes sobres as primitivas para criação de versões, bem como uma discussão sobre operações de versionamento compostas podem ser encontradas em [6]. No entanto, mesmo com as definições simplificadas podemos tirar várias conclusões.

Algumas conseqüências interessantes decorrem do comportamento diferente das primitivas *open* e *check-in* para a criação de versões de dados. De fato, seja *N* um nó do plano de objetos de armazenamento e *M* um nó contido recursivamente em *N* através de duas perspectivas. Suponha inicialmente que a versão de dados *N'* tenha sido derivada de *N* pela primitiva *check-in*. Observe que, pela definição de *check-in*, temos que *N* e, por conseqüência, *M* são necessariamente permanentes e que *N'* conterá recursivamente *M* exatamente como *N* contém. Ao atualizar *M* através de uma das perspectivas, o usuário criará então uma versão de dados de *M*, que não será visível pela outra perspectiva. Portanto, ao tentar atualizar *M* pela outra perspectiva, o usuário criará uma segunda versão de dados. Suponha agora que *N'* tenha sido criada através da primitiva *open*. Neste caso, *N'* já conterá uma (única) versão de dados *M'* de *M*, temporária, que poderá ser atualizada por quaisquer das duas perspectivas, desde que se faça uso apropriado da primitiva *check-in*. Desta forma, a primitiva *open* elimina a proliferação desnecessária de versões quando se quer que a mudança do conteúdo de um nó seja refletido em todas as perspectivas.

Por sua vez, a primitiva *Check-in-one (P, D)* pode evitar a proliferação desnecessária de versões de representação, evitando a criação de versão se já existe uma representação do nó com o mesmo descritor, ao contrário da primitiva *Check-in (P, D)* que sempre cria nova versão e só deve ser usada se necessário.

Com exceção do NCM, não existe nada semelhante às operações *open* e *check-in-one* em nos

outros sistemas mencionados neste texto. Também em nenhum deles é considerada a possibilidade de diferentes representações (exibições) de um mesmo objeto (de dados) serem tratadas como versões. Em geral é permitida uma única representação de cada objeto de dados, evitando-se assim o problema, mas apresentando um modelo limitado quando comparado à proposição do NCM.

4.3 - Criação automática de versões

No NCM, como em CoVer [9, 12], HAM e HyperPro [10], alterações em um nó no estado temporário não geram novas versões. Operações em atributos versionáveis de nós no estado permanente geram novas versões temporárias dos nós.

Os estados permanente e temporário do NCM são semelhantes às noções de *frozen* e *updatable* usadas em HyperPro e CoVer, e às noções de *archived* e *non-archived* usadas em HAM. Entretanto, no NCM esses estados são usados na tomada de várias decisões para criação automática de versões, o que não acontece naqueles modelos. Além disso, a inclusão do estado obsoleto é bastante útil na gerência do sistema.

No NCM, um objeto de armazenamento só pode estar nos estados permanente ou obsoleto, consistente com o padrão MHEG, onde objetos da camada de armazenamento estão sempre em sua forma final, isto é, não podem ter seus atributos versionáveis alterados. Versões de dados só podem ser derivadas de nós no estado permanente, mesmo sendo derivadas de outros objetos de dados.

As definições de estado de um nó podem parecer excessivamente restritivas à primeira vista. Por exemplo, poder-se-ia considerar interessante permitir que novos nós objetos de dados fossem derivados de um nó temporário. No entanto, esta possibilidade tornaria ainda mais difícil para o sistema garantir a consistência do histórico das versões[1] [5]. Nada impede, porém, que uma aplicação implemente uma modificação da operação de criação de versões tal que a criação de uma nova versão de um nó *N* temporário resulte na criação de uma nova versão do nó permanente do qual *N* é derivado, por exemplo.

Um objeto de dados, a princípio, poderia ser definido como podendo estar apenas nos estados permanente ou obsoleto, tal qual os objetos de armazenamento. Se assim fosse feito, versões de uma forma geral, e não apenas versões de dados, só poderiam ser derivadas de nós permanentes. Embora não explicitada na versão anterior do NCM [5], esta suposição foi assumida. No entanto, ao se permitir várias versões de representação diferentes de um mesmo

---

[1] No NCM o histórico de versões é mantido em uma outra especialização da classe nó de composição denominada contexto de versões.

objeto de dados, manter essa suposição resultaria em uma explosão muito grande do número de versões propagadas, o que levou a versão atual do NCM [6] a permitir a derivação de versões de representação de um objeto de dados no estado temporário. Tome como exemplo a Figura 2. Se A_2" for modificado e for realizada uma operação de *check-out*(A_2"), tem-se como resultado a Figura 4.

Se houvesse a exigência de *C'* estar no estado permanente, seria criada uma nova versão de dados de *C'* e de todos os nós que recursivamente contivessem *C'*, gerando uma propagação desnecessária de versões.

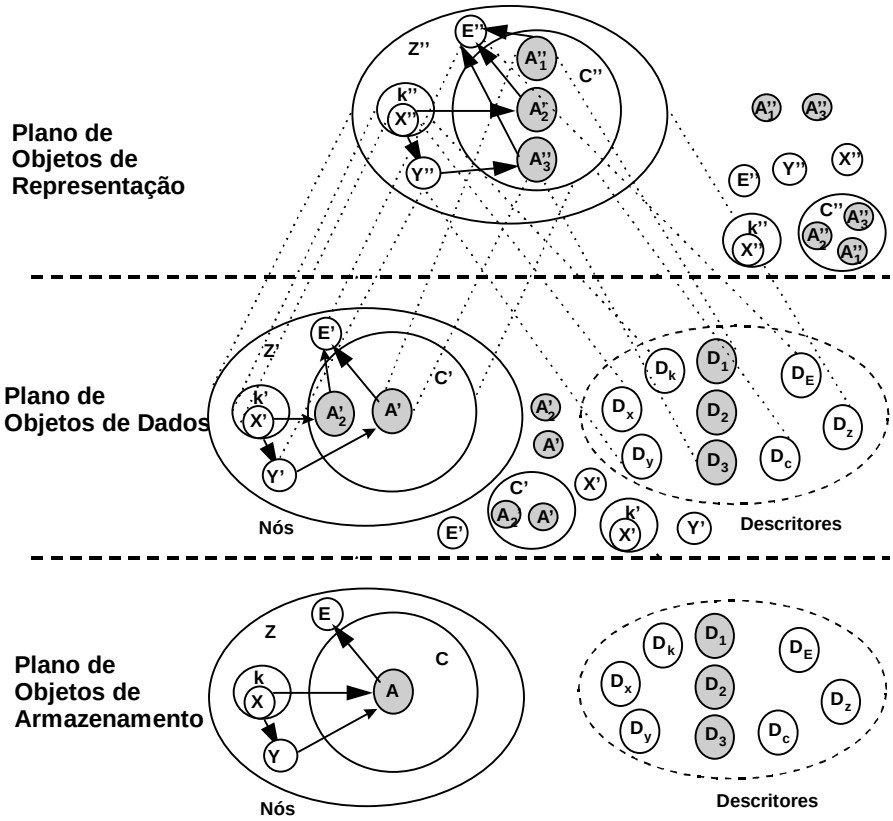


Figura 4 - Check-out A $_2$''

Note que na criação de versões de representação, elos são duplicados, como exemplificado na Figura 4 pelos elos que saem dos *k* (k igual a três) nós A". O problema seria ainda pior se tivéssemos *j* versões de representação do nó C', quando então teríamos *j*k* elos replicados. A situação ainda se agravaria mais caso o elo que parte de A' fosse n:m, isto é, se tivesse *n* pontos terminais de origem. Nesse caso teríamos *n*j*k* replicações possíveis do elo. Esses elos duplicados são na realidade versões, em sistemas com versionamento de elos. Quando permite-se a um objeto de dados gerar vários objetos de representação, a definição rigorosa dessa replicação torna-se de grande importância, pois entre outras coisas ela ditará a proliferação de versões de elos, problema que a princípio parece mais complexo que o controle de proliferação de versões de nós.

HAM e CoVer suportam versões de elos, embora não fique claro como são manipuladas. Não consideramos versões para elos em nosso primeiro protótipo, pois acreditamos que versões de composições serão suficientes. Este é um tópico para verificação em estudos futuros.

4.4 - Criação de versões a partir de resolução de estruturas virtuais

Para evitar o problema, no NCM, uma vez que a consulta é resolvida, ela se torna permanentemente resolvida para aquele plano de objetos (intuitivamente isto significa "para o resto da sessão de trabalho"). Mais ainda, quando nós são movidos de um plano de objeto de dados para o plano de objetos de armazenamento, o usuário pode escolher armazenar a configuração estática (com as consultas resolvidas) presente no plano de objetos de dados, ao invés da configuração dinâmica original, armazenada por *default*.

A facilidade descrita vai possibilitar o armazenamento de configurações estáticas, onde cada nó é uma versão específica e não uma versão corrente (mutável) dentro de um agrupamento de versões. Este é considerado um problema não resolvido no sistema HyperPro, e nem ao menos mencionado nos outros sistemas.

4.5 - Propagação automática de versões

Para minimizar a proliferação de versões, o NCM permite ao usuário decidir se ele quer propagação automática. Além disto, o NCM limita a propagação automática aos nós permanentes de composição que pertencem à perspectiva através da qual a nova versão está sendo construída, em linha com a restrição de que versões de dados não podem ser derivadas de versões que não sejam permanentes. Esta estratégia provê mais uma forma de coordenar atualizações em grupo.

A Figura 5 ilustra estes pontos. Assuma que a base inicial é a mesma usada na Figura 3(a) e que a perspectiva corrente é (*A0,B0,D0*). Se *B0* e *A0* são nós permanentes, então novas versões destes nós serão criadas, como indicado na Figura 5(b). Porém, se *A0* e *B0* não forem permanentes, nenhuma versão destes nós será criada, apenas *B0* será alterada para incluir *D1* (*A0* não será alterada).

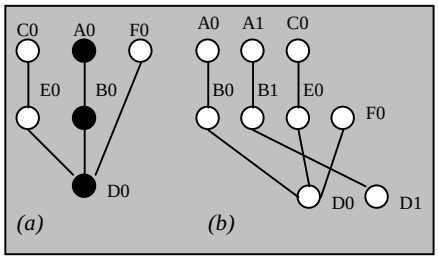


Figura 5 - Propagação guiada pela perspectiva

Se por qualquer motivo o sistema não tem elementos suficientes para decidir se deve ou não

propagar versões, o usuário pode ser chamado a interferir.

Note que uma propagação de versões no plano de objeto de dados pode exigir que sejam redefinidas as relações de derivação (que definem que nó derivou de qual outro nó)[2] dos nós objetos de representação, anteriormente derivados da perspectiva que sofreu a propagação, de forma a referenciar os nós da nova perspectiva criada.

5 - Conclusão

O NCM provê vários mecanismos para evitar a proliferação inútil de versões, baseados nos conceitos de sessões de trabalho, nos estados de um nó, no esquema de propagação de versões e em diferentes primitivas (open, check-in e check-in-one) para criação de versões. Não existe nada parecido com as primitivas *open* e *check-in-one*, e com os conceitos de propagação de versão em qualquer dos modelos já citados. As duas primitivas especiais para criação de versões e a propagação de versões vão ainda adicionar facilidades ao suporte para mudanças coordenadas em conjuntos de nós, suplantando o suporte oferecido pelos outros sistemas.

Embora não explorado neste trabalho, o nó de anotação NCM é outra entidade que limita a proliferação de versões [5]. Anotações NCM são bastante semelhantes ao conceito homônimo do CoVer. Anotações em CoVer são, no entanto, mais gerais. Maiores detalhes podem ser buscados nas referências aos dois sistemas.

O primeiro protótipo do sistema HyperProp foi desenvolvido no Laboratório TeleMídia do Departamento de Informática da PUC-Rio dentro de um projeto para construção de um sistema hipermídia aberto. O objetivo do projeto foi criar uma ferramenta para incorporar o tratamento de documentos hipermídia em qualquer aplicação através de um conjunto de classes, inicialmente implementadas em C^{++}. O protótipo incorpora todas as facilidades para autoria e apresentação de documentos com sincronismo temporal e espacial, para controle de versões e para trabalho cooperativo, como detalhado em [14]. 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. Este é um dos 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 sistemas de teleconferência. Mecanismos para controle de notificação, necessários para trabalho cooperativo, devem também fazer parte da próxima extensão do modelo, atualmente sendo implementado em Java e integrado ao WWW [4].

---

[2] No NCM essas relações são mantidas nos já referidos nós de contexto de versões

Referências

[1] Anderson, K.M. "Integrating Open Hypermedia Systems with the World Wide Web". The Eighth ACM International Hypertext Conference. Southampton, Inglaterra, Abril de 1997. pp.157-166.

[2] Andrews, K.; Kappe, F.; Maurer, H. “Serving Information to the Web with Hyper-G”. *Computer Networks and ISDN Systems*, 27 (6), 1995. pp.919-926.

[3] Grønbœk, K.; Bouvin, N.O.; Sloth, L. "Designing Dexter-based hypermedia services for the World Wide Web”. *The Eighth ACM International Hypertext Conference*. Southampton, Ingl., Abril de 1997.

[4] Rodrigues, R.F.; Soares, L.F.G. “Integração dos Sistemas HyperProp e WWW”. *Anais do IV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia*, Rio de Janeiro. Maio de 1998. Aceito para publicação.

[5] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.R. “Nested Composite Nodes and Version Control in an Open Hypermedia System”. *International Journal on Information Systems; Special issue on Multimedia Information Systems*, vol 20, no. 6. Elsevier Science Ltd. England. Setembro de 1995; pp.501-520.

[6] Soares, L.F.G.; Souza, G.L.; Rodrigues, R.; Muchaluat, D. “Versões de Representação em Modelos com Composições Aninhadas”. *Anais do IV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia*, Rio de Janeiro, RJ. Maio de 1998. Aceito para publicação.

[7] Soares L.F.G.; Casanova, M.A.; Colcher, S. “An Architecture for Hypermedia Systems Using MHEG Standard Objects Interchange”. *Information Services & Use*, vol.13, no.2. IOS Press. Amsterdam, The Netherlands. 1993; pp. 131-139.

[8] Delisle, N.; Schwartz, M. “Context - A Partitioning Concept for Hypertext”. *Proceedings of Computer Supporteed Cooperative Work*. Dezembro de 1985

[9] Haake, A. “Cover: A Contextual Version Server for Hypertext Applications”. *Procceedings of European Conference on Hypertext, ECHT'92*. Milano. Dezembro de 1992.

[10] Osterbye, K. “Structural and Cognitive Problems in Providing Version Control for Hypertext”. *Procceedings of European Conference on Hypertext, ECHT'92*. Milano. Dezembro de 1992.

[11] Haake, A.; Haake, J. “The CoVer: Exploiting Version Support in Cooperative Systems”. *Proceedings of INTERCHI’93 - Human Factors in Computing Systems*. Amsterdam. Abril de 1993.

[12] Haake, A.; Hicks, D. “VerSe: Towards Hypertext Versioning Styles”. *Proceedings of Hypertext 96*. Washington, EUA. Setembro de 1996.

[13] Durand, D.; Haake, A.; Hicks, D.; Vitali, F. (Eds). *Proceedings of the Workshop on Versioning in Hypertext Systems*, held in connection with ECHT’94. Disponível como *Arbeitspapiere der GMD 894*, GMD-IPSI, Darmstadt, Alemanha.

[14] Souza, G.L.; “Sincronismo na Modelagem e Execução de Apresentações de Documentos Multimídia”. Tese de Doutorado do Departamento de Informática da PUC-Rio, Setembro de 1997.