

Delineamento da Arquitetura de um Sistema para Processamento de Hiperdocumentos Multimídia

Luiz F.G. Soares¹, Noemi Rodriguez¹, José L. Rangel¹,
Maria Júlia D. Lima¹, Luiz Tucherman², Marco A. Casanova²

¹Departamento de Informática
Pontifícia Universidade Católica do RJ
R. Marquês de S. Vicente, 225
22.453, Rio de Janeiro, RJ - Brasil

²Centro Científico Rio
IBM Brasil
P.O. Box 4624
20.001, Rio de Janeiro, RJ - Brasil

SUMÁRIO

A arquitetura de um sistema localmente distribuído para processamento de documentos é descrita. Um documento é representado por um conjunto de nós ligados por elos, como nos sistemas usuais de hipertexto. Porém, os nós podem conter dados multimídia e podem ser organizados em contextos, aninhados até uma profundidade arbitrária. O sistema está estruturado em duas camadas: a camada de nós preocupa-se com o armazenamento persistente de nós e elos e a camada de apresentação implementa as facilidades navegacionais e oferece representações alternativas para cada tipo de nó.

1. INTRODUÇÃO

Este trabalho descreve brevemente a arquitetura de um sistema localmente distribuído para processamento de documentos. As aplicações previstas para o sistema são aquelas típicas do tratamento de documentos em um ambiente de escritório, tais como correio eletrônico inteligente e tele-conferência.

Representa-se um documento por um conjunto de nós conectados por elos, como nos sistemas familiares de hipertexto [3,6], justificando assim o uso do termo *hiperdocumento*. Porém, os nós são classificados em nós *terminais*, que contêm dados, e nós de *contexto*, que definem conjuntos de nós terminais e, recursivamente, conjuntos de nós terminais e de contexto. Nós terminais são por sua vez classificados em nós de texto, áudio, imagem, etc... Nós de contexto podem então ser usados para estruturar documentos e para definir visões diferentes do mesmo documento. O nome *sistema para processamento de hiperdocumentos multimídia* foi escolhido para enfatizar estas características.

O sistema está estruturado em duas camadas principais, conforme mostrado na Figura 1. A *camada de nós* oferece armazenamento persistente de nós e elos, enquanto que a *camada de apresentação* implementa as facilidades navegacionais e oferece, para cada tipo de nó, representações alternativas, sintonizadas para classes diferentes de aplicações.

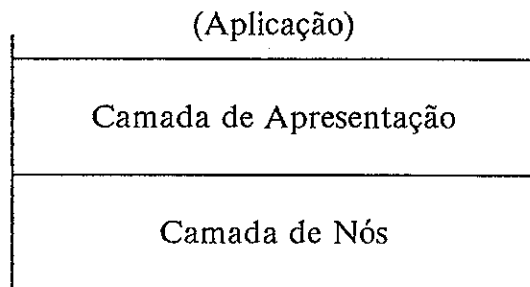


Figura 1. Estrutura de Camadas do Sistema

O conceito de nó de contexto generaliza o conceito homônimo introduzido no sistema Neptune [4], que se baseou por sua vez em algumas idéias do PIE [5]. Porém, contextos em Neptune não podem ser aninhados. Este conceito também generaliza os "webs" do Intermedia [8] e os "fileboxes" do Notecard [7]. De fato, é muito fácil implementar estas estruturas usando os nós de contexto aqui propostos. Exemplos de sistemas para processamento de documentos multimídia encontram-se em [1,2,10].

Este trabalho está assim organizado. A seção 2 informalmente introduz o modelo conceitual para hiperdocumentos. A seção 3 descreve as duas camadas do sistema. Finalmente, a seção 4 contém as conclusões.

2. MODELO CONCEITUAL PARA HIPERDOCUMENTOS

Descreveremos informalmente nesta seção o modelo conceitual que o sistema implementa, isto é, os mecanismos que propomos para estruturar documentos.

O modelo conceitual proposto possui dois conceitos básicos, nó e elo, como usual. O modelo distingue, porém, dois tipos diferentes de nós: terminais e de contexto. Os elos conectam nós, de ambos os tipos, formando grafos de nós correlacionados.

Intuitivamente, um *nó terminal* armazena dados que serão apresentados em algum meio de saída. Usamos o termo 'apresentado' aqui no sentido mais geral de apresentar textos,

figuras ou imagens em uma tela, produzir som em um meio apropriado, etc... Cada nó terminal possui um *tipo*, que pode ser tanto um dos tipos pré-definidos (*texto*, *áudio*, *imagem*, etc.) quanto um dos tipos definidos pelo usuário. O tipo de um nó determina os seus *atributos*, que contem informação definida pelos usuários ou exigida pelos vários subsistemas, como o subsistema de autorização. Em particular, um dos atributos define o *conteúdo* do nó, que é uma descrição dos dados associados ao nó. Um outro atributo, *palavras-chave*, descreve uma lista de palavras-chave que auxiliará a localização de nós com certas características.

✕

Um nó de *contexto* é um tipo especial de nó usado para agrupar um conjunto de nós terminais ou, recursivamente, um conjunto de nós terminais ou de contexto. O conceito de nó de contexto permite organizar, hierarquicamente ou não, conjuntos de nós e oferece um mecanismo para definir visões diferentes do mesmo documento, sintonizadas para aplicações diferentes ou para classes distintas de usuários. Definimos então uma *base de hiperdocumentos* como um conjunto de nós de contexto.

Por exemplo, para organizar hierarquicamente um livro-texto sobre aspectos formais de Ciência da Computação, podemos definir inicialmente um nó de contexto B que contém um conjunto de nós representando os capítulos do livro e um conjunto de elos indicando a organização dos capítulos, que não precisa ser uma seqüência. De forma similar, para o i -ésimo capítulo, podemos definir um 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 por diante. Além disto, como ocorre frequentemente, se uma organização diferente para os capítulos é mais adequada a uma segunda classe de leitores, podemos definir um outro nó de contexto B' contendo os mesmos nós que B (ou seja, os mesmos capítulos), mas um conjunto diferente de elos indicando a nova organização.

Mais precisamente, se um nó N é do tipo contexto, então o seu conteúdo será um par (S,L) , onde S é um conjunto de nós e L é um conjunto de elos (veja abaixo a definição deste conceito) cujos extremos pertencem a S . Porém, impomos uma restrição natural sobre S , definida da seguinte forma. Dizemos que N contém imediatamente um nó M sse M está em S e dizemos que N contém um nó M sse N contém imediatamente M ou existe um nó P tal que N contém imediatamente P e P contém M . Um nó de contexto N está bem definido sse N não contém ele próprio. Deve ficar bem claro que um nó N pode estar contido em mais de um nó de contexto e que toda mudança afetando N será visível através de todos os nós de contexto que contem N .

O conteúdo de um nó (terminal or contexto) pode na verdade ser uma consulta que contrói os dados associados ao nó a partir da base de hiperdocumentos cada vez que o nó é selecionado. Neste caso, o nó será chamado de *virtual*.

Retornando ao nosso exemplo do livro-texto, suponha que exista uma classe de usuários interessados apenas em Lógica e que cada nó representando um capítulo possua palavras-chave indicando os tópicos cobertos. Então, podemos definir um nó virtual de contexto contendo apenas aqueles capítulos cujas palavras-chave são 'Lógica' ou 'Lógica Matemática', por exemplo. Note porém que as consultas que definem o conteúdo dos nós virtuais não estão restritas a seleções baseadas em valores de palavras-chave, como neste exemplo.

De forma semelhante a todos os sistemas de hipertexto, um *elo* conecta dois nós, chamados de *extremidades* do elo. Como o valor do atributo *conteúdo* de um nó possui uma estrutura interna, que poderá ser bastante complexa, os elos também indicam, para cada extremidade, a região onde o elo está "ancorado". Por exemplo, para nós de texto, a região será simplesmente uma cadeia de caracteres dentro do texto e, para nós de imagem bidimensional, será um retângulo determinado por um par de coordenadas.

Mais precisamente, um elo possui uma lista de *atributos*, que sempre incluirá os atributos *origem* e *destino*, chamados de *pontos de ancoragem* do elo. O valor da *origem* será sempre um par (N_0, s_0) tal que N_0 é o *nó de origem* e s_0 é o *deslocamento de origem* do elo. O valor do *destino* é similarmente definido como um par cujo primeiro elemento é o *nó de destino* e cujo segundo elemento é o *deslocamento de destino* do elo.

Quando N_0 for um nó terminal, o deslocamento s_0 tanto poderá ser nulo quanto uma posição dentro do valor do atributo *conteúdo* de N_0 , onde a noção exata de posição depende do tipo de N_0 . Assim, quando um usuário, caminhando pelo elo, selecionar N_0 para apresentação, ele verá o elo "ancorado" na posição determinada por s_0 , se s_0 não for nulo, ou "ancorado" em N_0 como um todo, se s_0 for nulo. Por exemplo, se N_0 é um nó de áudio, o usuário escutará o som gravado da posição s_0 em diante. <

Quando N_0 for um nó de contexto, o deslocamento s_0 tanto poderá ser nulo quanto um par (N_1, s_1) tal que N_1 é um nó imediatamente contido em N_0 e s_1 é um deslocamento válido para N_1 . Como no caso de nós terminais, se s_0 for nulo, o usuário verá o elo "ancorado" em N_0 como um todo. A motivação para o segundo caso reside no fato de que, quando um usuário, caminhando pelo elo, selecionar N_0 para apresentação, ele verá o elo "ancorado" em N_1 com deslocamento s_1 . Sendo esta definição recursiva, se N_1 for um nó de contexto, s_1 poderá ser por sua vez um par (N_2, s_2) . Se o usuário selecionar N_1 , ele verá então o elo "ancorado" em N_2 com deslocamento s_2 , e assim por diante até chegar a um nó terminal.

Por exemplo, considere novamente o exemplo do livro-texto. Uma referência, feita no ponto a da seção i do capítulo m, ao ponto b da seção j do capítulo n, pode ser modelada

por um elo com origem $(m,(i,a))$ e destino $(n,(j,b))$. Suponha agora que o usuário esteja na origem do elo e que ele caminhe para o destino do elo. Ele chegará no capítulo n (um nó de contexto). Se ele selecionar n para apresentação, verá o elo ancorado na seção j (um nó de texto, suponhamos) no ponto b .

Não consideramos elos ou pontos de ancoragem virtuais pois ambos podem ser emulados com a ajuda de nós virtuais. Pela mesma razão, abandonamos a idéia de armazenar informação em elos através de algum atributo.

Para concluir, estamos estudando a inclusão de outras facilidades no modelo. De forma geral, para ser considerada, uma facilidade:

- *ss pode ser impl. eficientemente no sistema;*
- não deve ser trivial de implementar pela aplicação; ou
- deve ser útil para um grande número de aplicações.

x

3. ARQUITETURA DO SISTEMA

Nesta seção descrevemos as duas principais camadas do sistema, após apresentar brevemente alguns conceitos de orientação a objeto.

3.1 Preliminares acerca de Orientação a Objeto

Optamos por descrever nosso sistema segundo o enfoque de orientação a objeto principalmente porque os mecanismos de modularização, abstração e extensão que esse enfoque oferece são particularmente adequados para definir, de maneira uniforme, os diferentes e mesmo imprevistos tipos de dados que o sistema deve suportar.

Os conceitos de orientação a objeto que necessitamos são os seguintes. Dizemos que uma *classe* define um conjunto de *objetos*, um conjunto de *atributos* e um conjunto de *métodos*. Uma *característica* da classe [9] é um atributo ou um método da classe. Formalmente, um atributo associado a uma classe é uma função do conjunto de objetos da classe em algum domínio de valores. Informalmente, um atributo pode ser visto como variáveis locais de cada objeto, ou, de forma equivalente, como campos de um registro. Um método definido numa classe é uma operação (procedimento ou função) sobre os objetos da classe. De acordo com a noção de encapsulamento, somente os métodos da classe podem manipular os objetos dessa classe. Mesmo não sendo obrigatório, a definição de classe tipicamente inclui, para cada atributo, um método para recuperar os valores do atributo e um ou mais métodos para manipular esses valores.

Assumimos também que existem métodos genéricos especiais, *Cria* e *Destrói*, para criar e destruir objetos arbitrários no sistema, ou seja, classes, instâncias de classes, etc...

Uma classe *D* pode também ser definida como uma *subclasse* de outra classe *C*, sendo que nesse caso, *D* herda todas as características de *C*. A descrição de *D* pode, no entanto, redefinir alguma característica herdada de *C*.

Uma característica pode ser declarada como *postergada* numa classe *C*. Isto significa que a característica é indefinida em *C*, mas tem que estar definida em qualquer subclasse de *C*.

Por fim, assumimos que o modelo de orientação a objeto possui uma meta-classe *CLASSE* da qual todas as outras classes são instâncias. Essa classe possui características que capturam a estrutura geral de todas as classes, como o método *Mostra* que retorna a lista dos métodos definidos para uma dada classe.

3.2. A camada de nós

A camada de nós oferece armazenamento persistente de nós e elos. Nesta seção iremos destacar apenas a hierarquia de classes que essa camada suporta, mostrada na Figura 2. Os métodos declarados nas classes dessa hierarquia serão utilizados na comunicação da camada de apresentação com a camada de nós.

Em geral, cada classe na Figura 2, exceto a raiz *TOPO*, captura o conceito introduzido na seção 2 cujo nome coincide com o da classe. Por exemplo, a classe *ELO* captura o conceito de elo e, portanto, os objetos dessa classe serão simplesmente chamados de "elos". Somente permitiremos a adição à hierarquia de novas classes, abaixo das classes *NÓ* e *ELO* ou de suas subclasses.

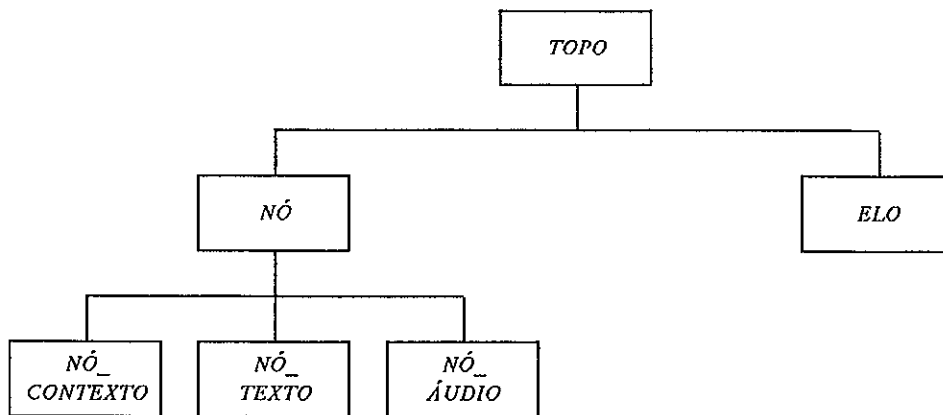


Figura 2. Hierarquia de classes da Camada de Nós

A classe *TOPO* define as características comuns a todas as classes na hierarquia da camada de nós. Ela possui um único atributo, *Id*, cujos valores atuam como identificadores únicos para os objetos existentes num dado instante. A classe *TOPO* também declara um método, *ObtemId*, para recuperar o valor de *Id* (que é automaticamente gerado na criação do objeto). A classe *TOPO* possui *NÓ* e *ELO* como subclasses imediatas.

A classe *NÓ* descreve as características comuns a todos os tipos de nós, além daquelas herdadas de *TOPO*. Ela possui o atributo *Conteúdo*, cujo valor, para cada nó, é o dado associado ao nó, o atributo *PalavrasChave*, que associa a cada nó uma lista de palavras-chaves, e quatro métodos, *ObtemConteúdo*, *MudaConteúdo*, *ObtemPalavrasChave* e *MudaPalavrasChave*, para recuperar e armazenar os valores desses atributos. Além disso, a classe *NÓ* possui outros atributos e métodos, necessários a vários sub-sistemas não descritos neste trabalho.

Cada objeto de *NÓ* deve pertencer necessariamente a alguma das subclasses de *NÓ*, isto é, não existe um tipo "genérico" de nó. Esse aspecto é obtido declarando-se o atributo *Conteúdo* e os métodos *ObtemConteúdo* e *MudaConteúdo* como postergados, garantindo assim que eles serão refinados em todas as subclasses de *NÓ*, mas continuarão indefinidos no nível da classe *NÓ*. Outros atributos, não discutidos neste trabalho, também são declarados como deferidos por razões similares.

A classe *NÓ_CONTEXTO* é definida como subclasse de *NÓ*. O valor do atributo *Conteúdo* para um objeto de *NÓ_CONTEXTO* será um conjunto *N* de identificadores de nós e um conjunto *L* de identificadores de elos entre estes nós, de acordo com a idéia de um nó contexto ser um conjunto de nós e um conjunto de elos entre esses nós.

NÓ_CONTEXTO possui um atributo novo, *NóDefault*, que indica um elemento de *N*, entendido como "o nó default de entrada" do nó contexto.

Além de *NÓ_CONTEXTO*, existe uma subclasse de *NÓ* para cada mídia diferente que o sistema suporta. Por simplicidade, a Figura 2 mostra somente as classes *NÓ_TEXTO* e *NÓ_ÁUDIO*, cujos objetos são nós contendo texto e áudio respectivamente.

Finalmente, a classe *ELO* captura o conceito de elo, como discutido da seção 2. Ela possui dois atributos, *Origem* e *Destino*, que descrevem os dois pontos de ancoragem de cada elo, e quatro métodos, *ObtemOrigem*, *ObtemDestino*, *MudaOrigem* e *MudaDestino*, para recuperar e armazenar os pontos de ancoragem.

3.3. A Camada de Apresentação

O objetivo dessa camada é oferecer diferentes representações para os nós, necessárias às aplicações, além de implementar as facilidades de navegação. Assim, esta camada tem que se preocupar com a manipulação de tipos arbitrários de dados e gerenciar tal manipulação.

Uma *representação* para a classe *NÓ* é uma outra classe *C* contendo um *método de conversão* r , que mapeia os objetos de *NÓ* em objetos de *C*, e um *método de reconversão* r^{-1} que mapeia os objetos de *C* em objetos de *NÓ*. Para cada nó *N*, chamamos o objeto $r(N)$ de *C-representação* de *N*. A noção de representação estende-se também às subclasses de *NODE*.

Tipicamente, uma representação *C* terá métodos apropriados aos tipos de aplicações que a utilizam, e o método de conversão fará a tradução dos valores do atributo *Conteúdo* e, talvez, de outros atributos do nó em valores dos atributos de sua representação. Para criar uma *C-representação* de um nó *N*, a camada de apresentação inicialmente solicitará à camada de nós que localize *N* e depois chamará o método de conversão para criar um objeto *c* de *C*.

Por exemplo, podemos definir a classe *ED* cujos métodos incluem os comandos de um certo editor ED e duas operações que mapeiam nós de texto em objetos de *ED* e vice-versa. Os atributos da classe *ED* incluem *ReprInterna*, cujos valores corresponderão ao formato interno do editor ED para texto. A classe *ED* será então uma representação para a classe *NÓ_TEXTO*. Se uma aplicação quiser manipular um nó de texto *N* através do editor ED, deverá solicitar à camada de apresentação que crie uma *ED-representação* *E* de *N* e, a partir daí, passará a utilizar *E*.

Em qualquer instante, uma aplicação pode ter acesso a várias representações e pode solicitar a criação de representações novas ou a eliminação de representações antigas, como resultado da navegação pelos nós e elos. Para capturar esses aspectos, introduzimos duas classes, *ESTADO* e *HISTÓRICO*. Os objetos da classe *ESTADO* contem, basicamente, listas de representações e são chamados de *estados*. Os objetos da classe *HISTÓRICO*, representam seqüências de estados e são chamados de *históricos*. A camada de apresentação associará então um histórico *h* a cada aplicação; o último elemento de *h* é o *estado corrente* da aplicação. Assim, os serviços de navegação da camada de apresentação serão implementados sobre os métodos definidos nessas duas classes.

Finalmente, identificamos quatro maneiras de navegação:

- 1) travessia de um elo (*navegação por elos*);
- 2) seleção do conteúdo de um nó contexto (*navegação pela hierarquia de contextos*);
- 3) solicitação de estados anteriores (*navegação pelo histórico*); e
- 4) seleção de um nó através da especificação de uma consulta (*navegação lógica*).

Como exemplo do segundo tipo de navegação, considere novamente nosso exemplo do livro-texto. Se o usuário inspecionar o conteúdo do nó de contexto mais abrangente, representando o livro como um todo, ele verá um conjunto de nós de contexto conectados por elos, onde cada um desses nós corresponderá a um capítulo do livro. Em seguida, ele poderá inspecionar o conteúdo de um dos nós de contexto, correspondendo a um capítulo, recebendo novamente como resposta um conjunto de nós e elos, e assim por diante.

É possível definir outras formas de navegação, combinando as anteriores ou criando especializações de cada uma. Por exemplo, podemos imaginar dois casos especiais de navegação por consulta: seleção de um nó especificando seu nome ou seleção de um nó apontando para uma referência a ele mostrada na tela (correspondendo à consulta trivial, onde é indicada a representação desejada).

4. CONCLUSÕES

Delineamos neste trabalho o modelo conceitual e a arquitetura de um sistema para processamento de hiperdocumentos multimídia. Planejamos trabalhar em um futuro imediato na formalização destes pontos e no detalhamento dos vários subsistemas, como controle de versões e autorização. A implementação em sí prosseguirá gradualmente de um sistema mono-usuário para o sistema distribuído multi-usuário completo. A especificação das aplicações dar-se-á em paralelo.

REFERÊNCIAS

- [1] Ahuja, S.R., Ensor, J.R. e Horn, D.N., "The Rapport Multimedia Conferencing System", *ACM SIGOIS Bulletin* Vol.9, N.2/3 (Abril/Julho 1988), 1-8.
- [2] Christodoulakis, S. e Graham, S., "Browsing within Time-Driven Multimedia Documents", *ACM SIGOIS Bulletin* Vol.9, N.2/3 (Abril/Julho 1988), 219-227.
- [3] Conklin, J., "Hypertext: An Introduction and Survey", *IEEE Computer*, Vol.20, N.9 (Set. 1987), 17-41.
- [4] Delisle, N. e Schwartz, M., "Neptune: A Hypertext System for CAD Applications", *Proc. ACM SIGMOD '86*, Washigton, D.C. (Maio 1986), 132-142.
- [5] Goldstein, I. e Bobrow, D., "A Layered Approach to Software Design", in *Interactive Programming Environments*, D. Barstow, H Shrobe e E. Sandewall (Eds.), McGraw-Hill, New York (1987), 387-413.
- [6] Halasz, F.G., "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems", *Communications of the ACM*, Vol.31, No.7 (Julho 1988), 836-852.
- [7] Halasz, F.G., Moran, T.P. e Trigg, R.H., "Notecards in a Nutshell", *Proc. of the 1987 ACM Conf. of Human Factors in Computing*, Toronto, Ontario (Abril 1987), 45-52.
- [9] Meyer, B., "Eiffel: A language and environment for software engineering", *The Journal of Systems and Software*, Vol.8, N.3 (Junho 1988), 199-246.
- [8] Meyrowitz, N., "Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework", *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications*, Portland, Oregon (Set. 1986), 186-201.
- [10] Poggio, A. et alii "CCWS: A Computer-Based Multimedia Information System", *IEEE Computer*, Vol.18, N.10 (Out. 1985), 92-103.