Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

HIP/WINDOWS: UM AMBIENTE DE AUTORIA DE HIPERBASES MULTIMÍDIA

MARIA das GRAÇAS VOLPE NUNES
FABIANO MAGRIN da COSTA VIEIRA
RICARDO RASEGAWA

Nº 38

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
1996

SYSNO 906225
DATA / /
ICMC - SBAB

HIP/Windows:
um ambiente de autoria de hiperbases multimídia[1]

Maria das Graças Volpe Nunes[2]
Fabiano Magrin da Costa Vieira
Ricardo Rasegawa
{mdgvnune, fabiano, rh}@icmsc.sc.usp.br

Departamento de Ciências da Computação e Estatística
Instituto de Ciências Matemáticas de São Carlos
Universidade de São Paulo

Resumo

Este trabalho discute o projeto e a implementação do ambiente de autoria de hiperbases multimídia (HIP) para Windows. Constituem hiperbases multimídia os hiperdocumentos construídos segundo o Modelo dos Contextos Aninhados (MCA) [9], contendo objetos multimídia (texto, som, imagem, etc.).

Palavras-Chaves: Sistemas Hipermídia, Modelo dos Contextos Aninhados, Multimídia, Sistemas de Autoria.

1. Introdução

Um dos maiores problemas enfrentados com a proliferação de sistemas hipermídia é o fato de muitos deles serem desenvolvidos como aplicações únicas e auto-contidas, impossibilitando a reutilização de seus objetos (unidades de informação) em outras aplicações. No entanto, não é desprezível o investimento feito na construção de tais sistemas, de modo que torna-se imperativo disponibilizar essas informações, de modo que incompatibilidade de plataformas ou estruturas de dados usadas nas diversas aplicações não seja um impecilho para isso.

HyperProp [8] é uma proposta de sistema hipermídia que, assim como outras exceções, como Neptune [4], MultiCard [6], HyperBase [7] e Hyperform [11], possibilitam a criação de sistemas que atendem o requisito acima. HyperProp propõe uma arquitetura aberta, distribuída ou não, cujo modelo de interface separa os componentes de dados e de exibição dos objetos através do uso de padrões de representação, permitindo a construção de interfaces independentes da plataforma final de exibição. Além disso, baseia-se num modelo conceitual de dados hipermídia, o Modelo de Contextos Aninhados (MCA), que será descrito na próxima seção. A combinação desses componentes permite a criação tanto de sistemas hipermídia monolíticos,

[1] Este trabalho tem o apoio financeiro do Protem-CC-II, CNPq, #680063/94-3
[2] Pesquisador parcialmente financiado pelo CNPq, #301365/91.1

como a introdução de características hipermídia em aplicações quaisquer. Adicionalmente, permitem que os mecanismos de armazenamento possam ser adaptados aos requisitos de eficiência e tamanho impostos por uma aplicação particular.

A construção de hiperdocumentos com estrutura baseada no MCA pode ocorrer com a ajuda de um sistema de autoria de hiperbases - HIP. A idéia é que a partir da hiperbase, diferentes aplicativos possam ser desenvolvidos. Considerando-se, ainda, que os objetos dessa hiperbase possam estar representados em um padrão internacional para objetos multimídia - MHEG [MHE93] -, essas hiperbases, ou parte delas, poderiam ser reusadas por outros sistemas e mesmo em outras plataformas. Torna-se atraente, assim, possuir máquinas HIP em diferentes plataformas. Tais sistemas devem prover facilidades para que um autor de hiperdocumentos defina os objetos multimídia que o compõem, estruture-os segundo o MCA, num ambiente que inclua recursos criação, edição, visualização e navegação das variadas mídias (texto, som, gráfico, imagem).

Um primeiro protótipo de máquina HIP foi construído em máquinas SUN/ UNIX, C++, e provê facilidades para construção de hiperbases que contenham apenas objetos do tipo texto [1]. A extensão dessa versão, com tratamento de outras mídias, já está em fase de implementação [10]. Algumas aplicações foram implementadas com o uso desse protótipo [2].

Com vistas a diversificar as plataformas de desenvolvimento, este trabalho trata da implementação de uma versão da máquina HIP para Windows, e estende a anterior nos seguintes aspectos: (a) acrescenta a possibilidade de tratar, de maneira ainda limitada, objetos com informações de áudio, vídeo e gráficas, desde que armazenados em formatos aceitos pelo ambiente Windows; (b) disponibiliza acesso a um editor de texto com recursos específicos para marcação de elementos do MCA, como âncoras, por exemplo; (c) tem, como recurso de interface ao autor, um *browser*, onde a estrutura do hiperdocumento pode ser visualizada, em níveis determinados pelo usuário, (d) possibilita a manipulação gráfica direta dos elementos do modelo, representados por ícones, através do uso do mouse, além de oferecer funcionalidade via menus, barras de ferramenta e janelas *pop-up*.

Permanecem ainda como limitações, em relação aos objetivos finais: 1) a integração de ferramentas de edição das mídias de som e vídeo (objetos nessas mídias deve ser construídos e/ou editado fora do ambiente); apenas sua visualização e a navegação sobre eles são possíveis; 2) a representação dos objetos ainda não segue o padrão MHEG, fazendo com que sua visualização seja dependente da plataforma final de exibição; 3) o sincronismo de várias mídias ainda não é tratado -- apenas a simultaneidade pode ocorrer entre objetos de diferentes mídias (de fato, tal tratamento é previsto nos objetos MHEG); 4) em relação a elementos do MCA, os nós de anotação e o contexto de versões ainda não estão sendo considerados; 5) apenas bases privadas foram implementadas, faltanto, portanto, considerar o tratamento de hiperbases públicas.

Numa segunda versão do protótipo, além de se avançar na implementação dos objetivos iniciais, pretende-se atacar principalmente os pontos 4 e 5 acima.

É importante salientar que, além de servir de ferramenta a toda a comunidade que usa o MCA como modelo conceitual de hiperdocumentos multimídia, esta implementação visa ser o primeiro módulo de um Sistema de Autoria de Sistemas Hipermídia aplicados ao Ensino (SASHE), que está em fase de desenvolvimento pela equipe ProTeM/HyperProp do ICMSC-USP (fig. 1). Os dois outros módulos do SASHE, que de fato constituem dois outros pontos de entrada do ambiente, visam atender a outros tipos de usuário: (1) o autor de um curso ou sessão (um professor, por exemplo), não necessariamente o mesmo autor da hiperbase, e que encontrará recursos de composição de "versões" da hiperbase existente que atendam a seus objetivos de ensino; (2) o estudante, que terá acesso aos documentos criados pelo módulo anterior, e que deverão obeceder requisitos próprios de sistemas dedicados ao ensino/aprendizagem, possibilitando, por exemplo, a criação de um modelo do estudante, a partir da navegação sobre o hiperdocumento.

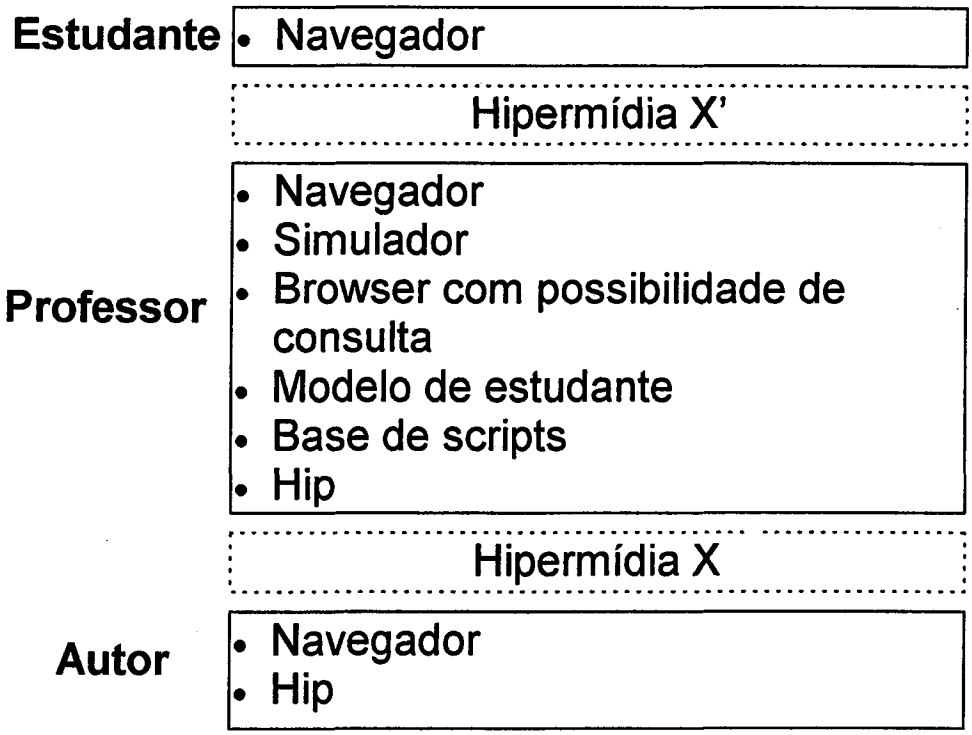


Figura 1: O ambiente SASHE

A seção 2 descreve o Modelo dos Contextos Aninhados. A seção 3 descreve a interface do ambiente de autoria de hiperbases. As seções 4 e 5 apresentam a biblioteca de classes utilizadas pelo sistema e o gerenciador da hiperbase, descrevendo alguns pontos de implementação. Finalmente a seção 6 conclui o trabalho e aponta para trabalhos subsequentes.

2. O Modelo dos Contextos Aninhados

Apresentamos, de maneira informal, na seção 2.1. as definições das entidades básicas do MCA; na seção 2.2 apresentamos a extensão do modelo para o tratamento de controle de versões; e na seção 2.3 discutimos os modelos de apresentação e navegação do MCA.

2.1. Definições do Modelo Básico

Documentos, no modelo MCA, são definidos com base nos conceitos usuais de nós e elos [3]. *Nós* são fragmentos, de tamanho arbitrário, de informação e *elos* interconectam os nós que mantém alguma relação entre si. Há duas classes de nós: os nós terminais e os nós de composição (fig. 2).

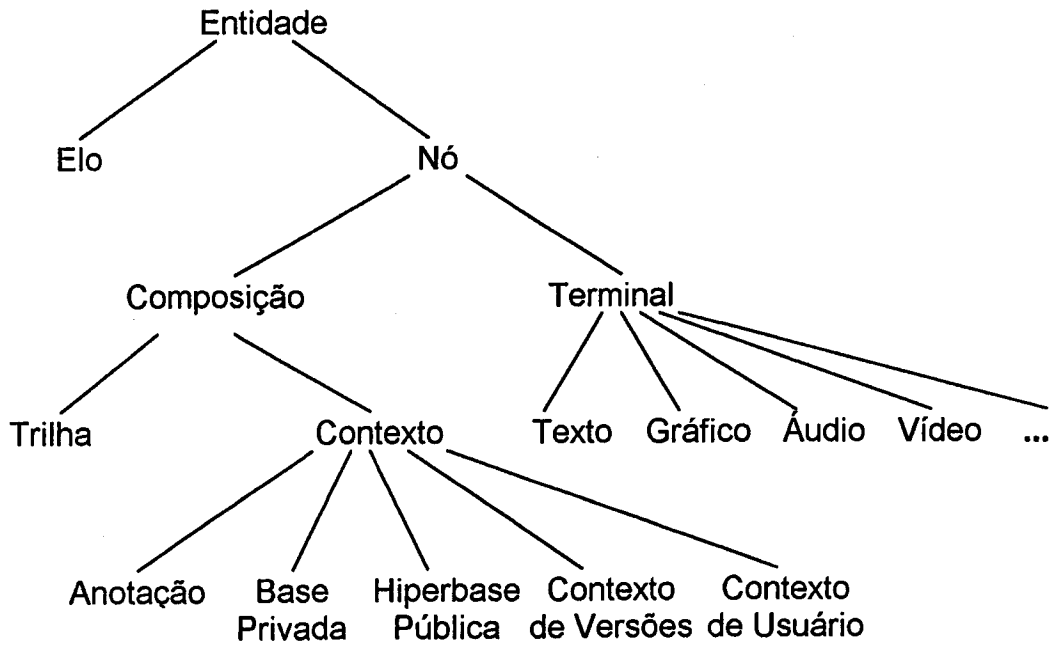


Figura 2: Hierarquia de classes do MCA [9]

Um *nó terminal* caracteriza-se por conter dados cuja estrutura interna é dependente da aplicação. Essa classe contém, assim, objetos com informações textuais, gráficas, de áudio, vídeo, etc.

Um *nó de composição* é um agrupamento de outros nós que pode incluir outros nós de composição. Esse agrupamento não forma necessariamente um conjunto, já que um nó pode aparecer mais de uma vez na composição. Distingue-se duas classes de nós de composição: os nós de contexto e os nós do tipo trilha.

Um *nó de contexto* agrupa um conjunto de elos, trilhas, nós terminais e de contexto, recursivamente. Os nós de contexto se especializam nas classes de nós de anotação, contexto de versões, bases privadas, hiperbase pública e contexto de usuário, que serão discutidas a seguir.

Um *nó de contexto de usuário* agrupa um conjunto de elos, nós terminais e de contexto de usuário, recursivamente. Este tipo de nó permite o aninhamento em qualquer profundidade, permitindo organizar hierarquicamente, ou não, conjuntos de nós. Considere, como exemplo, a organização de um livro texto. Um nó de contexto de usuário seria definido como um conjunto de nós representando os capítulos do livro, e um conjunto de elos indicando a organização dos capítulos, não necessariamente linear. Cada capítulo, por sua vez, daria origem a outro nó de contexto de usuário, contendo um conjunto de nós representando as seções e um conjunto de elos indicando sua organização, e assim por diante.

A conexão de dois nós por um elo deve ser melhor detalhada. Um elo deve indicar a região onde ele *toca* o nó. Dependendo do tipo do nó, essa região pode variar em complexidade. Por exemplo, para nós texto, a região pode ser uma simples cadeia de caracteres; para nós com imagens bidimensionais, essa região pode ser determinada por um par de coordenadas; para nós vídeo, pode ser o número de um frame, e para um nó de contexto, pode ser uma região detnro de um de seus nós componentes. De qualquer forma, uma região é de fato definida com a ajuda da noção de *âncora*.

Âncoras definem regiões em um nó, e têm o papel de servir como interfaces para outros nós, uma vez que através delas outros nós se referem a pontos do conteúdo do nó. Os elos são, assim, definidos com base nas âncoras e qualquer mudança no conteúdo de um nó não se refletirá num elo, se as âncoras forem atualizadas adequadamente. Cada âncora é definida por um par (*id, valor*), onde *id* é único no nó e é chamado de *base da âncora*, e *valor* depende do tipo do nó, conforme discutido acima. Um valor especial de âncora, e que vale para todo tipo de nó, é denotado por λ e indica que a ancoragem é feita no nó como um todo, ou seja, no limite do nó. No caso de um nó de contexto, C, o valor de uma âncora pode ser λ; um subconjunto dos nós contidos em C; ou um caminho de nós $(N_k,...,N_1)$ e uma âncora dentro do nó N_1. No último caso, o caminho é uma sequência de identificadores de nós que representa o aninhamento de contextos, ou seja, N_{i+1} é um nó de contexto e N_i está contido em N_{i+1}, para $i \in [1, k)$; e N_k está contido em C.

Diz-se, então, que um elo é definido pelos *conjuntos de extremidades de origem e de extremidades de destino*. Os valores desses dois atributos são conjuntos cujos elementos, chamados extremidades do elo, são pares da forma $<(N_k,...,N_1),A>$, onde N_1 é um nó, N_{i+1} é um nó de composição e N_i está contido em N_{i+1}, para todo $i \in [1, k)$; $k > 0$, e A é uma âncora de N_1. O nó N_k é chamado de nó base do elo. Embora sempre unidimensional, um elo pode ser percorrido, durante a navegação, em ambas as direções. A definição mostra a possibilidade de se ter elos um-para-um e um-para-muitos.

O modelo exige, ainda, que todo elo contido num nó de composição, C, possua as bases de suas extremidades (N_k acima) contidas em C.

Assim, um nó de contexto de usuário C é um nó de contexto que agrupa:

1. Um conjunto S de nós terminais ou de contexto de usuário.
2. Um conjunto de elos L, tal que cada elo tem os nós bases contidos em C.

Trilhas são especializações de nós de composição que contêm uma lista ordenada de nós, incluindo, recursivamente, nós trilhas. A idéia é representar caminhos que devem ser, ou que foram, percorridos pelo usuário. Dessa forma, um nó poder estar na lista em mais de uma posição.

Uma *hiperbase* é qualquer conjunto H de nós, tais que, para qualquer nó N $\in$ H, se N é um nó de composição, então todos os nós contidos em N também pertencem à hiperbase H. Assim, uma hiperbase define um hiperdocumento com estrutura aninhada.

2.2 Controle de Versões

O modelo MCA também acomoda controle de versões, isto é, um conjunto comum de nós pode ter conjuntos distintos de elos, formando versões distintas de um mesmo conteúdo. Por exemplo, um mesmo conjunto de capítulos de um livro poderia dar origem a diferentes organizações, dependendo do leitor alvo que se tem em mente. O que diferencia essas versões são os conjuntos de elos existentes entre esses capítulos. O modelo MCA faz esse tratamento através da criação de diferentes nós de contexto de usuário, que oferecem visões distintas do mesmo conteúdo. No entanto, para suprir requisitos de gerenciamento de configurações, manutenção do histórico do documento, tratamento de grupos de versões, versões de elos, suporte a documentos alternativos, representações distintas e uso concorrente da mesma informação, quatro novas classes de contexto complementam o MCA: anotação, hiperbase pública, base privada e contexto de versões.

Apenas nós terminais e de contexto de usuário são passíveis de versionamento no MCA. Cada atributo, incluindo o conteúdo, desses tipos de nós pode ser especificado como versionável ou não. Assim, um atributo não versionável pode ter seu valor modificado sem que uma nova versão seja criada. Modificações em atributos versionáveis, por outro lado, só podem ser realizadas em uma nova versão do objeto. Faz-se necessário, então, um mecanismo de notificação que estende o mecanismo de controle de versões.

O usuário pode, ainda, especificar se a adição de novos atributos é permitida sem a criação de novas versões do objeto. Essa possibilidade faz com que eventuais ferramentas adicionadas ao sistema possam introduzir informações específicas nos nós, sem que novas versões sejam necessárias.

2.2.1. Contexto de Versões

Uma classe especial de nós de contexto, chamados de contextos de versões, é introduzida para tratar a manutenção da história de um documento.

Os nós de um contexto de versões V são chamados de versões correlatas e não precisam pertencer à mesma classe (isso possibilita representações

distintintas de uma mesma informação). Um elo da forma (<v_1, i_1>, <v_2, i_2>) em V indica que v_2 é derivada de v_1. As âncoras de um elo em V servem apenas para indicar com mais precisão quais partes de v_1 geraram quais partes de v_2.

Um nó de contexto de versão poder conter nós de contexto de usuário, uma vez que estes também estão sujeitos a versionamento. Assim, é possível versionar a estrutura de um documento, fazendo com que o estado anterior do hiperdocumento possa ser recuperado.

Uma aplicação pode definir um nó de V como sendo sua versão corrente, de acordo com um critério específico. Isso pode ser feito através de uma âncora específica para a versão corrente, enquanto outras especificam outras versões de acordo com outros critérios de seleção.

Agora, o ponto terminal, N_1, de um elo <(N_k,...,N_2, N_1), α> contido num contexto de usuário C pode ser, além de um nó terminal ou um nó de contexto de usuário, um nó de contexto de versões. Neste caso, dizemos que N_2 contém o nó especificado pela âncora α, e não mais N_1.

Contextos de versões podem ser usados para (a) manter a história de um documento D, já que trata-se de um nó de contexto e, portanto, admite os mecanismos genéricos de navegação sobre ele; (b) a atualização automática de referências; e (c) a obtenção de documentos alternativos. Por exemplo, suponha que *d* tenha um componente *c* em cujas versões uma aplicação está interessada. Seja *C* o nó contexto de versões que contém as versões de *c*. A aplicação deverá referenciar *C*, e não uma versão dentro de *C*. Todos os elos em *D* tocando *C* apontarão para a mesma âncora de *C*, que por sua vez sempre apontará para o nó que a aplicação considera como versão corrente de *c*. Navegando em *C*, a aplicação poderá recuperar versões anteriores. A aplicação poderá obter alternativas para o sub-documento *c*, por exemplo, através da consulta (retornando um ou mais nós).

Para otimizar o espaço de armazenamento, ao invés de se armazenar cada elemento de um grupo de versões, apenas as diferenças entre estes elementos devem ser armazenadas. Este mecanismo é conhecido como *técnica delta.* Sua desvantagem é que a recuperação de uma versão específica requer computação partindo do primeiro elemento, incluindo as mudanças. Além disso, algoritmos delta são específicos para cada mídia. Técnicas dessa natureza não fazem parte do modelo do contexto de versões, mas sim são tarefas do sistema de armazenamento, através de seus algoritmos.

2.2.2. Consistência de nós

Um nó terminal ou um nó de contexto de usuário N pode estar em um dos seguintes estados: *permanente, temporário e obsoleto.* Um nó é sempre criado como temporário e assim permanece enquanto estiver sendo modificado. Quando se torna estável, pode ser promovido ao estado permanente, através de uma operação explícita do usuário ou como efeito colateral de certas operações do modelo, por exemplo, via uma operação de criação de versão

aplicada a ele. Um nó permanente não pode ser diretamente removido, mas pode se tornar obsoleto, permitindo que outros nós que o referenciam, ou que são derivados dele, sejam notificados. O conceito de estado do nó só tem sentido para o nós versionáveis, e portanto, os nós terminais e de contexto de usuário.

Mais precisamente, um nó terminal ou de contexto de usuário no estado permanente, ou simplesmente, um nó permanente, possui as seguintes características:

- seus atributos versionáveis não podem ser modificados (ou seja, os atributos explicitamente definidos não podem ser modificados);
- se for um nó de contexto de usuário, só pode conter nós permanentes ou obsoletos;
- pode ser usado para derivar outros nós, isto é, versões;
- não pode ser diretamente removido;
- pode se tornar obsoleto, mas não pode voltar a ser temporário.

Um nó terminal ou de contexto de usuário no estado temporário, chamado de nó temporário, possui as seguintes características:

- todos os seus atributos podem ser modificados;
- se for um nó de contexto de usuário, pode conter nós em qualquer estado;
- não pode ser usado para derivar versões;
- pode ser diretamente removido;
- pode se tornar permanente, mas não pode se tornar obsoleto.

Um nó terminal ou de contexto de usuário no estado obsoleto, chamado de nó obsoleto, possui as seguintes características:

- nenhum de seus atributos pode ser modificado (ou seja, os atributos explicitamente definidos não podem ser modificados);
- se for um nó de contexto de usuário, só pode conter nós permanentes ou obsoletos;
- não pode ser usado para derivar versões;
- é automaticamente removido pelo sistema, através de processo de coleta de lixo, quando não estiver sendo referenciado por outros nós;
- não pode mais mudar de estado

Sempre que o usuário criar um elo representando derivação de versões, o estado do objeto-fonte é automaticamente alterado para permanente, a fim de preservar consistência.

2.2.3. Hiperbase Pública e Base Privada

Os conceitos de hiperbase pública e base privada são úteis para aplicações em processos de autoria cooperativa, que consiste em criar ou modificar a hiperbase, ou um subconjunto dela, por um grupo de usuários.

Neste tipo de ambiente, o sistema deve permitir que usuários compartilhem informações, oferecer alguma forma de criar informação privada, e permitir a fragmentação do espaço global de trabalho em frações menores para reduzir o espaço de navegação.

Considere o conjunto de todos os nós de contexto de usuário e nós terminais particionado em vários subconjuntos. Um e apenas um deles forma a hiperbase pública, denotada por Hb, que corresponderá à informação pública e estável. Os demais formam as bases privadas, usadas para modelagem da interação do usuário com o hiperdocumento, de acordo com o paradigma de sessão de trabalho proposto pelo Modelo Dexter [5]. Uma base privada pode conter outras bases privadas, o que permite organizar a sessão de trabalho em várias subsessões aninhadas.

Uma *hiperbase pública, Hb,* é um nó de contexto contendo nós terminais e nós de contexto de usuário. Todos os nós em Hb devem ser pemanentes ou obsoletos e, se um nó de contexto de usuário *C* está em Hb, então todos os nós contidos em *C* devem também estar contidos contidos em Hb.

Uma *base privada* é também definida como um nó de contexto que abriga qualquer entidade, exceto nós de contexto de versões e a hiperbase pública, tal que:
i) uma base privada pode pertencer a no máximo uma base privada;
ii) se um nó de composição N está contido na bse privada PB, seus componentes ou estão contidos em PB, ou na hiperbase pública, ou em qualquer base privada de um aninhamento de bases privadas contido em PB; e
iii) se um elo está contido em uma base privada, o nó base de seu ponto terminal de origem deve ser um nó de anotação (ver subseção abaixo).

Intuitivamente, uma base privada abriga todas as entidades usadas pelo usuário durante uma sessão de trabalho.

Através de primitivas, o usuário pode mover um nó terminal ou de contexto de usuário de uma base privada para a hiperbase pública, desde que o nó seja permanente. Se o nó movido for de contexto de usuário, então todos os nós contidos em C devem também ser movidos para Hb. Se, ainda, tratar-se de nó de contexto de versões, então a movimentação para Hb só precisa ser realizada quando ocorrer alguma modificação no conteúdo original. Se após a criação de uma versão V de um nó N de Hb em uma PB, V não sofrer qualquer modificação, V é simplesmente destruída ao ser movida para Hb, pois não há a necessidade de replicação de nós. Os nós de composição que contêm V devem ser atualizados para conterem agora N, ao invés de V. Da mesma forma, todas as versões criadas a aprtir de V devem ser transformadas em versões de N no contexto de versões apropriado.

Deve-se notar que não se pode migrar um nó de uma hiperbase pública para uma base privada; mas versões deste nó podem ser criadas em diversas bases privadas. Assim, no HyperProp, manipular um hiperdocumento já existente implica na criação de novas versões na base privada correspondente

(sempre a partir de nós permanentes da hiperbase), ou ainda na criação de novos nós terminais, pertencentes apenas à base privada.

O ambiente HIP/Windows destina-se ao autor de uma hiperbase privada. Isto significa que, neste primeiro protótipo, todas as operações de edição sobre os elementos da hiperbase são permitidas. Trabalho posterior será dedicado a implementar as funções necessárias para o gerenciamento da hiperbase pública e consequente construção de bases privadas a partir da hiperbase pública. No contexto educacional a que nos propomos, isto significa dizer que o ambiente HIP terá como usuário um professor/especialista em algum domínio de conhecimento, que estará construindo um hiperdocumento (hiperbase pública) naquele domínio. Através do segundo módulo, um professor ou monitor, não necessariamente especialista no domínio, poderá se basear na hiperbase existente para criar um hiperdocumento (base privada) que se adeque ao modelo de sessão de estudo que pretende criar. Um terceiro e último módulo cuidará da interface com o usuário-estudante.

Maiores detalhes sobre versionamento e criação de bases privadas serão introduzidos e discutidos quando da documentação do segundo módulo do SASHE, e também podem ser obtidos em [9].

2.2.4. Nós de Anotação

Em aplicações para trabalho cooperativo, ou ainda em situações onde várias versões podem ser feitas a partir de uma única hiperbase pública, como será o caso de nosso ambiente voltado ao ensino, é usual a existência de comentários, em qualquer formato ou mídia, que fazem referências às versões objetos do comentário e às versões consideradas respostas ao comentário.

No MCA esses comentários aparecem na forma de nós de anotação, que são nós de contexto que agrupam um conjunto de elos, nós terminais, nós de contextos de usuários e trilhas. Intuitivamente, elos de uma base privada podem ligar os componentes de um nó de anotação a nós da mesma base privada, indicando nós que originaram o comentário e nós de resposta aos comentários. Desta forma, anotações permitem a criação de observações a um nó permanente, sem que novas versões sejam criadas.

2.3. O Modelo de Apresentação do HyperProp

Uma vez que o MCA define um hiperdocumento como uma estrutura de dados passiva, faz-se necessário definir um modelo complementar de apresentação, que forneça embasamento para ferramentas para que o usuário possa ter acesso à estrutura, exibí-la, manipulá-la e navegar por ela.

Alguns conceitos importantes para a navegação serão apresentados aqui. Mais detalhes sobre esse assunto podem ser encontrados em [9] e [10].

A noção de perspectiva de um nó contribui para o problema de se ter acesso a um nó que esteja presente em diferentes composições. Ou seja, é necessário que exista uma forma de identificar sob que sequência de composições aninhadas um dado nó está sendo pesquisado, e que elos de fato tocam este nó a partir do aninhamento.

Uma perspectiva de um nó N é uma sequência $P=(N_1,...,N_m)$, $m \geq 1$ tal que $N_1 = N$, N_{i+1} é um nó de composição e N_i está contido em N_{i+1}, para $i \in [1,m)$. Dizemos que N_i está recursivamente contido em N_{i+1}, N_{i+2}, ...N_m. Note que podem existir várias perspectivas para um mesmo nó N, se este nó estiver presente em mais de uma composição. A perspectiva corrente é aquela usada na última navegação para alcançar o nó.

O modelo Dexter [5] fundamenta a interação do usuário com um hiperdocumento no MCA. *Instâncias* do modelo Dexter são versões no MCA, presentes nas bases privadas, que correspondem às *work sessions* daquele modelo.

No MCA, cada nó terminal ou de contexto de usuário tem associado a si um conjunto de especificações de apresentação do nó, assim como no modelo Dexter. Para cada tipo de mídia, essas informações em nós terminais indicam particularidades de apresentação, como por exemplo, fonte de palavras em um texto, altura do som, velocidade do vídeo, etc. Como a apresentação também pode variar de acordo com o contexto em que o nó pertence, e portanto, de acordo com sua perspectiva, os elos também possuem um conjunto de especificações de apresentação de seus respectivos nós destinos. Analogamente, nós de contexto de usuário podem guardar informações sobre o sincronismo de apresentação dos nós nele contidos. Num nível mais externo, cada aplicação que utilize a hiperbase pode definir explicitamente essas especificações para cada nó. Neste caso, na apresentação de um nó, esta especificação se sobrepõe àquela definida pelo nó de contexto de usuáio que contém o nó, que se sobrepõe à especificação de apresentação definida pelo elo usado para atingir o nó, que por sua vez se sobrepõe àquela definida pelo criador do nó. Além disso, se nenhuma dessas for definida, o sistema deve prever uma especificação *default*. Finalmente, as especificações de apresentação podem ser armazenadas como um atributo das entidades, ou alternativamente, como um nó terminal.

HyperProp oferece suporte para pelo menos cinco tipos de navegação: em profundidade, por elos, por consulta, em browsers e em trilhas.

Nagega-se em profundidade quando segue-se o aninhamento de nós de composição, o que permite ao usuário subir e descer na hierarquia de composições. Essencial à noção de hipermídia, a navegação por elos também é fornecida. Através de consultas, um usuário também pode alcançar um nó específico, descrevendo propriedades que ele deve satisfazer. A navegação em browsers é outra forma pré-definida de navegação no HyperProp. Um browser apresenta uma visão pictorial hierárquica do hiperdocumento ou de parte dele. A navegação em trilha possibilita o usuário seguir uma trilha previamente estabelecida por outrém ou em sessão de trabalho anterior. Como

uma trilha é uma lista ordenada de nós, um usuário pode selecionar *o próximo, o anterior, o primeiro*. Uma trilha especial, *system private base trial* grava toda a navegação realizada durante uma sessão, de forma que o usuário pode mover-se aleatoriamente de nó a nó e retornar, passo a passo.

Concluindo, o MCA também permite extensões para acomodar a noção de estruturas virtuais (conteúdos de nós, âncoras e elos), isto é, estruturas que resultam da avaliação de alguma expressão. Assim, um nó virtual pode ter seu conteúdo e âncoras computados quando de sua seleção durante o processo de navegação. Em particular, uma âncora pode ser computada quando um elo que a aponta for selecionado. Analogamente, um elo pode ter seus pontos terminais computados quando selecionado.

3. O Ambiente de Autoria de Hiperbases - Hip/Windows

O Hip, versão para Windows, é um sistema para desenvolvimento de aplicações hipermídia que faz parte do ambiente SASHE, como primeiro módulo de autoria de hiperdocumentos (fig. 1). Através de sua interface gráfica, um usuário pode criar e navegar sobre um hiperdocumento (uma hiperbase no MCA). A Figura 3 mostra a arquitetura básica do Hip/Windows.

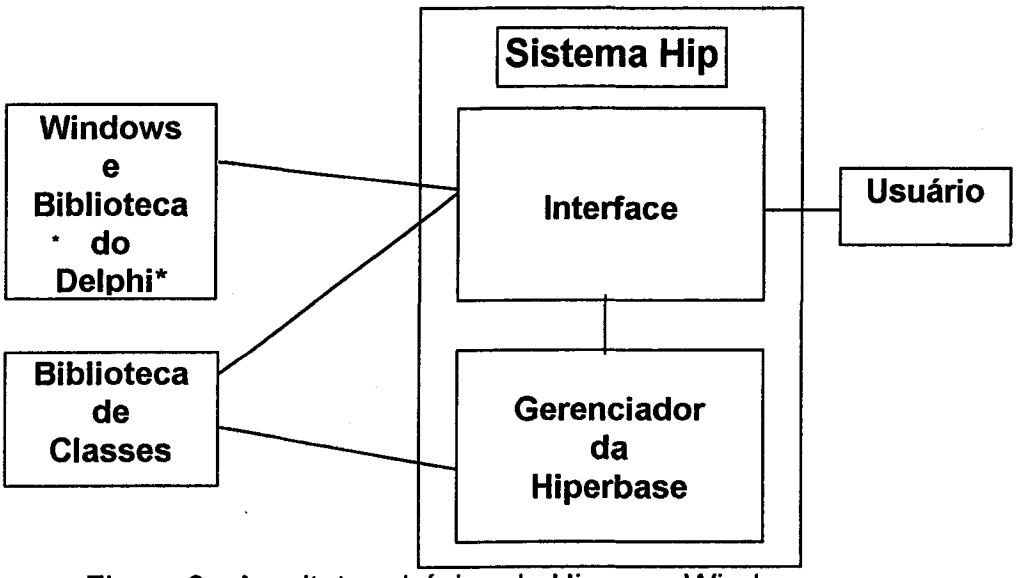


Figura 3 - Arquitetura básica do Hip para Windows.

O módulo Interface é responsável pela interação do sistema com o usuário. No seu desenvolvimento foram utilizados os recursos oferecidos pela linguagem Borland Delphi* . A Biblioteca de Classes contém as funções utilizadas na criação dos objetos que seguem o MCA, tais como: nós de contexto, contexto de versões, texto, vídeo, imagem, áudio, âncoras e elos. O Gerenciador é responsável pela manipulação dos objetos que compõem a hiperbase.

* Borland Delphi - Ferramenta para Desenvolvimento de Aplicações em Ambiente Windows.

As seções seguintes apresentam algumas características dos módulos citados.

3.1. A Interface

Ao iniciar, o Hip apresenta uma janela principal, de onde podem ser alcançadas todas as informações da hiperbase (fig. 4).

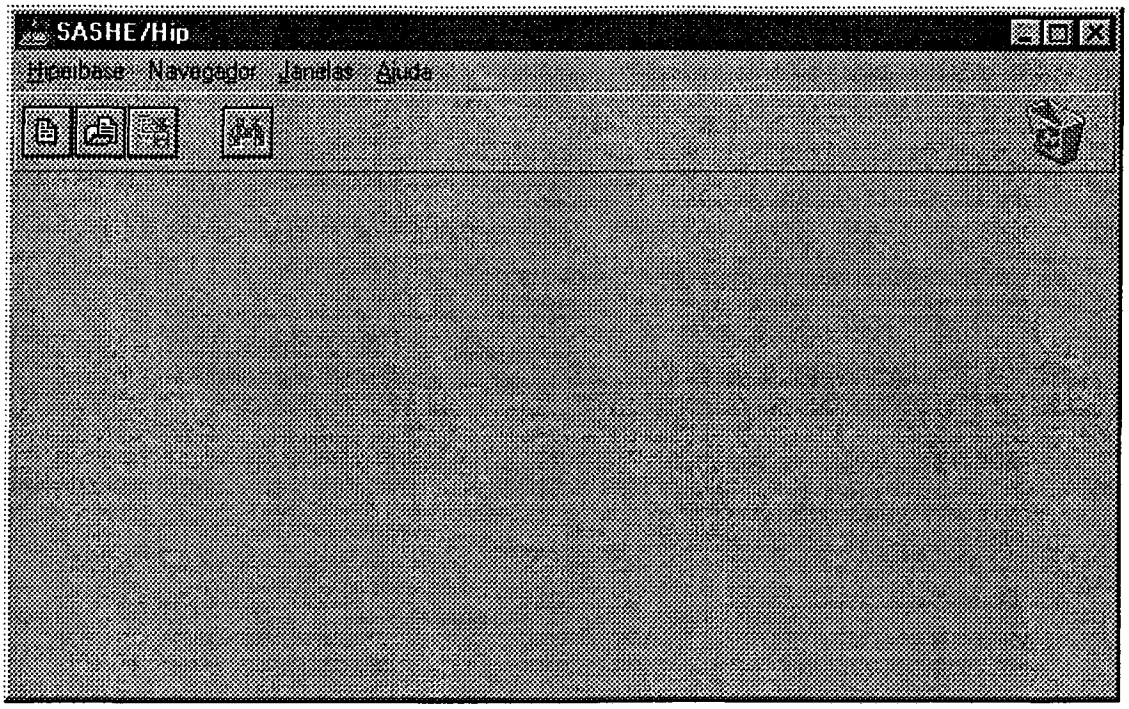


Figura 4 - Janela principal.

Criar, abrir e gravar são as operações básicas que podem ser aplicadas sobre uma hiperbase. Ao criar ou abrir uma hiperbase, suas informações são apresentadas através de uma janela secundária, denominada janela *Nós e Elos.* (fig. 5).

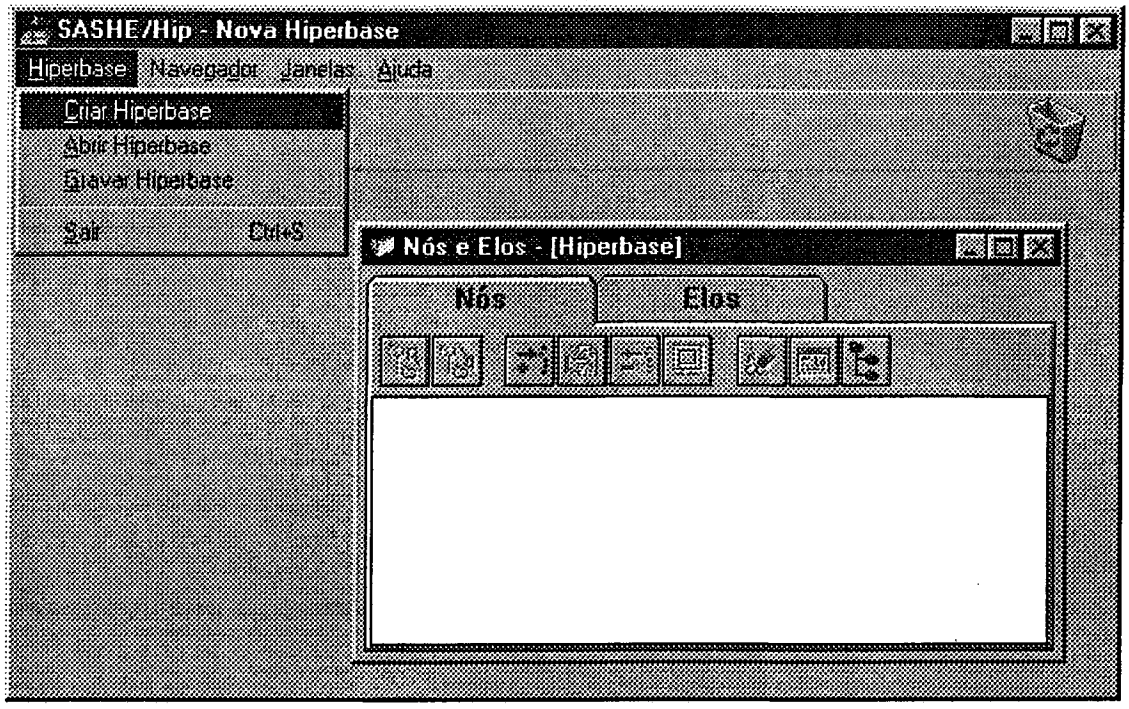


Figura 5 - Janela principal com uma hiperbase criada ou aberta.

Sempre que o usuário desejar, ele poderá gravar o conteúdo da hiperbase. Para isso, basta apertar o botão Gravar Hiperbase (ou a opção Hiperbase|Gravar Hiperbase do menu principal). Em seguida surge uma caixa de diálogo para entrada do nome do arquivo que conterá as informações (fig. 6).

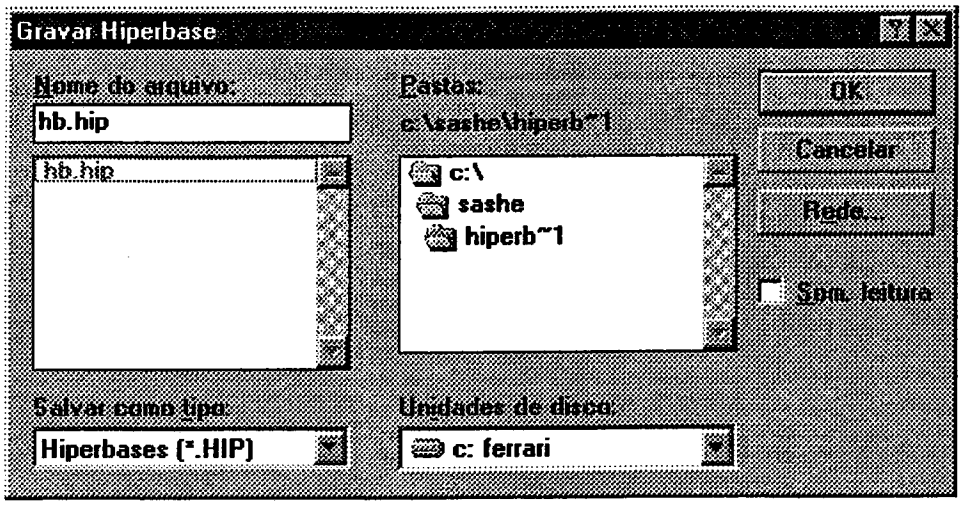


Figura 6 - Gravando uma hiperbase.

3.1.1. A Janela *Nós e Elos*

Esta é a janela mais importante do sistema, já que todas as operações realizadas sobre os nós e elos estão localizadas nela.

A escolha sobre qual dos elementos (Nós ou Elos) o usuário deseja visualizar é feita através dos botões em forma de uma "pasta".

Para cada uma das "pastas" há uma barra de ferramentas associadas às operações que podem ser aplicadas sobre os nós e elos, e uma lista onde são apresentados os nomes dos nós e dos elos pertencentes ao contexto atual. Para saber qual o contexto atual de uma janela *Nós e Elos,* basta verificar o seu título.

Outra forma de ativar as operações é apertar o botão direito do mouse dentro das listas.

Como no modelo é permitida a criação de vários nós de contexto, e cada janela representa um contexto, também é permitida a abertura de mais de uma janela *Nós e Elos*, sendo que cada uma possui seu próprio contexto atual (fig. 7).

As operações apresentadas na barra de ferramentas e no menu da "pasta" *Nós* são as seguintes:

1. Retorna ao contexto anterior: habilitada quando o contexto atual não for o de mais alto nível.

2. Visualiza o nó de contexto selecionado: habilitada quando o nó selecionado na lista for de contexto.

3. Cria um novo nó na hiperbase e o coloca no contexto atual da janela *Nó* ativa: sempre habilitada.

4. Altera as informações do nó selecionado: habilitada quando há um nó da lista selecionado.

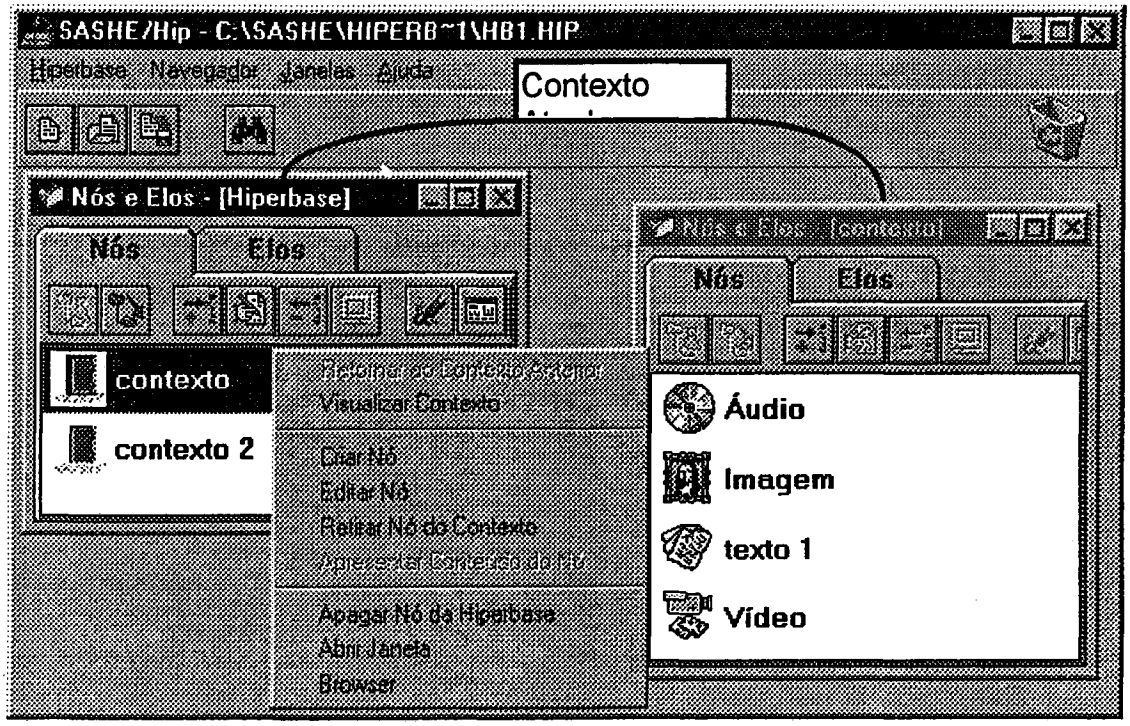


Figura 7 - Janela *Nós e Elos*

5. Retira o nó selecionado do contexto atual: habilitada quando há um nó da lista selecionado.

6. Apresenta o conteúdo do nó selecionado: habilitada quando o nó selecionado na lista for terminal.

7. Apaga o nó selecionado da hiperbase: habilitada quando há um nó da lista selecionado.

8. Abre uma nova janela *Nós e Elos* com o contexto atual igual ao nó contexto selecionado: habilitada quando o nó selecionado na lista for de contexto.

9. Apresenta a janela *Browser*: sempre habilitada.

As operações 1 e 2 são utilizadas na navegação por contextos. Neste modo de navegação, o usuário pode descer e subir através dos nós de contexto apresentados na lista de nós.

Segundo o MCA, um nó pode estar contido em vários nós de contexto, ou seja, cada contexto contém uma referência ao nó (imagem do nó), porém, fisicamente, existe apenas um nó armazenado na hiperbase. Portanto, quando o usuário desejar retirar um determinado nó do contexto atual, a operação 5 deve ser executada. O Hip, após esta operação, verifica se o nó retirado não pertence a nenhum outro contexto. Se não pertencer, é colocada para o autor a opção de apagar o nó ou deixá-lo isolado. Os nós isolados não estão contidos na hiperbase, mas podem ser recuperados. Quando o autor sair do Hip, os nós isolados são eliminados. Se ele pertencer a outro contexto, apenas a referência ao nó será eliminada.

A operação 6 tem como função apresentar o conteúdo do nó terminal selecionado. Vale ressaltar que o Hip permite a apresentação simultânea de vários nós terminais. A Figura 8 apresenta um exemplo desta propriedade.

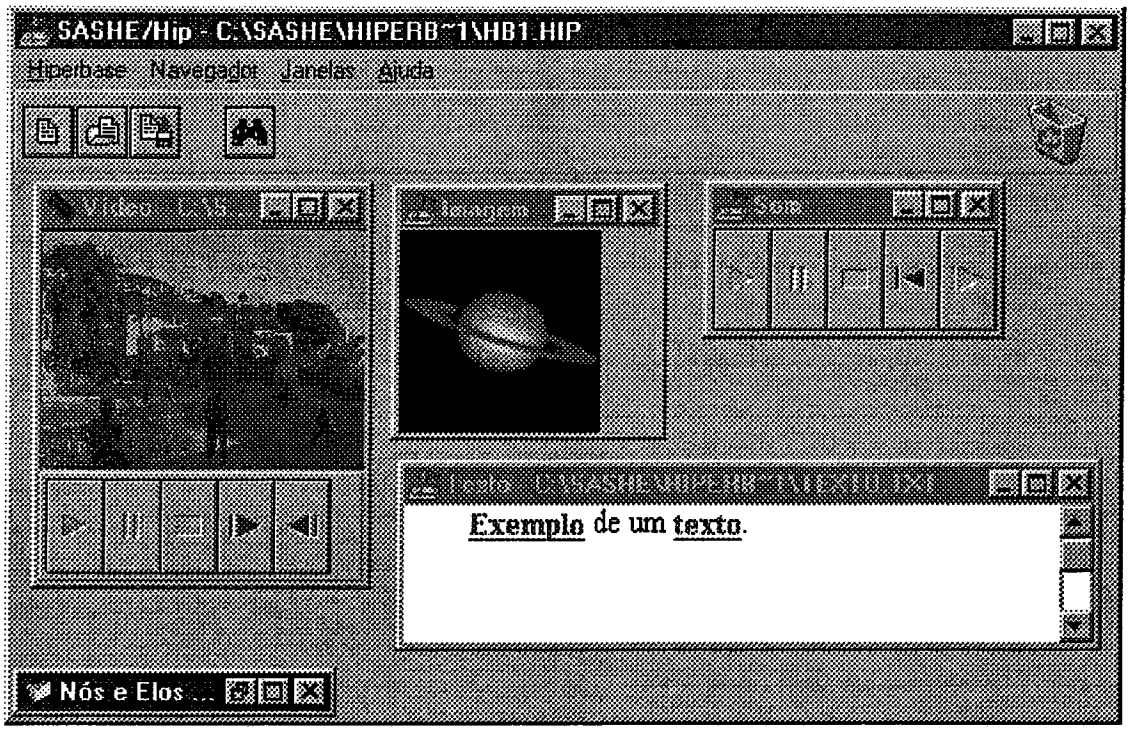


Figura 8 - Apresentação de nós terminais.

Se o usuário desejar apagar um nó da hiperbase, a operação 7 deverá ser utilizada. Depois de concluída esta operação, não é possível recuperar o conteúdo do nó. Caso o nó seja de contexto, no entanto, os nós contidos nele não são eliminados.

Para visualizar um nó de contexto sem abandonar o contexto atual de uma janela *Nós e Elos*, deve-se selecionar o nó e executar a operação 8 para abrir uma nova janela *Nós e Elos*. Esta janela é idêntica à anterior, porém com o contexto atual igual ao nó selecionado anteriormente (fig. 9).

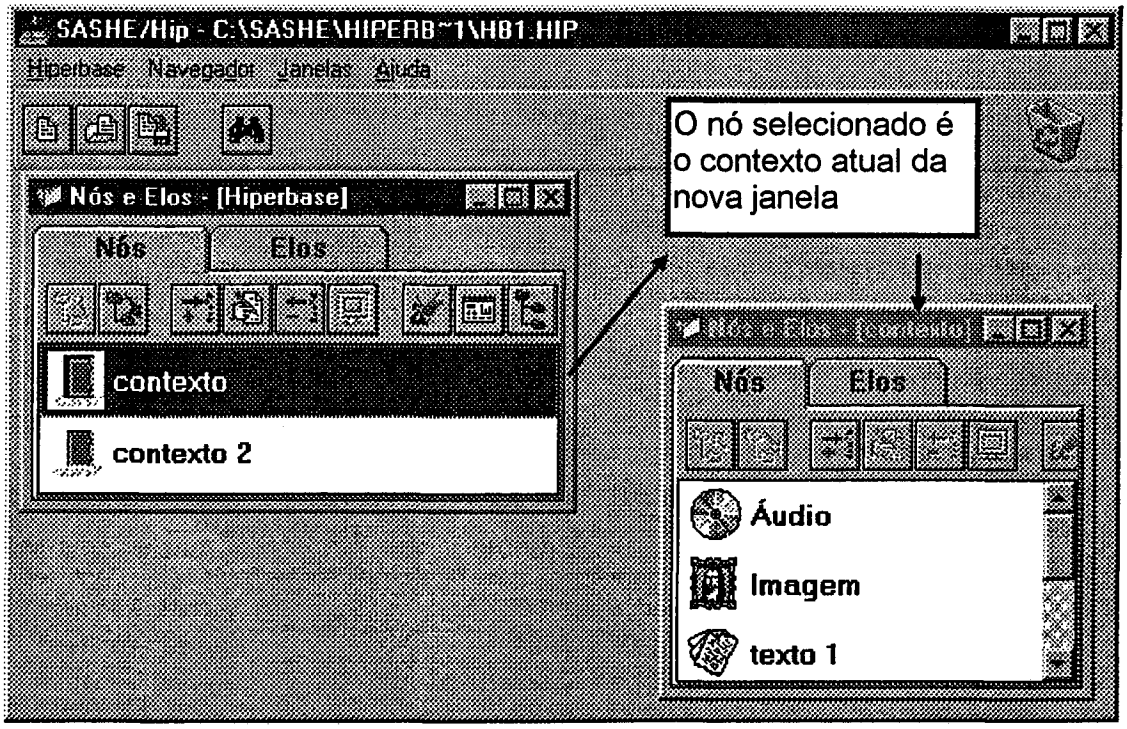


Figura 9 - Cada janela *Nós e Elos* representa um nó de contexto.

Para visualizar todos os contextos e seus respectivos nós, o usuário executa a operação 9, abrindo a janela *Browser*. São apresentados todos os nós que estão abaixo do contexto atual da janela *Nós e Elos* de onde foi executado o *Browser* (fig. 10).

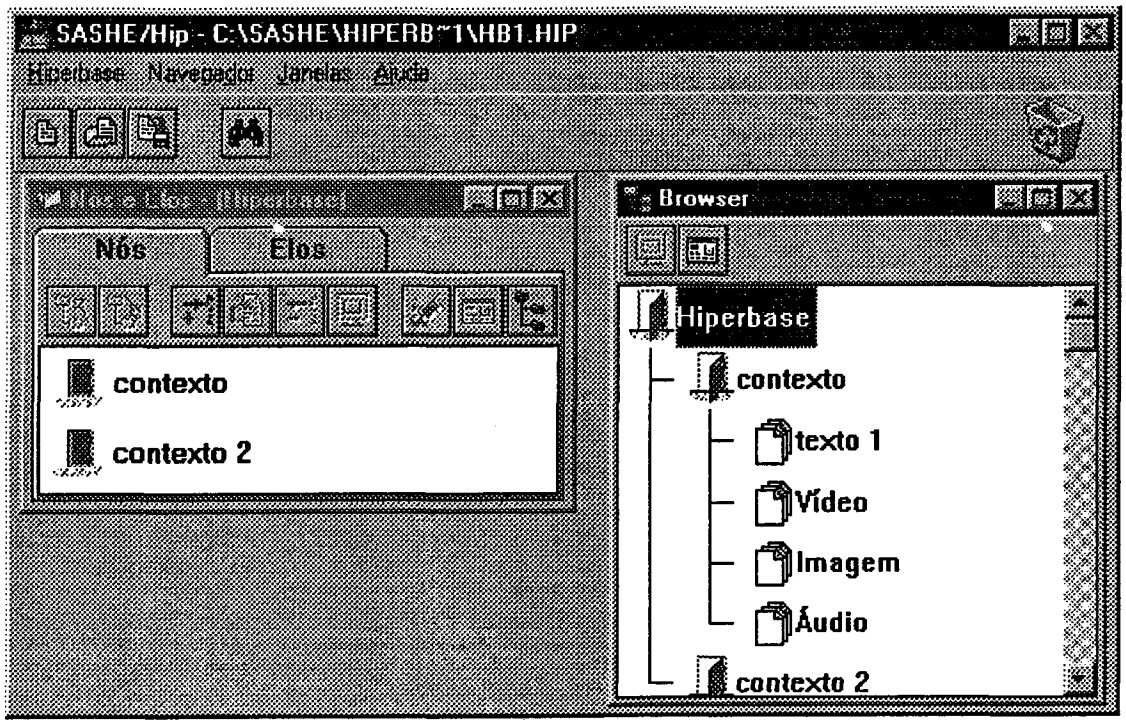


Figura 10 - *Browser*.

Além de apresentar todo aninhamento dos nós que estão abaixo do contexto atual (fig. 10), o browser permite a execução das operações 6 (Apresentar Conteúdo do Nó) e 8 (Abrir Janela).

Com a operação 3 pode-se criar um nó dentro do contexto atual da janela *Nós e Elos* ativa.

Para concluir a operação, o usuário deve fornecer o nome do nó e seu tipo: contexto, terminal ou trilha (ainda não disponível). Se o tipo do nó for terminal, mais duas informações devem ser escolhidas: tipo do nó terminal (texto, áudio, vídeo ou imagem) e seu arquivo de conteúdo. Este último também pode ser preenchido na edição do nó (Operação 4). A Figura 11 ilustra a janela de criação de nós.

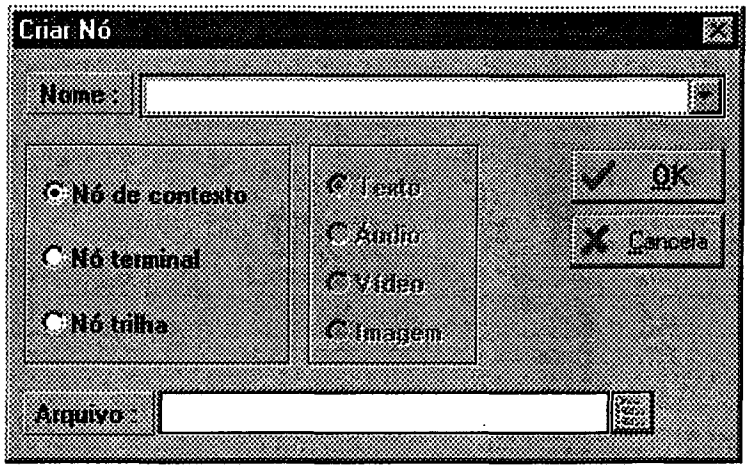


Figura 11 - Criação de nós.

Na operação 4 (fig. 12) é possível alterar-se o nome do nó e o arquivo de conteúdo, caso o nó seja terminal. Além disso, a opção *Editar conteúdo* permite modificar as informações contidas no interior do arquivo de conteúdo. Inicialmente, os nós terminais que possuem esta opção são os nós texto e imagem.

Para edição de texto é usado um editor contendo as principais funções de edição utilizadas no Windows, além de permitir a criação de âncoras. Atualmente, as imagens são editadas pelo editor gráfico *PaintBrush*[*] .

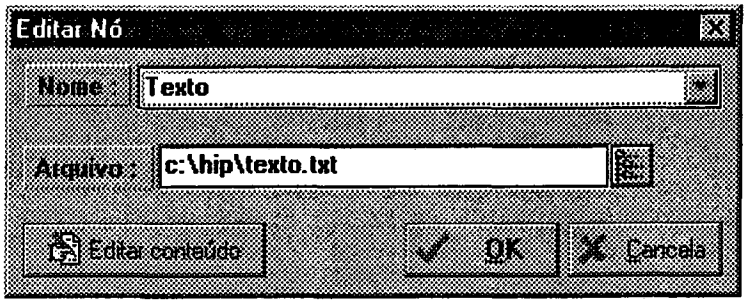


Figura 12 - Edição de nós

[*] PaintBrush - Editor Gráfico do Windows.

Além das operações descritas anteriormente, o usuário pode copiar e/ou mover nós de um contexto para outro.

Para mover um nó, basta ter aberto na janela principal duas janelas *Nós e Elos*, onde cada uma tem como contexto atual os nós de contexto sobre os quais será efetuada a operação. A seguir o usuário deve, através do mouse, mover o nó desejado da janela origem à janela destino.

Se durante a operação a tecla *shift* estiver pressionada, será realizada a cópia do nó. É necessário que a tecla seja pressionada antes da utilização do mouse.

3.1.2. Criando Âncoras

Depois de criada, pode-se ter acesso à hiperbase através dos seguintes modos: navegação por contexto, navegação por elos e navegação por consulta.

Na navegação por contexto o usuário pode descer e subir através dos contextos criados na hiperbase e, assim, ter acesso às informações contidas nos nós terminais.

Se o usuário desejar selecionar um grupo de informações sobre um determinado assunto para posteriormente processá-las, a navegação por consulta deve ser escolhida (opção ainda não disponível).

Finalmente, a navegação por elos é colocada como o principal modo de acesso à hiperbase, pois permite a ligação entre dois ou mais nós pertencentes a hiperbase. Esta ligação é realizada pelo elo, que na seção 3.1.3, é tratado com mais detalhes.

Nesta fase do projeto, apenas os nós texto podem estar ligados a outros nós (nós texto ou nós de contexto). Estas ligações podem ser ativadas através de regiões do texto denominadas **âncoras**. O autor da hiperbase é responsável por esta marcação.

Para criar âncoras, inicialmente o autor seleciona o nó texto desejado e, na janela *Editar Nó* (fig. 12), seleciona a opção *Editar conteúdo* para abrir a janela de edição do texto. A seguir, basta selecionar a região do texto desejada e pressionar o botão para criar a âncora.

3.1.3. Criação de Elos

A barra de ferramentas localizada na "pasta" *Elos* contém todas as operações que o o usuário pode aplicar sobre os elos, tais como, *criar elo* e *apagar elo*. A opção *editar elo* será adicionada futuramente. Além disso, é possível visualizar todos os elos criados no contexto atual da janela *Nós e Elos* em questão (fig. 13).

Como foi dito anteriormente, um elo faz a ligação entre dois ou mais nós. No primeiro caso de ligação (1 para 1 ou 1:1), a origem e o destino possuem apenas um nó. No segundo caso (1 para Muitos - 1:M), a origem possui um nó e o destino mais de um nó, quando se denomina elo de múltiplos destinos. A versão atual do Hip permite a criação de elos 1:1. O segundo caso (1:M) será tratado futuramente.

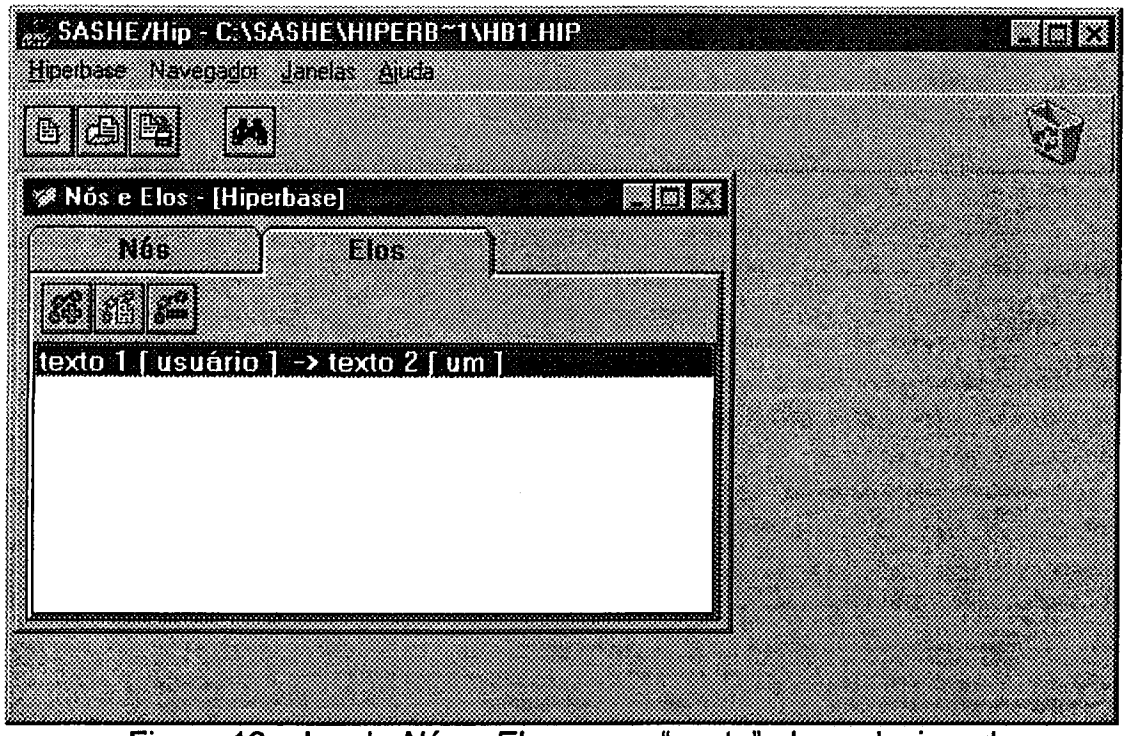


Figura 13 - Janela *Nós e Elos* com a "pasta" elos selecionada.

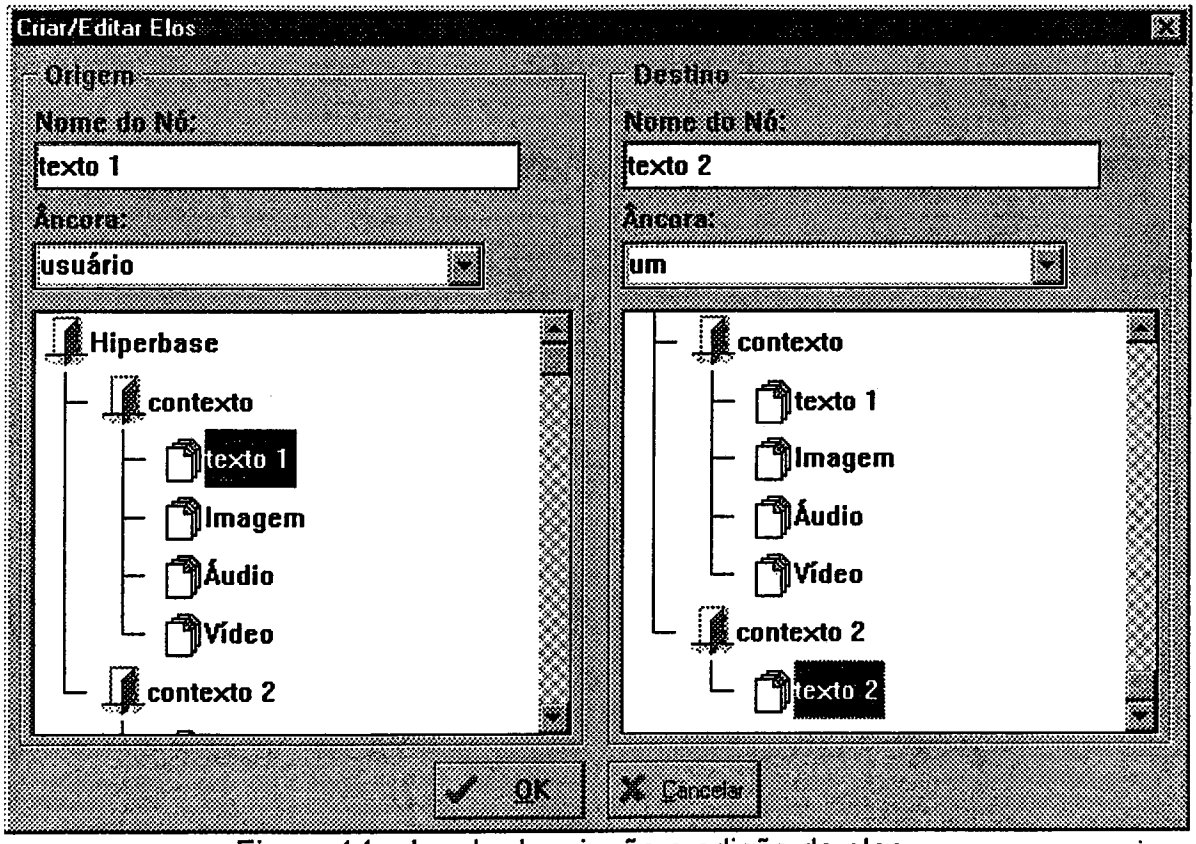


Figura 14 - Janela de criação e edição de elos.

Nas extremidades de cada elo estão localizadas as âncoras dos nós origem e destino, respectivamente. Portanto, o usuário deve escolher como origem um nó texto e uma de suas âncoras, e como destino um nó de contexto ou os mesmos atributos que a origem (fig. 14). Quando o destino é um nó de

contexto, o sistema se encarrega de criar a âncora contexto para o nó escolhido.

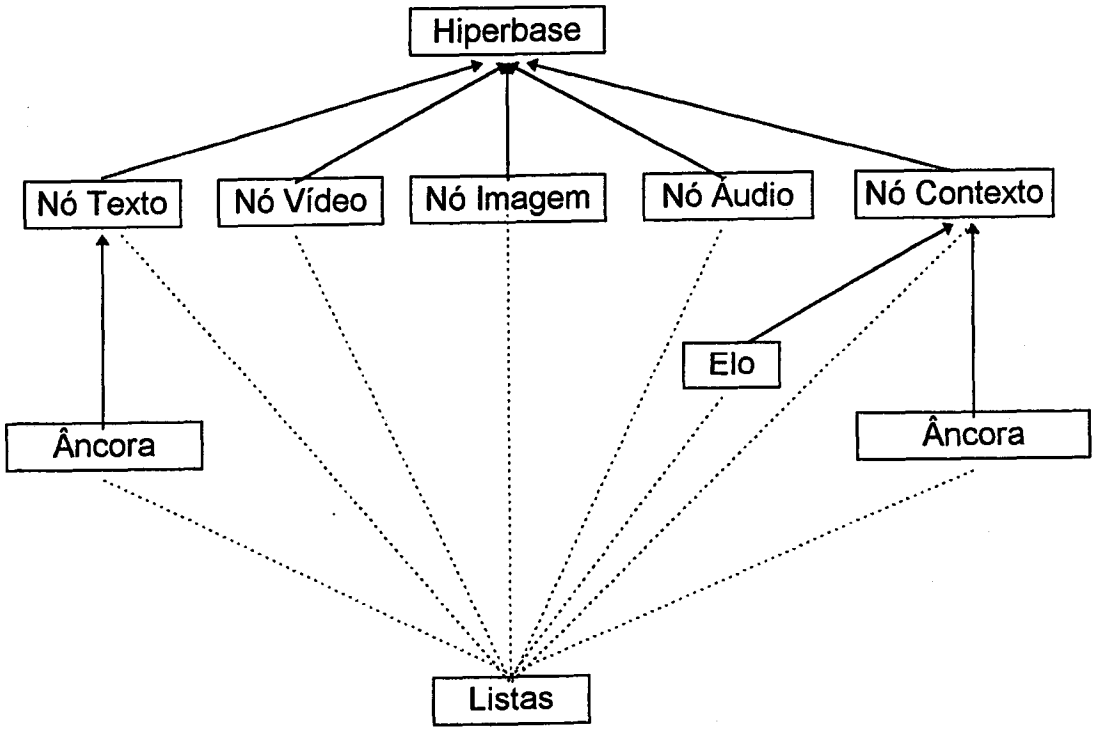
4 - A Biblioteca de Classes

A biblioteca de classes é o único módulo totalmente independente dos outros módulos do ambiente. Ela tem por finalidade implementar as abstrações definidas pelo modelo conceitual de Contextos Aninhados, ou seja, nela encontram-se as classes responsáveis pela criação, manipulação e eliminação dos objetos da hiperbase.

Assim, a biblioteca de classes oferece basicamente as seguintes operações:

- criar e eliminar nós na hiperbase;
- criar e eliminar âncoras em um nó, tal operação depende da classe do nó;
- criar e eliminar elos em um nó contexto;
- inserir e retirar nós em um nó contexto;
- mudar o valor de uma âncora e
- mudar o estado do nó.

Na figura 15 são apresentadas as principais classes de objetos que compõem a biblioteca de classes e sua hierarquia.



Notação:

⟶ contém

····· lista de

Figura 15 - As principais classes de objetos e sua hierarquia.

Conforme visto na seção 2.1, o MCA é composto por uma hierarquia de objetos e para cada objeto são definidos operações e um conjunto de atributos. Assim, nada mais natural do que implementá-las usando uma linguagem orientada para objetos [12]. Assim, o código fonte da biblioteca de classes foi implementado utilizando a linguagem C++ e foram definidas classes com seus atributos e métodos e outros tipos abstratos de dados auxiliares, como listas, para a construção dos objetos do MCA. Listas de nós, de elos e de âncoras são implentadas usando o conceito de herança.

Também foram definidas outras classes genéricas, chamadas de classes-base, que possuem características compartilhadas pelas classes derivadas. Estas classes-bases são as classes Nó e Âncora. As classes derivadas de Nó são as classes nó áudio, nó vídeo, nó texto, nó imagem e nó contexto, e as classes derivadas de Âncora são as classes âncora texto e âncora contexto. Como a interface atual do SASHE não possui recursos de edição de vídeo e áudio, não foram implementadas as classes âncora vídeo, imagem e áudio. Estas classes necessitam de alguns mecanismos que permitam a edição das mídias vídeo, imagem e áudio.

Para suportar algumas características necessárias ao ensino, algumas alterações foram feitas nas classes de objetos:

- na classe Nós (texto, áudio, vídeo, imagem e contexto) foi acrescentada uma lista que contém o nome dos nós contextos nos quais cada nó está contido;
- na classe Âncoras foi acrescentado um identificador utilizado somente pela interface, chamado de *identificador externo*, que é independente do identificador definido pelo MCA, que passou a ser chamado de *identificador interno* e que é gerado automaticamente quando uma âncora é crida. Esta separação foi necessária para que a interface pudesse manipular e alterar o identificador externo sem acrescentar inconsistência na hiperbase. Por exemplo, se a interface alterasse o identificador interno de uma determinada âncora que fosse a origem ou destino de algum elo, este passaria a não mais ancorar naquela âncora, pois o identificador guardado pelo elo não seria o mesmo da âncora;
- para maior comodidade ao usuário do SASHE, foi acrescentada uma lixeira que contém uma lista de nós excluídos da hiperbase pelo usuário durante uma seção de trabalho. Assim, o usuário pode recuperar os nós que foram excluídos por engano;
- o conteúdo dos nós texto, vídeo, imagem e áudio são gravados em arquivos separados dos outros atributos destes nós. Assim, acreditamos que tornaremos o sistema mais portável para diferentes sistemas operacionais e otimizado quando for adaptado em um ambiente distribuído;

5 - O Gerenciador da Hiperbase

O gerenciador utiliza as operações oferecidas pela biblioteca de classes e gerencia a criação, destruição e modificação dos objetos da hiperbase. Ele funciona como um intermediário entre a interface e a biblioteca de classes.

As rotinas que compõem o gerenciador são ativadas pelo usuário através da interface quando aquele quer realizar tarefas que alteram o estado dos objetos na hiperbase. Essas rotinas se encarregam de verificar se determinadas condições estão satisfeitas para que a tarefa possa ser realizada.

Este gerenciamento é necessário para que a biblioteca de classes tenha uma arquitetura mais enxuta, evitando que seus métodos tenham a preocupação de verificar o estado de outros objetos. Por exemplo, durante a criação de elos, o gerenciador verifica se o contexto, os nós e âncoras origem e destino de fato existem para que a operação seja válida.

Se o gerenciador não existisse, o gerenciamento ficaria a cargo da interface ou da biblioteca de classes. Ficando a cargo da interface, esta passa a ter uma maior dependência do modelo, sendo encarregada não só da sua visualização, mas também do controle de operações. Se o gerenciamento fosse de responsabilidade da biblioteca de classes, seus métodos ficariam com algoritmos mais complexos e com mais interdependência entre si.

Além disso, para as operações que não satisfazem as condições necessárias, mensagens deveriam ser enviadas ao usuário. Isso aumentaria a interação da biblioteca de classes com a interface.

A figura 16 mostra os três módulos do SASHE - interface, biblioteca de classes e gerenciador da hiperbase. A biblioteca de classes e o gerenciador da hiperbase fazem parte de uma biblioteca de ligações dinâmicas (também chamadas de DLL, *Dynamic Link Libraries*). A biblioteca de ligações dinâmicas geralmente não é executada diretamente. Ela é um arquivo separado que contém funções que podem ser chamadas por programas e outras DLLs para executar determinadas tarefas [14]. A biblioteca de ligações dinâmicas do SASHE é trazida à atividade quando o módulo de interface chama uma das rotinas na biblioteca. Na versão para Windows, o gerenciador é quem possui as principais rotinas manipuladas pela interface. Estas rotinas são exportadas pela DLL e visíveis pela interface.

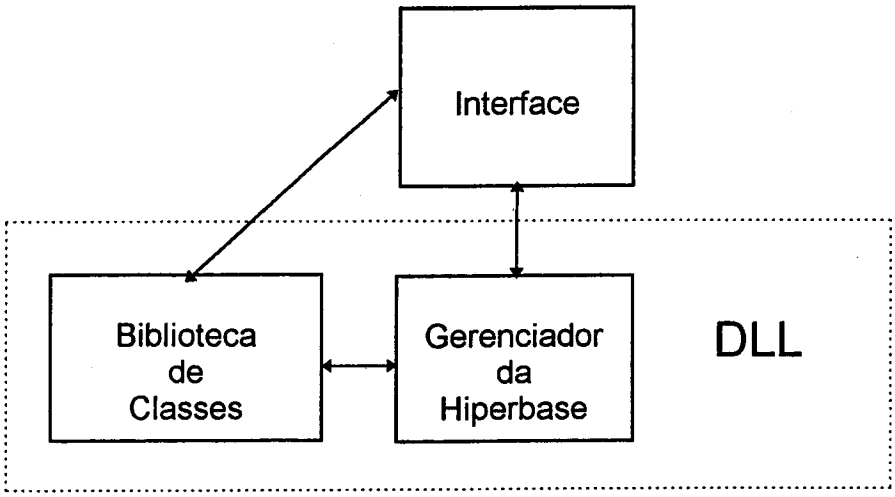


Figura 16 - Os principais módulos do SASHE comunicando-se por meio de chamadas de rotinas.

O gerenciador da hiperbase foi implementado em linguagem C para facilitar a exportação de suas rotinas pela DLL. Como C++ é uma extensão da linguagem C, foi utilizado o mesmo compilador no desenvolvimento de ambos os módulos, o Visual C++ da Microsoft [13].

As rotinas exportadas pela DLL que fazem parte do gerenciador da hiperbase e da biblioteca de classe são:

int salvar(char *nome)
char *nome: nome do arquivo que vai conter a hiperbase.
Retorna: 0: Ok.
caso contrário: Erro.
Descrição: Salva os atributos da hiperbase em arquivo.

int recuperar(char *nome)
char *nome: nome do arquivo que contém a hiperbase.
Retorna: 0: Ok.
caso contrário: Erro.
Descrição: Recupera os atributos da hiperbase gravados em arquivo.

int primeiro_no_hiperbase(char *nome)
Retorna: **char *nome:** nome do primeiro nó na lista da hiperbase.
0: Ok.
caso contrário: Lista vazia.
Descrição: Coloca na variável **nome** o nome que identifica o primeiro nó na lista de nós da hiperbase.

int proximo_no_hiperbase(char *nome)
Retorna: **char *nome:** nome do próximo nó na lista da hiperbase.
0: Ok.
caso contrário: Fim da lista ou lista vazia.
Descrição: Coloca na variável **nome** o nome que identifica o próximo nó na lista de nós da hiperbase.

int limpa_hiperbase()
Retorna: sempre 0.
Descrição: Apaga todos os nós, âncoras e elos da hiperbase e libera a memória utilizada pela hiperbase.

int retorna_identificador(char *nome)
char *nome: nome que identifica o nó na hiperbase.
Retorna: o identificador do nó.
-1: se o nó com este nome não existe na hiperbase.
Descrição: Dado o nome do nó esta rotina retorna o identificador do nó.

int retorna_nome(int ident, char *nome)
int ident: identificador do nó na hiperbase.
Retorna: **char *nome:** nome do nó na hiperbase

0: Ok.
1: se o nó com este identificador não existe na hiperbase.
Descrição: Dado o identificador do nó esta rotina retorna o nome do nó.

int cria_contexto(char *nome)
char *nome: nome que identifica o nó criado.
Retorna: 1: Um nó com este nome já existe na hiperbase.
0: Ok.
Descrição: Cria um nó contexto com o nome passado como argumento.

int cria_texto(char *nome)
char *nome: nome que identifica o nó criado.
Retorna: 1: Um nó com este nome já existe na hiperbase.
0: Ok.
Descrição: Cria um nó texto com o nome passado como argumento.

int cria_video(char *nome)
char *nome: nome que identifica o nó criado.
Retorna: 1: Um nó com este nome já existe na hiperbase.
0: Ok.
Descrição: Cria um nó vídeo com o nome passado como argumento.

int cria_audio(char *nome)
char *nome: nome que identifica o nó criado.
Retorna: 1: Um nó com este nome já existe na hiperbase.
0: Ok.
Descrição: Cria um nó audio com o nome passado como argumento.

int cria_imagem(char *nome)
char *nome: nome que identifica o nó criado.
Retorna: 1: Um nó com este nome já existe na hiperbase.
0: Ok.
Descrição: Cria um nó imagem com o nome passado como argumento.

int insere_no_contexto(char *nome_cont, char *nome_no)
char *nome_cont: nome que identifica o nó contexto e que conterá o nó (nome_no).
char *nome_no: nome que identifica o nó que será inserido dentro do nó contexto (nome_cont).
Retorna: 1: Não existe nó contexto com o nome passado como argumento.
2: Não se pode inserir em nó que não seja contexto.
3: O nó contexto de versões é de responsabilidade do sistema.
4: O nó contexto não é temporário, portanto não pode ser alterado.
5: Um nó contexto não pode conter ele mesmo.
6: Não existe nó com o nome (nome_no) passado como argumento.
7: O nó (nome_no) já está no nó contexto (nome_cont), impossível inserir novamente.
0: Ok.

Descrição: Insere um nó (texto, imagem, audio, vídeo, trilha e contexto) em um nó contexto.

int retira_do_contexto(char *nome_contexto, char *nome_no)
char *nome_contexto: nome que identifica o nó contexto e que contém o nó a ser retirado.
char *nome_no: nome que identifica o nó que será retirado do nó contexto.
Retorna: 1: Não existe nó contexto com o nome passado como argumento.
2: Não se pode retirar de nó que não seja contexto.
3: O nó contexto de versões é responsabilidade do sistema.
4: O nó contexto não pode ser alterado. Não é temporário.
5: Não existe nó com o nome (nome_no) passado com argumento na hiperbase.
6: O nó passado como argumento (nome_no) não está no nó contexto.
0: Ok.
Descrição: Retira do nó contexto um determinado nó.

int primeiro_no_contexto(char *nome_cont, char *nome_no)
char *nome_cont: nome que identifica o nó contexto.
Retorna: **char *nome_no:** nome do primeiro nó na lista do nó contexto.
1: Não existe nó contexto com o nome passado como argumento.
2: O nó (nome_cont) não é um contexto.
0: Ok.
Descrição: Retorna em **nome_no** o nome do primeiro nó que pertence à lista do nó contexto.

int proximo_no_contexto(char *nome_cont, char *nome_no)
char *nome_cont: nome que identifica o nó contexto.
Retorna: **char *nome_no:** nome do próximo nó na lista do nó contexto.
1: Não existe nó contexto com o nome passado como argumento.
2: O nó (nome_cont) não é um contexto.
0: Ok.
Descrição: Retorna em **nome_no** o nome que identifica o próximo nó na lista do nó contexto.

int recupera_nome_arquivo_video(char *nome_video, char *arq_video)
char *nome_video: nome que identifica o nó do tipo vídeo.
Retorna: **char *arq_video:** nome do arquivo que contém o vídeo.
1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo vídeo.
0: Ok.
Descrição: Recupera o nome do arquivo que contém o vídeo a ser mostrado.

int insere_nome_arquivo_video(char *nome_video, char *arq_video)
char *nome_video: nome que identifica o nó do tipo vídeo.
char *arq_video: nome do arquivo que contém o vídeo.
Retorna: 1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo vídeo.
3: O nome do arquivo vídeo excedeu o tamanho máximo.

0: Ok.
Descrição: Atualiza o nome do arquivo que contém o vídeo.

int recupera_nome_arquivo_texto(char *nome_texto, char *arquivo)
char *nome_texto: nome que identifica o nó do tipo texto.
Retorna: **char *arquivo:** nome do arquivo que contém o texto.
1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo texto.
0: Ok.
Descrição: Recupera o nome do arquivo que contém o texto a ser mostrado.

int insere_nome_arquivo_texto(char *nome_texto, char *arquivo)
char *nome_texto: nome que identifica o nó do tipo texto.
char *arquivo: nome do arquivo que contém o texto.
Retorna: 1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo texto.
3: O nome do arquivo texto excedeu o tamanho máximo.
0: Ok.
Descrição: Atualiza o nome do arquivo que contém o texto.

int recupera_nome_arquivo_audio(char *nome_audio, char *nome_arq)
char *nome_audio: nome que identifica o nó do tipo audio.
Retorna: **char *nome_audio:** nome do arquivo que contém o audio.
1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo audio.
0: Ok.
Descrição: Recupera o nome do arquivo que contém o audio a ser tocado.

int insere_nome_arquivo_audio(char *nome_audio, char *nome_arq)
char *nome_audio: nome que identifica o nó do tipo audio.
char *nome_arq: nome do arquivo que contém o audio.
Retorna: 1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo audio.
3: O nome do arquivo audio excedeu o tamanho máximo.
0: Ok.
Descrição: Atualiza o nome do arquivo que contém o audio.

int recupera_nome_arquivo_imagem(char *nome_imagem, char *nome_arq)
char *nome_imagem: nome que identifica o nó do tipo imagem.
Retorna: **char *nome_arq:** nome do arquivo que contém o imagem.
1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo imagem.
0: Ok.
Descrição: Recupera o nome do arquivo que contém a imagem a ser mostrada.

int insere_nome_arquivo_imagem(char *nome_imagem, char *nome_arq)
char *nome_imagem: nome que identifica o nó do tipo imagem.
char *nome_arq: nome do arquivo que contém a imagem.

Retorna: 1: Não existe nó com o nome passado como argumento.
2: O nó não é um nó do tipo imagem.
3: O nome do arquivo imagem excedeu o tamanho máximo.
0: Ok.
Descrição: Atualiza o nome do arquivo que contém a imagem.

int tipo_no(char *nome)
char *nome: nome que identifica o nó na hiperbase.
Retorna: 0: Não existe nó com o nome passado como argumento.
1: Nó do tipo contexto.
2: Nó do tipo contexto de versões.
3: Nó do tipo texto.
4: Nó do tipo vídeo.
Descrição: Identifica o tipo do nó.

int retira_da_hiperbase(char *nome_no)
char *nome_no: nome que identifica o nó na hiperbase.
Retorna: 1: Não existe nó com o nome passado como argumento.
2: O nó contexto de versões não pode ser retirado pelo usuário.
3: O nó não pode ser alterado. Não é um nó temporário.
0: Ok.
Descrição: Retira um determinado nó da hiperbase.

int mostra_primeiro_lixo(char *nome)
Retorna: **char *nome:** nome que identifica o nó na hiperbase.
1: O nome passado como parâmetro não consta na hiperbase.
0: Ok.
Descrição: Recupera o nome do primeiro nó que foi colocado no lixo.

int mostra proximo_lixo(char *nome)
Retorna: **char *nome:** nome que identifica o nó na hiperbase.
1: O nome passado como parâmetro não consta na hiperbase.
0: Ok.
Descrição: Recupera o nome do próximo nó que foi colocado no lixo.

int muda_atributos(char *nome_old, char *nome_new, int num_estado)
char *nome_old: nome atual que identifica o nó na hiperbase.
char *nome_new: novo nome que será atribuído ao nó na hiperbase.
int num_estado: número que altera o estado do nó - os estados são (Temporário, Permanente e Obsoleto).
Retorna: 1: O nome passado como parâmetro não consta na hiperbase.
2: O nome do nó não pode ser alterado.
3: O nó não é Temporário. Tentou passar de Temporário para Permanente; mas o nó não é Temporário.
4: O nó não é Permanente. Tentou passar de Permanente para Obsoleto, mas o nó não é Permanente.
0: Ok.

Descrição: Altera os atributos do nó, trocando o nome e/ou o estado que se encontra o nó. Os número válidos para trocar o estados são:
1: se o nó for Temporário este passará para Permanente.
2: se o nó for Permanente este passará para Obsoleto.
Qualquer outro número não altera o estado do nó.

int cria_ancora_texto(char *nome_no, int ident, char *palavra)
char *nome_no: nome que identifica o nó na hiperbase.
int ident: número que é dado à âncora e serve para identificá-la no nó a qual pertence (identificador externo).
char *palavra: palavra âncora.
Retorna: 1: O nome passado como parâmetro não consta na hiperbase.
2: O nó não é temporário. Portanto, não pode ser alterado.
3: Não se pode criar ancoras em um nó versão.
0: Ok.
Descrição: Cria uma ancora texto em um nó texto.

int cria_ancora_contexto(char *nome, int ident)
char *nome: nome que identifica o nó contexto na hiperbase.
int ident: número que é dado à âncora e serve para identificá-la no nó a qual pertence (identificador externo).
Retorna: 1: Não existe nó contexto com esse nome na hiperbase.
2: Esse nó não pode ser alterado.
3: Não se pode criar ancoras em um nó versão.
4: Já existe uma âncora neste contexto.
0: Ok.
Descrição: Cria uma âncora contexto em um nó contexto.

int retira_ancora(char *nome, int ident)
char * nome: nome que identifica o nó na hiperbase.
int ident: identificador interno da âncora dentro do nó.
Retorna: 1: A lista de âncora está vazia.
2: Erro na estrutura da lista de âncoras.
3: Não existe âncora com este identificador.
4: Não existe nó com esse nome na hiperbase.
5: Esse nó não pode ser alterado.
6: Não se pode retirar ancoras de um nó versão.
0: Ok.
Descrição: Retira uma âncora da lista de um determinado nó (texto ou contexto).

int recupera_identificador_interno(char *nome, int ident)
char *nome: nome que identifica o nó na hiperbase.
int ident: identificador criado e gerenciado pela interface (identificador externo).
Retorna: -1 Não existe nó com esse nome na hiperbase.
caso contrário: Retorna o identificador interno da âncora.
Descrição: Dado o nome do nó e o identificador externo da âncora esta função retorna o identificador que é utilizado internamente pelo sistema.

int numero_de_ancoras(char *nome)
char *nome: nome que identifica o nó na hiperbase.
Retorna: Retorna o número total de âncoras.
-1: Ocorreu erro.
Descrição: Dado o nome de um nó esta rotina retorna o total de âncoras que ele contém.

int recupera_nome_ancora(char *nome, int ident, char *nome_ancora)
char *nome: nome que identifica o nó texto na hiperbase.
int ident: identificador interno da âncora;
Retorna: **char *nome_ancora:** nome marcado pelo usuário no texto como âncora.
1: Não existe nó com esse nome na hiperbase.
2: O nó não é um nó do tipo texto.
3: Não existe âncora com este identificador.
0: Ok.
Descrição: Esta rotina funciona apenas para âncoras textos. Dado o identificador interno de uma âncora, esta rotina retorna o nome marcado pelo usuário como âncora.

int cria_elo(char *nome, int id, char *origem, int anc_origem, char *destino, int anc_destino)
char *nome: nome que identifica o nó contexto na hiperbase.
int id: identificador que identifica o elo no nó contexto.
char *origem: lista de identificadores dos nós aninhados, onde o primeiro da lista é o identificador do nó origem do elo.
int anc_origem: identificador interno da âncora que encontra-se no nó origem.
char *destino: lista de identificadores dos nós aninhados, onde o primeiro da lista é o identificador do nó destino do elo.
int anc_destino: identificador interno da âncora que encontra-se no nó destino.
Retorna: 1: Esse contexto não existe na hiperbase.
2: O Nó origem não existe na hiperbase.
3: O Nó destino não existe na hiperbase.
4: Esse nó não é contexto.
5: O contexto de versões é de responsabilidade do sistema.
6: Não conseguiu alocar memória para o elo.
7: O nó origem e o nó destino são os mesmos.
0: Ok.
Descrição: Dado o nome do nó contexto e dois nós com âncoras que estão neste contexto, esta rotina é responsável pela criação de um elo entre os dois nós.

int retira_elo(char *nome, int ind_elo)
char *nome: nome que identifica o nó contexto na hiperbase.
int ind_elo: identificador do elo no nó contexto.
Retorna: 1: Não existe nó contexto com esse nome na hiperbase.
2: Esse nó não é contexto.

3: O contexto de versões é de responsabilidade do sistema.
4: Esse nó não pode ser alterado.
5: Não existe esse elo no contexto.
0: Ok.
Descrição: Dado o nome do nó contexto e o identificador do elo contido no nó contexto, esta rotina retira o elo da lista de elos do nó contexto.

int numero_de_elos(char *nome)
char *nome: nome que identifica o nó contexto na hiperbase.
Retorna: Retorna o número total de elos.
-1: Ocorreu erro.
Descrição: Dado o nome de um nó, esta rotina retorna o total de ancoras que ele contém.

int recupera_elo(char *nome, int id_elo, char *id_no_orig, int *id_anc_orig, char *id_no_dest, int *id_anc_dest)
char *nome: nome que identifica o nó contexto na hiperbase e o qual contém o elo.
int id_elo: identificador do elo no contexto;
Retorna: **char *id_no_orig:** lista de identificadores dos nós aninhados na origem.
int *id_anc_orig: identificador interno da âncora origem.
char *id_no_dest: lista de identificadores dos nós aninhados no destino.
int *id_anc_dest: identificador interno da âncora destino.
1: Não existe nó contexto com esse nome na hiperbase.
2: O nó não é um nó do tipo contexto.
3: Não existe elo com este identificador.
0: Ok..
Descrição: Dado o nome do nó contexto e o identificador do elo contido no nó contexto, esta rotina retorna a lista dos nós aninhados origem e destino e as âncoras origem e destino.

6. Conclusões e Trabalhos Futuros

Foi descrito neste trabalho o primeiro protótipo do módulo HIP/Windows, parte do ambiente SASHE. Trata-se de uma ferramenta de autoria de hiperdocumentos que seguem o modelo MCA. Nesta implementação, constrói-se bases privadas, ou seja, hiperdocumentos passíveis de modificação por qualquer usuário. Encontra-se em fase de implementação a ferramenta de apresentação, ou seja, a ferramenta através da qual um usuário pode navegar pelo hiperdocumento criado.

O primeiro módulo da primeira versão do sistema SASHE foi implementado utilizando-se o paradigma de orientação a objeto. Este paradigma permitiu que o sistema fosse divido em três módulos básicos e bem modulares - interface, biblioteca de classes e gerenciador da hiperbase.

O protótipo implementado utilizou as linguagens Pascal orientada a objeto na implementação da interface, C++ na implementação da biblioteca de classes e C na implementação do gerenciador da hiperbase. As plataformas utilizadas foram computadores com processadores 486 DX2 e Pentium rodando sobre o Windows 3.1 e Windows 95.

A escolha da linguagem e da plataforma de implementação do protótipo foi realizada tendo em vista possibilitar futuras extensões do modelo e o desenvolvimento de aplicações sobre ele.

Como objetivos da fase subsequente do projeto, destacam-se a implementação de mecanismos de gerenciamento da hiperbase pública, bem como do controle de versões em bases privadas. Além disso, projeto paralelo encarrega-se de definir a arquitetura dos demais módulos do ambiente SASHE referentes aos objetivos da aplicação voltada ao ensino. Mais especificamente, devem ser definidas as operações sobre a hiperbase que estarão disponíveis aos usuários professor e aluno, e que dizem respeito, por exemplo, a tipos de organização de material selecionado do hiperdocumento construído pelo autor/especialista e funções de busca que visam esta organização, formas de navegação dos diferentes usuários, gerenciamento de informações sobre a navegação feita pelo aluno (modelo do aluno), entre outras funções.

7. Referências

[1] Batista, T.L. *Controle de Versões no Modelo Hipermídia de Contextos Aninhados.* Tese de Mestrado. Departamento de Informática, PUC-Rio, Mar 1994.

[2] Batista, T.L., Oliveira, J.C. *Aplicações Hipermídia Distribuídas: Correio Eletrônico, Vídeotelefonia e Vídeoconferência.* Anais do I Workshop em Sistemas Hipermidia Distribuídos, ICMSC-USP, São Carlos, Jul. 1995.

[3] Casanova, M.A.; Tucherman, L.; Lima, M.J.; Rangel Netto, J.L.; Rodrigues. N.R.; Soares, L.F.G. *The Nested Context Model for Hyperdocuments.* Proceedings of Hypertext'91. Texas, Dez. 1991.

[4] Deslisle, N., Schwartz, M. *Neptune, a hypertext system for CAD applications.* Proceedings of ACM SGMOD'86. Washington, D.C., Mai. 1986.

[5] Halasz, F.G., Schwartz, M. *The Dexter Hypertext Reference Model.* NIST Hypertext Standardization Workshop. Gaitherburg, Jan. 1990.

[6] Rizk, A., Sauter, L. *MultiCard: an open hypermedia system.* Proceedings of European Conference on Hypertext, ECHT'92, Milano, Dez. 1992.

[7] Schutt, H.A., Streitz, N.A. *HyperBase: a hypermedia engine based on a relational database management system.* Proceedings of European Conference on Hypertext. ECHT'90. 1990.

[8] Soares, L.F.G., Casanova, M.A., Colcher, S. *An architecture for hypermedia systems using MHEG standar objects interchange.* Proceedings of the Workshop on Hypermedia and Hypertext Standards. Amsterdam, The Netherlands, Abr. 1993.

[9] Soares, L.F.G.; Rodriguez, N.L.R.; Casanova, M.A. Modelo de Contextos Aninhados: Um Modelo Conceitual Hipermídia. *Revista Brasileira de Computação,* Jan/Jun 1994, Rio de Janeiro, v.7 n.2 p.35-48.

[10] Sousa, G.; Costa, F. *O Modelo de Apresentação do HyperProp.* Anais do I Workshop em Sistemas Hipermidia Distribuídos, ICMSC-USP, São Carlos, Jul. 1995.

[11] Wiil, U.K., Leggett, J.J. *Hyperform: using extensibility to develop dynamic, open and distributed hypertext systems.I Proceedings of European Conference on Hypertex, ECHT'92. Milano, Dez. 1992.*

[12] Eckel, B. *C++ - Guia do Usuário.*, Ed. Makron Books, 536p., 1991.

[13] MICROSOFT , *Introducing Visual C++.*, Manual do usuário, 436p. 1994.

[14] PETZOLD, C. *Programando para Windows 3.*, Ed. Makron Books, 1005p., 1992.

Agradecimentos

Os autores gostariam de agradecer a colaboração de Maria da Graça Campos Pimentel e Renata P. Mattos Fortes neste projeto. Este trabalho tem o apoio financeiro do CNPq, através do projeto Protem-CC-II, no. 680063/94-3, e de projeto pesquisa no. 301365-91-1.