5
Trabalhos Relacionados

Este capítulo compara vários trabalhos relacionados à autoria de grafos e documentos hipermídia com o editor proposto nesta dissertação. Com o objetivo de organizar as comparações, o capítulo está estruturado nas seguintes seções: a Seção 5.1 apresenta o sistema GraphViz (Gansner, 2003) para autoria e visualização de grafos. O editor XML (XML, 2000) Salix (Aoki & Seraphim, 2003) é analisado na Seção 5.2. A Seção 5.3 apresenta o sistema Kaomi (Jourdan et al., 1999) para autoria de documentos hipermídia. A Seção 5.4 discute as principais características do sistema GRiNS (Bulterman et al., 1998), ferramenta para autoria de apresentações multimídia baseada em documentos SMIL (SMIL, 2001). Por fim, a Seção 5.6 analisa as ferramenta Ariadne (Jühne et al., 1998) e o ambiente Arakne (Bouvin, 2000).

5.1.
GraphViz

GraphViz (Gansner, 2003) é um sistema para visualização de grafos desenvolvida pela *AT&T,* distribuído na comunidade através da licença "*AT&T License*". A linguagem de programação utilizada no desenvolvimento da ferramenta foi *C++ ANSI* juntamente com *TCL/Tk 8.3* (*Tool Command Language*) (TCL/TK, 2004), sendo assim, portável em outras plataformas como Unix, (Linux, Solaris, IRIX, AIX, BSD), Windows, Macintosh etc.

A ferramenta fornece três construtores (algoritmos) para desenhos de grafos. Basicamente, todos os construtores trabalham da mesma forma, tendo como entrada um arquivo textual na linguagem *dot* (Gansner et al., 2002) e como saída o desenho do grafo em um arquivo, podendo o usuário escolher o formato de saída (gif, png, jpeg, PostScript etc.) do mesmo.

O *parser* utilizado na ferramenta para converter documentos criados na linguagem *dot* em figuras encontra-se nas bibliotecas *libgraph* e *libagraph*. Caso alguma inconsistência seja encontrada durante o processamento do arquivo fonte (arquivo *dot*), uma mensagem de erro é enviada ao usuário informando o número da linha do arquivo onde o erro se encontra.

Existem algumas ferramentas disponíveis na Internet (GraphViz, 2002) para converter arquivos textuais de representação de grafos em arquivos na linguagem *dot*, tal como a ferramenta *GXL2DOT*, que recebe como entrada um arquivo na linguagem GXL - *Graph eXchange Language* (Winter et al., 2002) (Apêndice A). A Figura 5-1 ilustra a interface principal da ferramenta GraphViz.

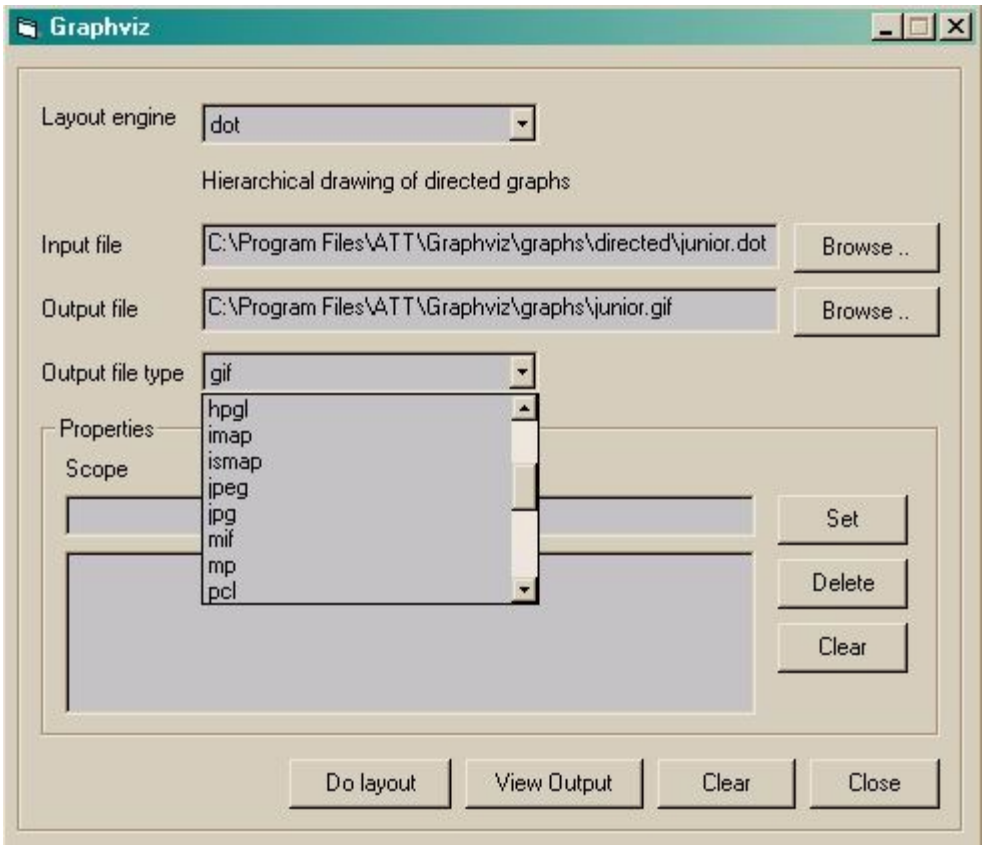


Figura 5-1 Sistema GraphViz

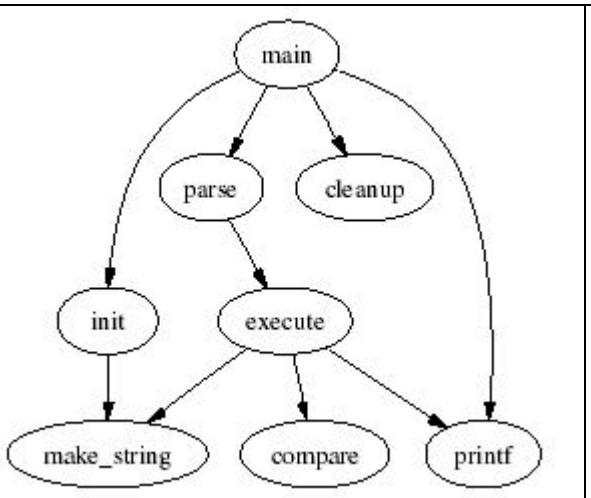
A utilização da ferramenta é simples. O usuário inicialmente escolhe qual construtor (*Layout engine*) será utilizado no desenho do grafo. Em seguida, o usuário deve informar o nome do arquivo de entrada (arquivo textual na linguagem *dot),* o nome do arquivo de saída e, por último, o formato do arquivo de saída (jpg, gif, jpeg etc).

Cada construtor (*dot, neato e towapi*) da ferramenta GraphViz está voltado para um determinado tipo de grafo. Conhecer o funcionamento desses construtores torna-se importante pois, de acordo com o tipo de grafo elaborado na linguagem *dot*, o usuário saberá escolher qual deles melhor se aplica à representação do seu grafo. Nas próximas subseções cada construtor será analisado juntamente com a linguagem *dot*.

5.1.1. Construtor *dot*

O construtor *dot* (Gansner et al., 2002) é destinado para desenhar grafos hierárquicos e direcionados. Os algoritmos usados no desenho são baseados nos trabalhos de (Warfield, 1977), (Carpano, 1980) e (Sugiyama et al., 1981) que operam com a seguinte estratégia: os primeiros vértices definidos no arquivo fonte (linguagem *dot*) apresentam maior prioridade hierárquica com relação aos demais, ou seja, os primeiros elementos definidos no arquivo serão desenhados acima dos demais. Observe na parte direita da Figura 5-2 (fonte na linguagem *dot*) que o elemento *main* foi o primeiro vértice a ser definido no arquivo e conseqüentemente o primeiro a ser desenhado na figura do grafo, criando assim uma hierarquia no desenho conforme os elementos são definidos no arquivo fonte da linguagem *dot*.

O sinal “->” presente no arquivo fonte (linguagem *dot*), indica o sentido da aresta entre dois vértices. A Figura 5-2 ilustra um grafo hierárquico desenhado pelo construtor *dot*. Ao lado da figura, encontra-se a especificação do grafo na linguagem *dot*.




```
digraph G {
        main -> parse -> execute;
        main -> cleanup;
        main -> init;
        execute -> make_string;
        execute -> printf
        execute -> compare;
        init -> make_string;
        main -> printf;
}
```

Figura 5-2 Grafo Hierárquico (*dot*)

Uma deficiência encontrada no construtor *dot*, em alguns casos, é a sobreposição de arestas e vértices quando o grafo desenhado apresenta uma grande quantidade (número superior a 50) de vértices e arestas.

5.1.2. Construtor *neato*

O construtor *neato* é destinado para desenhar grafos não orientados, comuns na representação de redes de computadores, modelos físicos que identificam pontos de baixas energias, sistemas de software etc. (North, 2002).

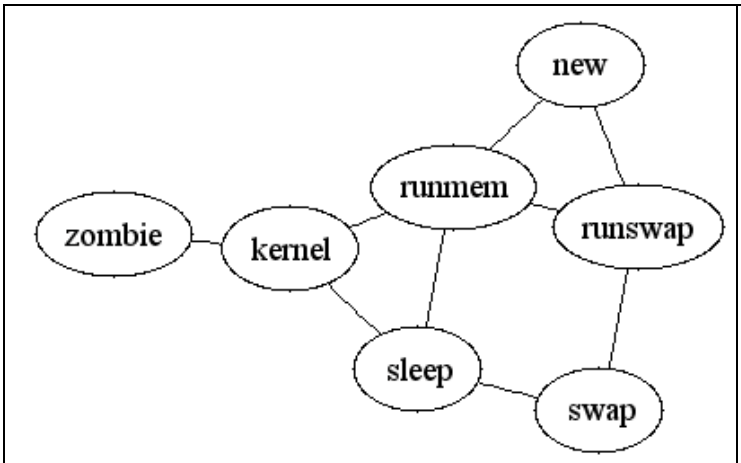
Os algoritmos utilizados para o desenho dos grafos no construtor *neato* foram propostos por (Kamada & Kawai, 1989). O conceito de elasticidade (*Spring*) é aplicado entre todos os pares de vértices de forma que o tamanho do grafo desenhado seja o menor caminho entre os pontos terminais do mesmo. Maiores detalhes do algoritmo do algoritmo de *Spring* podem ser encontrados em (North, 2002) e (Pinto, 2000).

Com o construtor *neato* é possível configurar tamanhos de arestas em relação a outras arestas, por exemplo, o usuário pode especificar que a aresta entre os vértices *v1* e *v2* tenha o dobro do tamanho da aresta que une os vértices *v1* e *v3*.

Tanto o construtor *neato* como o *dot* oferecem as mesmas possibilidades de configurações de desenho de grafos, tais como: cor do vértice, formato do vértice

(quadrado, elipse, triângulo etc.) e fontes dos textos para legenda de vértices e arestas. Além disso, o usuário pode determinar a posição exata de vértices no desenho do grafo com base nas coordenadas cartesianas da figura a ser criada. Nesse caso, o ponto de origem (0,0) do eixo cartesiano está localizado na parte superior esquerda da figura.

A Figura 5-3 apresenta um grafo não direcionado desenhado pelo construtor *neato*, tendo ao seu lado a especificação do grafo na linguagem *dot*. Observe que todas as arestas são representadas pelo símbolo "--", utilizado na linguagem *dot* para representar uma aresta não orientada.




```
graph G {
kernel -- zombie;
kernel -- sleep;
kernel -- runmem;
sleep -- swap;
swap -- runswap;
runswap -- new;
runswap -- runmem;
new -- runmem;
sleep -- runmem;
}
```

Figura 5-3 Grafo não direcionado (*neato*)

Análogo ao construtor *dot*, uma deficiência encontrada no construtor *neato*, em alguns casos, é a sobreposição de nós em arestas quando o grafo desenhado apresenta uma grande quantidade (número superior a 50) de vértices e arestas.

5.1.3. Construtor *twopi*

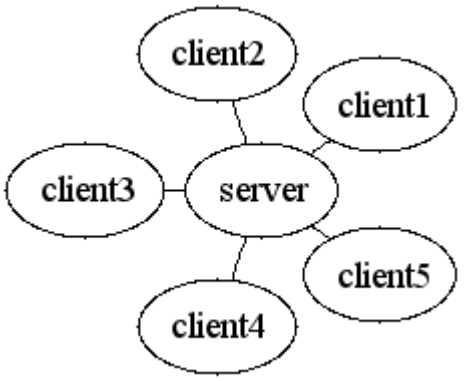
Destinado a desenho de grafos circulares (Wills, 1997), o algoritmo desse construtor procede da seguinte forma: um vértice é escolhido como centro da circunferência e todos os outros vértices são dispostos em volta dele. Caso a quantidade de vértices desenhados em volta do vértice central gere sobreposições de vértices e arestas, outros círculos podem ser criados com raios diferentes até que todos os vértices sejam desenhados.

Na linguagem *dot*, o usuário pode determinar a distância entre cada anel circular, a forma das figuras dos vértices (quadrado, triângulo, imagens etc.), cores de preenchimento, fonte de texto etc.

É importante destacar que a linguagem *dot* disponibiliza recursos para todos os construtores. No entanto, caso alguma informação seja passada no fonte do arquivo (linguagem *dot)* que o construtor não consiga interpretar, será enviada uma mensagem de erro para o usuário informando a linha na qual o erro foi detectado. Por exemplo, inserir a aresta "->" no fonte do arquivo para representar uma aresta no construtor *twopi*.

O construtor é pré-configurado para trabalhar apenas com um anel, podendo gerar sobreposição de vértices caso o número de vértices a serem desenhados seja grande (número maior que 30).

A Figura 5-4 ilustra um exemplo simples de grafo desenhado com o construtor *twopi*.




```
graph G {
    server -- client1;
    server -- client2;
    server -- client3;
    server -- client4;
    server -- client5;
}
```

Figura 5-4 Grafo circular simples (*twopi*)

A Figura 5-5 ilustra um grafo com um número maior de vértices que o da Figura 5-4. Devido à utilização de apenas um anel, houve sobreposição.

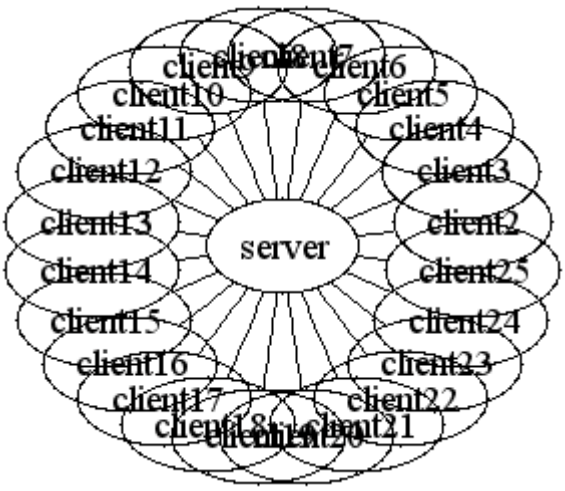


Figura 5-5 Grafo circular com sobreposição (*twopi*)

```
Graph G {
        server -- client2;
        server -- client3;
        server -- client4;
        server -- client5;
        server -- client6;
        …
        server -- client22;
        server -- client23;
        server -- client24;
        server -- client25;
}
```

A Figura 5-6 apresenta o mesmo grafo da Figura 5-5, porém sem sobreposição de vértices, através da utilização de dois anéis. Observe, no código, as linhas em negrito "**ranksep = 1.75"** e **"overlap = false"** que foram adicionadas ao código fonte do arquivo em relação ao código da Figura 5-5. A variável *ranksep* determina a relação de distância entre os raios circulares a serem desenhados e a variável *overlap* com o valor igual a *"false"* impede a sobreposição de vértices. No caso da Figura 5-6 o valor de *ranksep* igual a 1.75 significa que o raio do próximo anel será 75% maior que o inicial.

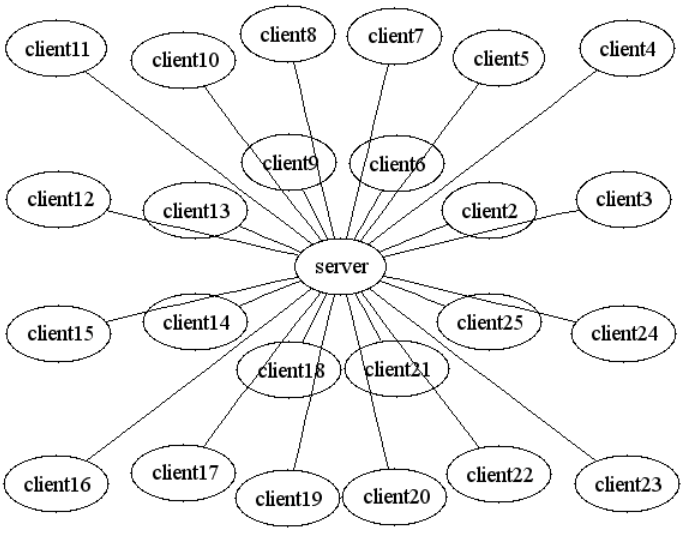


Figura 5-6 Grafo circular sem sobreposição (*twopi*)

```
graph G {
        ranksep = 1.75;
        overlap = false;
        server -- client2;
        server -- client3;
        server -- client4;
        …
        server -- client22;
        server -- client23;
        server -- client24;
        server -- client25;
}
```

5.1.4. Comparações entre o GraphViz e HyperProp

A visão estrutural atual do sistema HyperProp (Soares et al., 2003) apresenta apenas o algoritmo *Spring* (Kamada & Kawai, 1989) disponível para representação de grafos. Já a ferramenta GraphViz, além do algoritmo de *Spring*, disponibiliza ao usuário outros dois algoritmos, *Hierárquico* (Sugiyama et al., 1981) e *Circular* (Wills, 1997).

Na nova versão da visão estrutural do HyperProp (Seção 3.2.1), que está em desenvolvimento, o usuário pode escolher o algoritmo aplicado no desenho da visão estrutural. Atualmente, já estão implementadas as opções *Spring* (Kamada & Kawai, 1989), *Hierárquico* (Sugiyama et al., 1981) e *Circular* (Wills, 1997).

Uma deficiência encontrada na ferramenta GraphViz é não apresentar uma interface gráfica que permita uma interação com o usuário, para construção de grafos passo a passo. O uso da ferramenta limita-se a desenhar grafos descritos na linguagem *dot*. Já o HyperProp possibilita ao usuário o uso de interfaces gráficas na criação de vértices e arestas, oferecendo técnicas de filtragens para visualização do grafo.

O problema da sobreposição de vértices e arestas encontradas na ferramenta GraphViz é melhor solucionado no HyperProp, embora também ocorra. No HyperProp esse problema é parcialmente contornado, pois o desenho do grafo é mais flexível no que tange ao dimensionamento da figura. No entanto, para grafos grandes, o dimensionamento pode, às vezes, prejudicar o entendimento do desenho por parte do usuário. As soluções empregadas para resolver esse problema foram o uso de *scroll* e as técnicas de filtragem (Capítulo 4).

5.2. SALIX

SALIX (Aoki & Seraphim, 2003) é um ambiente para autoria de documentos XML (XML, 2000), desenvolvido na Universidade Estadual Paulista, que tem como característica principal o uso de uma interface gráfica (WYSIWYG - *What You See Is What You Get*) com recursos que agilizam a produção de documentos XML.

A ferramenta SALIX trabalha com gerenciamento de janelas MDI (*Multiple Document Interface*), permitindo que vários documentos sejam abertos ao mesmo tempo. As seguintes janelas estão disponíveis no editor:

- XML (texto) – para a edição do documento XML em modo texto.
- DTD (texto) – para edição textual da DTD - *Document Type Declaration* referenciada pelo fonte XML;
- XML (gráfico) – para edição do documento XML em modo gráfico.

Assim, duas formas de visualização de documentos XML são apresentadas. A primeira é puramente textual, como em um editor de texto convencional. Porém, por ser um editor de uma linguagem de marcação, as marcas (*tags*) dos elementos são diferenciadas (apresentadas em cores diferentes) do conteúdo do texto propriamente dito. Esse recurso é usado nas guias DTD e XML (texto), conforme ilustrado na Figura 5-7.

A segunda forma de visualizar o documento XML é através da interface gráfica criada com uma tabela de valores para cada elemento XML presente no documento e uma visão em árvore do documento XML. A Figura 5-8 apresenta o editor na forma gráfica.

O quadro de estrutura hierárquica, lado esquerdo do editor, apresenta os nomes dos elementos, nomes de atributos de cada elemento do documento XML, formando, assim, a árvore do documento. Nesse quadro, o usuário pode adicionar e excluir elementos com auxílio do mouse, além de expandir e fechar uma folha na árvore.

O quadro grade de valores, lado direito do editor, apresenta o nome do elemento e o valor de cada atributo do elemento selecionado pelo usuário na árvore esquerda do editor. Nessa visão, o usuário pode alterar o conteúdo dos atributos, assim como o nome do elemento.

É importante destacar que a visão gráfica só está disponível para o usuário se o arquivo estiver bem formado, ou seja, respeitando as regras de formação definidas pela DTD.

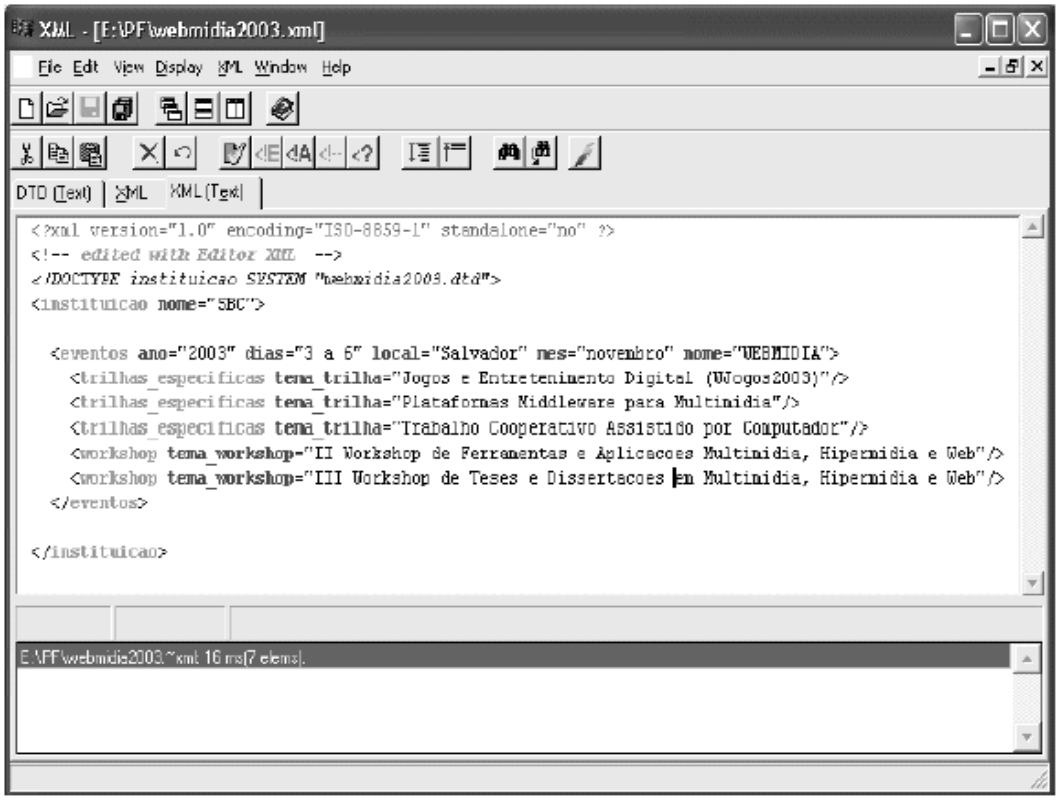


Figura 5-7 Ambiente SALIX (Autoria Declarativa)

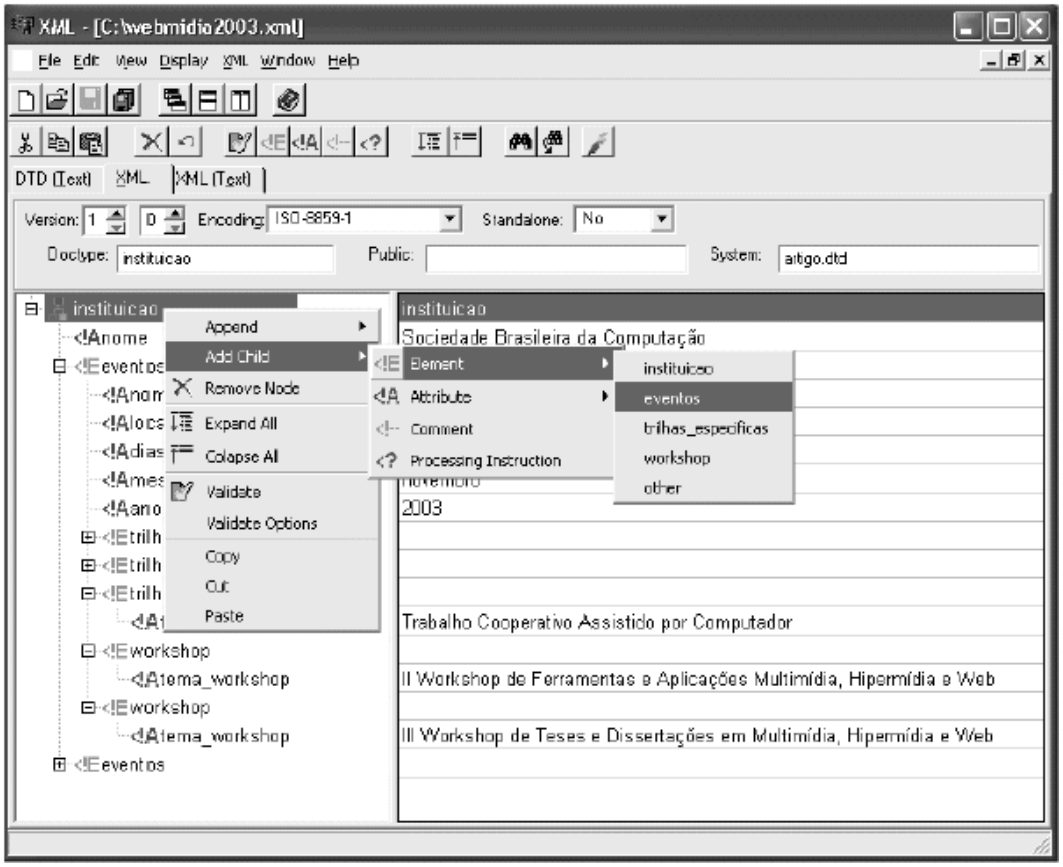


Figura 5-8 Ambiente SALIX (Autoria em árvore e tabelas)

O editor SALIX apresenta as seguintes funcionalidades, além da própria edição de documentos:

1. *Validação do documento* – Permite validar o documento XML editado pelo usuário caso exista alguma DTD associada ao arquivo;
2. *Depurador* – Permite ao usuário localizar os erros (caso existam) no documento editado depois da validação.

5.2.1. Comparações entre o SALIX e HyperProp

A visão declarativa do sistema HyperProp e o sistema SALIX são muito semelhantes. As funcionalidades de validação e depuração presentes no ambiente SALIX também estão disponíveis na visão declarativa do HyperProp, assim como a presença do *highlight* nos nomes e atributos de cada elemento do documento na forma textual.

As principais diferenças entre os editores são:

1. Visão em árvore do documento XML. A visão em árvore do editor declarativo do sistema HyperProp (parte esquerda do editor, Figura 3-1) informa apenas o nome dos elementos XML do documento. Já na árvore do editor SALIX, para cada elemento XML presente no documento, uma nova sub-árvore é criada, tendo como folhas todos os atributos do elemento. Note que, em alguns casos, o excesso de folhas criadas pode acabar dificultando o usuário na busca por uma determinada informação no documento.
2. Destaque para elementos selecionados na árvore do documento. No sistema SALIX, quando o usuário seleciona um determinado elemento na árvore XML do documento, a parte direita do editor é automaticamente atualizada mostrando apenas as informações do elemento selecionado. Já na visão declarativa do sistema HyperProp, quando o usuário seleciona um determinado elemento da árvore XML, a linha textual referente ao elemento selecionado é destacada em azul e o cursor do editor posicionado no início da linha.
3. Técnicas de Filtragens. O sistema SALIX não apresenta nenhuma técnica de filtragem associada ao editor. A visão declarativa do HyperProp apresenta a técnica de filtragem olho-de-peixe conforme apresentado no Capítulo 4.

5.3. Kaomi

O sistema Kaomi (Jourdan et al., 1999) tem, como base do processo de autoria, um conjunto de visões do documento, onde a sincronização entre as visões é realizada através da seleção de objetos ou por intervalos de tempo predefinidos, cabendo ao usuário do sistema configurar o valor desses intervalos.

É interessante destacar que o Kaomi é flexível pois trabalha com uma grande variedade de formatos declarativos de documentos multimídia, tais como documentos SMIL, XML, formato declarativo do modelo Madeus e etc.

O Kaomi fornece diferentes visões para trabalhar sobre o documento multimídia, proporcionando ao usuário facilidades na navegação e edição de documentos multimídia

baseadas no paradigma WYSIWYG (*What You See Is What You Get*). As visões oferecidas pela ferramenta são:

1. *Visão de Apresentação* – oferece funções de controle básicas (tocar, parar, pausar) para determinados tipos de mídia.
2. *Visão Estrutural* – apresenta o conjunto de objetos que pertencem ao documento. A estrutura formada é uma árvore onde as folhas representam os objetos básicos de mídia e os nós não terminais são os objetos compostos. É interessante destacar que, nesse tipo de visão, é possível aplicar filtros de exibição da estrutura, tais como: ordem alfabética e tipo de mídia a ser exibida.
3. *Visão Declarativa* – exibe o documento fonte na forma textual em um editor semelhante ao programa *Notepad*, com recursos básicos de edição;
4. *Visão Temporal* – projeta, em um grafo, a execução de um documento no tempo, preservando a noção de sua estruturação hierárquica. Nessa visão, as arestas representam elos e, os vértices, os objetos de mídias.

Quando ocorre alguma alteração durante a edição no documento em uma determinada visão, essa alteração é propagada para todas as outras visões, garantindo com isso a consistência dos dados.

O sistema Kaomi também disponibiliza as funções elementares para a autoria de documentos tais como: abrir, fechar, criar e salvar. Além dessas, o Kaomi apresenta funções temporais (por exemplo, exibir em série objetos de mídia ou apresentar os objetos em paralelo, inserir atraso entre os objetos etc.) e funções de relação hipertexto, por exemplo, operações do tipo *goto*, ou seja, caso algum objeto de mídia seja selecionado pelo usuário durante a apresentação do documento um outro objeto de mídia pode ser acionado.

O ambiente de autoria produzido pelo Kaomi pode ser ajustado para se adequar ao contexto da aplicação. A ferramenta oferece meios de ser expandida, permitindo que novas visões sejam adicionadas, mantendo as características de edição e sincronismo. É possível acrescentar ainda novas funções em uma visão, assim como definir novas relações temporais e ampliar o suporte a novos tipos de formato de dados.

5.3.1. Comparações entre o Kaomi e HyperProp

O sistema HyperProp apresenta observadores que atuam sobre as visões (Seção 3.4), responsáveis pelo controle da sincronização entre elas. Já o sistema Kaomi possibilita ao usuário determinar intervalos de tempo para a sincronização, além dos eventos de sincronização pré-configurados no ambiente, tais como: seleção de objetos, exclusão de elementos, inserção de elementos etc.

O Kaomi oferece a possibilidade de trabalhar com novos formatos de arquivos diferentes daquele para o qual a ferramenta vem pré-configurada (SMIL, 2001). Para isso, o usuário deve criar compiladores para converter seus documentos na base de dados utilizada pelo sistema (Jourdan et al., 1999). O HyperProp também pode ser usado por diferentes formatos de arquivos, desde de que o usuário crie compiladores que convertam o arquivo do usuário para a base de dados (objetos Java representando um grafo) utilizada pelo sistema HyperProp.

A visão declarativa do Kaomi é muito simples, apresentando apenas os elementos na forma puramente textual sem recursos de destaque (*highlight*), ausência da visão em árvore do documento interna ao editor etc. A visão declarativa do sistema HyperProp disponibiliza validação do documento baseado no *Schema* do arquivo XML, técnicas de filtragem, recursos de *highlight*, visão em árvore do documento interna ao editor declarativo etc.

A visão estrutural do Kaomi limita-se a apresentar apenas a estrutura do documento na forma de árvore. Além disso, a visão estrutural do Kaomi não disponibiliza qualquer técnica de filtragem nas visões e nada é mencionado com relação aos algoritmos de desenho grafos. Já a visão estrutural do HyperProp mostra a árvore do documento na parte esquerda do editor (Figura 3-3), o desenho do grafo com todos os relacionamentos entre os vértices na parte direita da visão (Figura 3-3) e um conjunto de funcionalidades para edição de documentos já mencionados anteriormente na Subseção 3.2.1.

5.4. GRiNS

GriNS (Bulterman et al, 1998) é um ambiente para autoria de documentos hipermídia com suporte nativo à linguagem SMIL (SMIL, 2001), padrão W3C, que tem como característica principal o uso da visão temporal no processo de autoria com recursos gráficos que facilitam a complexa tarefa de organização temporal dos objetos de mídia no documento.

As apresentações (documentos SMIL) criadas no ambiente GRiNS podem ser visualizadas no próprio ambiente e em diferentes sistemas (*players*) que são compatíveis com a linguagem SMIL, tais como: RealOne, IE-6 e 3GPP.

O sistema trabalha com arquivos de áudio (mp3, wav e aiff etc.), vídeo (avi, mpeg etc.), imagem (bmp, jpg, gif, png etc.) e texto (HTML, XHTML etc.) de forma integrada dentro da mesma apresentação hipermídia.

A Figura 5-9 apresenta a tela inicial do ambiente GRiNS com suas barras de ferramentas (*general toolbar*, *container toolbar*, *region/alignment toolbar*, *linking/interation toolbar e previewer control panel*) e barra de menu (*menu bar*). As funcionalidades das principais barras serão explicadas nas próximas subseções, juntamente com as cinco visões disponíveis no ambiente.

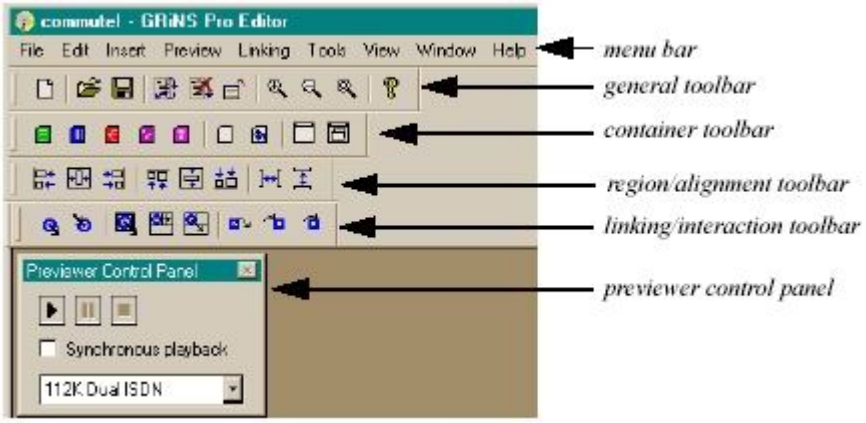


Figura 5-9 Janela principal do ambiente GRiNS

5.4.1.
Visão de Apresentação

Permite ao autor verificar a apresentação do seu documento hipermídia antes do mesmo ser disponibilizado para diferentes sistemas (RealOne, IE-6 ou 3GPP).

A apresentação hipermídia pode ser simulada de acordo com a largura de banda selecionada pelo autor. Conforme o usuário altera o valor da largura de banda, os tempos de pré-busca (*prefetching*) dos objetos de mídia são recalculados na visão temporal e refletidos durante a apresentação do documento. A Figura 5-10 ilustra a visão de apresentação do sistema GRiNS.

Na visão de apresentação (*"Previewer Control Panel"*) é possível que o autor escolha partes do documento a serem apresentadas, por exemplo, o ponto de início e fim do documento.

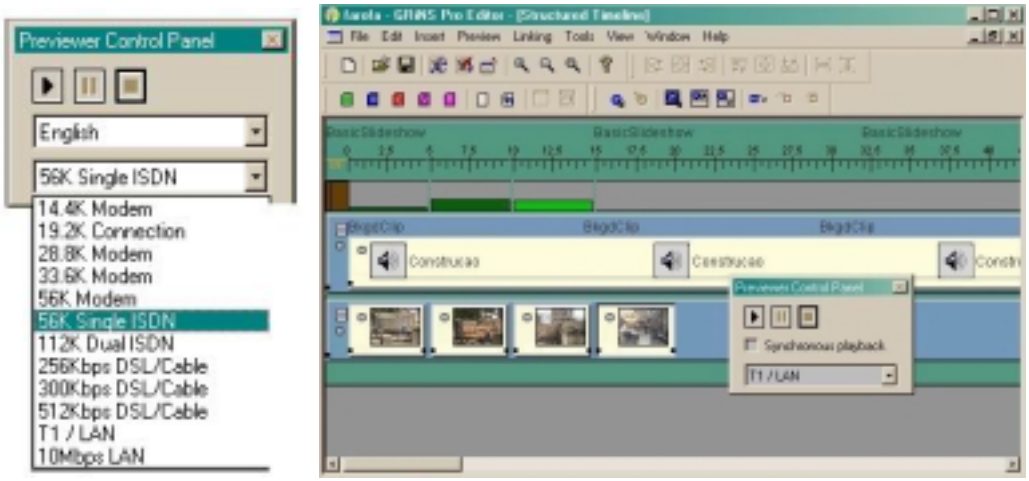


Figura 5-10 Visão de apresentação do sistema GRiNS

5.4.2.
Visão Temporal

A visão temporal do sistema GRiNS permite ao autor criar documentos hipermídia baseados nos elementos de semântica temporal definidos na linguagem SMIL-2.0.

Os principais elementos temporais presentes no ambiente GRiNS são: paralelo, seqüencial e escolha (*switch*). Cada elemento temporal é representado no sistema como um contêiner de forma que todos os elementos internos a esse contêiner estão sujeitos à sua semântica temporal.

A barra de menu *"Container toolbar"* apresenta um conjunto de botões coloridos, onde cada botão está associado a um tipo diferente de contêiner. O botão verde cria um contêiner com semântica temporal paralelo. O botão azul cria um *contêiner* com semântica temporal seqüencial e o botão vermelho cria um *contêiner* com semântica temporal escolha (dentre todos os elementos internos a ele apenas um será escolhido durante a apresentação do documento).

A Figura 5-11 mostra a visão temporal com a apresentação *"Sample Slideshow"*, *contêiner* azul mais externo. Dentro desse *contêiner* existe um único *contêiner* verde (*"Audio-and-Image"*) com outros três *contêiners:* o primeiro, de cor azul (*"Image-Sequence"*), contendo três objetos de mídia imagem. O segundo, apenas com um nó de texto e, o terceiro *contêiner*, de cor vermelha (*"Audio-Switch"*), com dois nós de áudio. Observe que os elementos estão organizados dentro dos *contêiners* de acordo com sua semântica temporal.

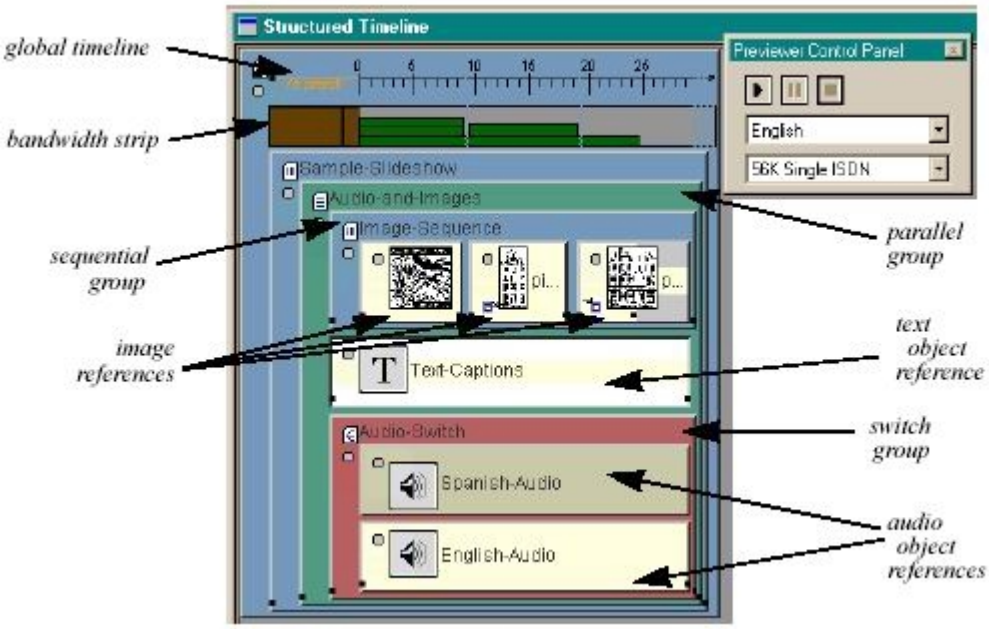


Figura 5-11 Visão Temporal do GRiNS

Na visão temporal o autor sabe quais são os tipos de objetos de mídia envolvidos na sua apresentação, os tipos de estruturas criadas na composição da apresentação e uma estimativa do tempo da apresentação do documento. Além disso, a visão temporal fornece recurso de integração com a plataforma *Windows*, onde arquivos selecionados no

sistema operacional *Windows* são inseridos no ambiente através de *drag-and-drop* (arraste e copie).

5.4.3. Visão Espacial

Na visão espacial, o autor pode criar e editar regiões de apresentações nas quais os objetos de mídias serão exibidos durante a execução do documento.

A Figura 5-12 ilustra a visão espacial do GRiNS. Na área esquerda do editor, as regiões criadas no documento são exibidas na forma de árvore e todos os objetos de mídias que estão associados a essa região aparecem como suas folhas. Na janela da direita, é ilustrado o objeto de mídia selecionado pelo autor na árvore (parte esquerda do editor) dentro da região que o objeto de mídia está associado.

Na visão espacial, o autor pode alterar os parâmetros (altura, largura etc.) da região com auxílio do *mouse*, assim como inserir novos objetos de mídia.

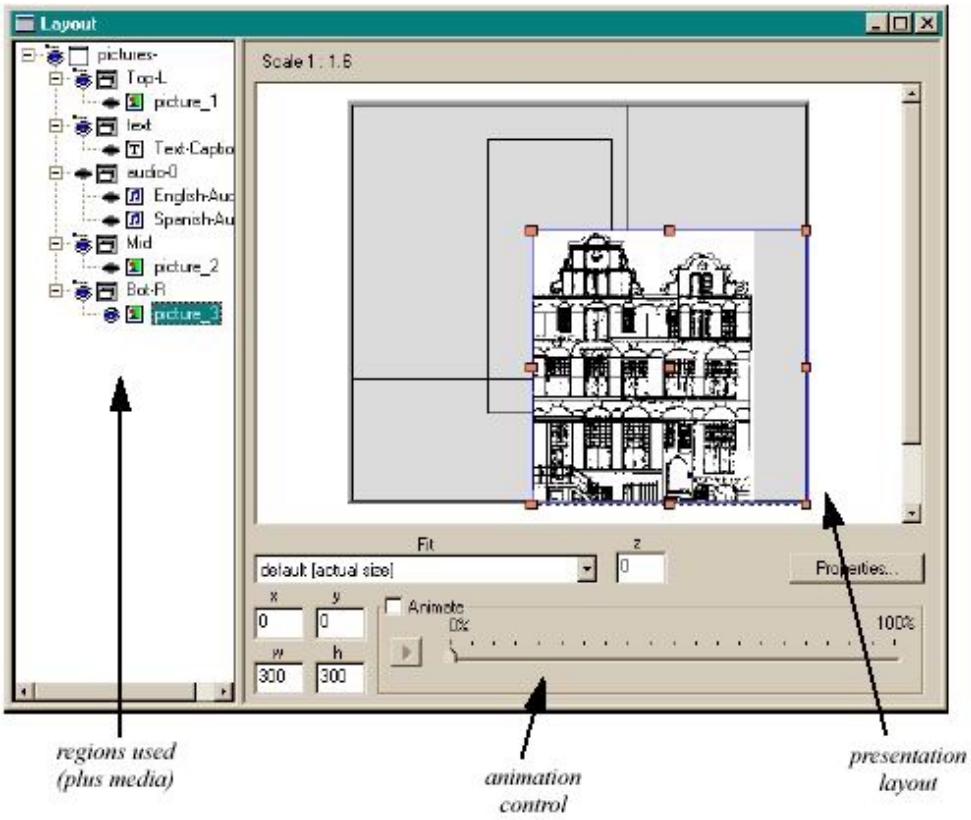


Figura 5-12 Visão Espacial do sistema GRiNS

5.4.4.
Visão Declarativa

A visão declarativa do sistema GRiNS apresenta o fonte do documento hipermídia na linguagem SMIL-2.0, permitindo ao autor editar o documento na forma textual. As alterações feitas na forma textual são atualizadas nas outras visões (temporal e espacial) desde que o documento não contenha erros.

Para atualizar as alterações realizadas na visão declarativa com as demais visões gráficas, o usuário precisar selecionar o botão "*apply*" no editor declarativo.

A Figura 5-13 exibe a visão declarativa do ambiente GRiNS. Note que o editor é simples, sem grandes recursos para edição textual.

Os únicos elementos que recebem destaque (*"highlight"*) no editor textual são aqueles de semântica temporal (*par*, *seq* e *switch*).

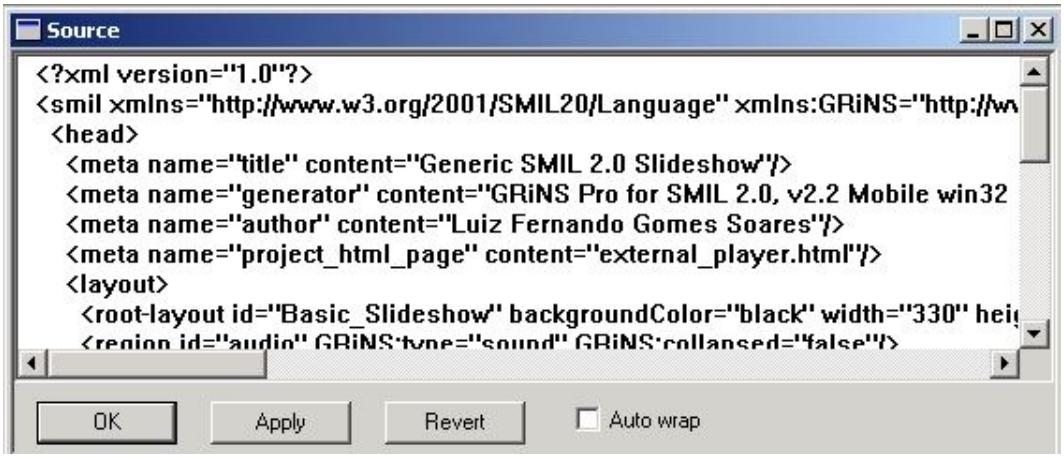


Figura 5-13 Visão Textual do GRiNS

5.4.5.
Comparações entre o Sistema GRiNS e o HyperProp

O ambiente GRiNS não apresenta a visão estrutural para autoria de documentos hipermídia. A principal visão de autoria do sistema GRiNS é a visão temporal. Já o sistema HyperProp além de disponibilizar a visão temporal oferece também a visão estrutural para edição de documentos.

A visão espacial do GRiNS apresenta algumas funcionalidades interessantes com relação à visão espacial atualmente desenvolvida no HyperProp, tais como: pré-

visualização do objeto de mídia na região (possibilitando ao autor alterar os parâmetros da região de acordo com a visualização do objeto de mídia na região) e visão em árvore dos elementos que compõem o *layout* da apresentação. Cada região espacial definida no documento forma um ramo da árvore tendo como folhas todos os objetos de mídia associados a essa região. Esse tipo de visão em árvore ainda não está implementada na visão espacial atual do HyperProp.

A visão declarativa do ambiente GRiNS apresenta sempre todo o documento na forma textual, tornando o documento de difícil entendimento quando o mesmo torna-se relativamente grande. A visão declarativa do HyperProp exibe apenas a parte textual do documento referente à visão gráfica que o usuário tem como foco. Por exemplo, quando o usuário ativa a visão gráfica espacial, apenas o texto referente à edição espacial é mostrado na visão declarativa.

Uma limitação do sistema GRiNS é o fato do mesmo ser executado apenas na plataforma *Windows*. Já o Hyperprop é multiplataforma por ser desenvolvido através da linguagem Java.

A opção de *"highlight"* no editor textual do GRiNS é simples, identificando apenas os nomes de elementos de semântica temporal (*par*, *seq* e *switch*). Já no editor declarativo do HyperProp, todos os nomes de elementos apresentam-se no formato negrito, o nome e o valor dos atributos são exibidos também em destaque. Por fim, o editor textual do GRiNS não exibe a visão em árvore do documento criado e nenhuma técnica de filtragem no editor é empregada.

5.5. Outras Ferramentas de Edição

As ferramentas apresentadas nas próximas subseções possuem funcionalidades que atualmente não estão disponíveis no sistema HyperProp. Entretanto, algumas delas, tal como a utilização de trilhas na navegação de documentos, já foram anteriormente implementadas no sistema (Muchaluat-Saade & Soares, 1995) (Muchaluat-Saade, 1996) e serão apresentadas como referências a outros trabalhos que permitem edição de documentos.

5.5.1. Ariadne

Ariadne é uma ferramenta que auxilia usuários da *WWW* (Berners-Lee et al., 1994a) a administrarem os caminhos (trilhas) percorridos durante sua navegação de páginas na *Web* (Jühne et al., 1998).

Com o sistema Ariadne, o usuário pode criar, editar e compartilhar trilhas sobre determinados assuntos na *Web*, assim como visualizar a trilha num formato gráfico.

Ao usar uma rota já criada no sistema Ariadne, o usuário passa a ter um mapa gráfico com os nós já percorridos, o ponto atual no qual ele se encontra e os nós ainda não percorridos pelo usuário.

É interessante destacar que, a qualquer momento, o usuário pode sair de uma rota pré-determinada e até mesmo adicionar novos nós na rota utilizada inicialmente. A Figura 5-14 ilustra o uso do mapa gráfico do sistema Ariadne.

Uma outra característica interessante do Ariadne é que ele não altera o conteúdo das páginas *WWW*, diferenciando-se das outras ferramentas com mesma funcionalidade, por exemplo: Walden's Path (Furuta et al., 1997).

O Ariadne torna-se muito útil quando o compartilhamento de rotas ocorre entre usuários. Por exemplo, ao se pesquisar um tema específico na *Web*, o usuário pode perder muito tempo filtrando informações relevantes à sua pesquisa, devido à grande quantidade de informações disponíveis na *Web* sobre o mesmo tema. Ao obter uma rota já criada, o usuário passa a ter um guia de navegação sobre seu tema, reduzindo com isso seu tempo de pesquisa.

As principais características do Ariadne são:

1. *Uso de interface independente* – O Ariadne mantém a apresentação das páginas *WWW* em seu formato original, ou seja, os conteúdos das páginas não são alterados;
2. *Ramificação de navegação* – Permite que o usuário navegue por rotas alternativas dentro de um guia de navegação;
3. *Composição de navegação* – Permite que uma navegação seja adicionada a outro guia de navegação;

4. *Orientação na navegação* – Provê uma representação gráfica da navegação do usuário e de seu histórico.

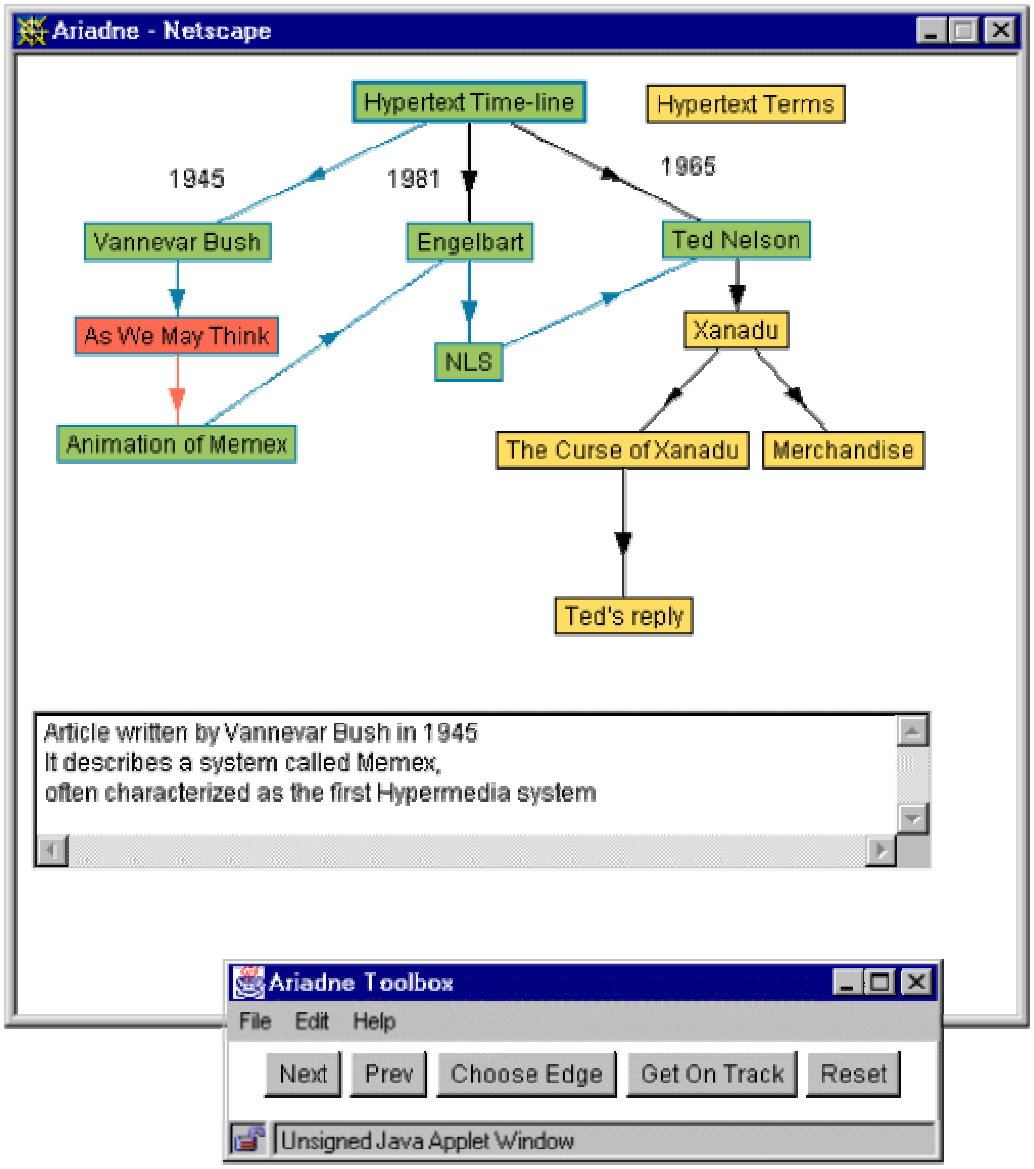


Figura 5-14 Mapa de navegação da ferramenta Ariadne

5.5.2. Arakne

No modelo atual da *WWW* é impossível criar relacionamentos (elos) ou anotações nas páginas da *Web* onde não se tem o direito de escrita. Isso dificulta os usuários a

organizarem as informações encontradas, haja vista que seria interessante adicionar comentários e novos elos em algumas páginas encontradas.

Uma alternativa para contornar esse problema foi através da criação de ferramentas que possibilitam o usuário modificar o conteúdo das páginas *WWW* localmente (na própria máquina). Assim, cada usuário pode "moldar" as páginas *WWW* pesquisadas da forma que melhor lhe convier.

Essas ferramentas permitem, entre outras facilidades, criar elos e adicionar comentários e conteúdo nas páginas. Geralmente, a ferramenta é instalada na máquina do cliente, mas pode também estar associada a um *proxy*, de maneira que as alterações feitas por um usuário possam ser vistas por todos os usuários que compartilham o mesmo *proxy*. Assim, a ferramenta facilita o trabalho colaborativo na busca por determinadas informações.

Uma dessas ferramentas é o Arakne (Bouvin, 2000), elaborada na Universidade de Aarhus. Inicialmente, a ferramenta foi desenvolvida como *applets*, possibilitando que usuários criassem comentários e elos nas páginas. Logo depois a ferramenta virou um aplicativo desenvolvido em Java com suporte a outros tipos de mídias como vídeo e áudio.

O usuário, ao assistir um vídeo através do sistema Arakne, pode adicionar elos em determinados intervalos de vídeo. A Figura 5-15 exibe um vídeo associado à ferramenta. Observe que o usuário criou o "*link 5"* com dois pontos de ativação (tempo de início e fim do elo). Caso o usuário (ou até mesmo outro usuário que assista o filme através da ferramenta Arakne) clique no vídeo durante o intervalo de tempo predeterminado pelo *"link 5"*, o elo será ativado, realizando um determinado evento, por exemplo, abrindo uma página HTML (HTML, 1999) com algumas informações sobre o ator principal da cena exibida nesse intervalo de tempo.

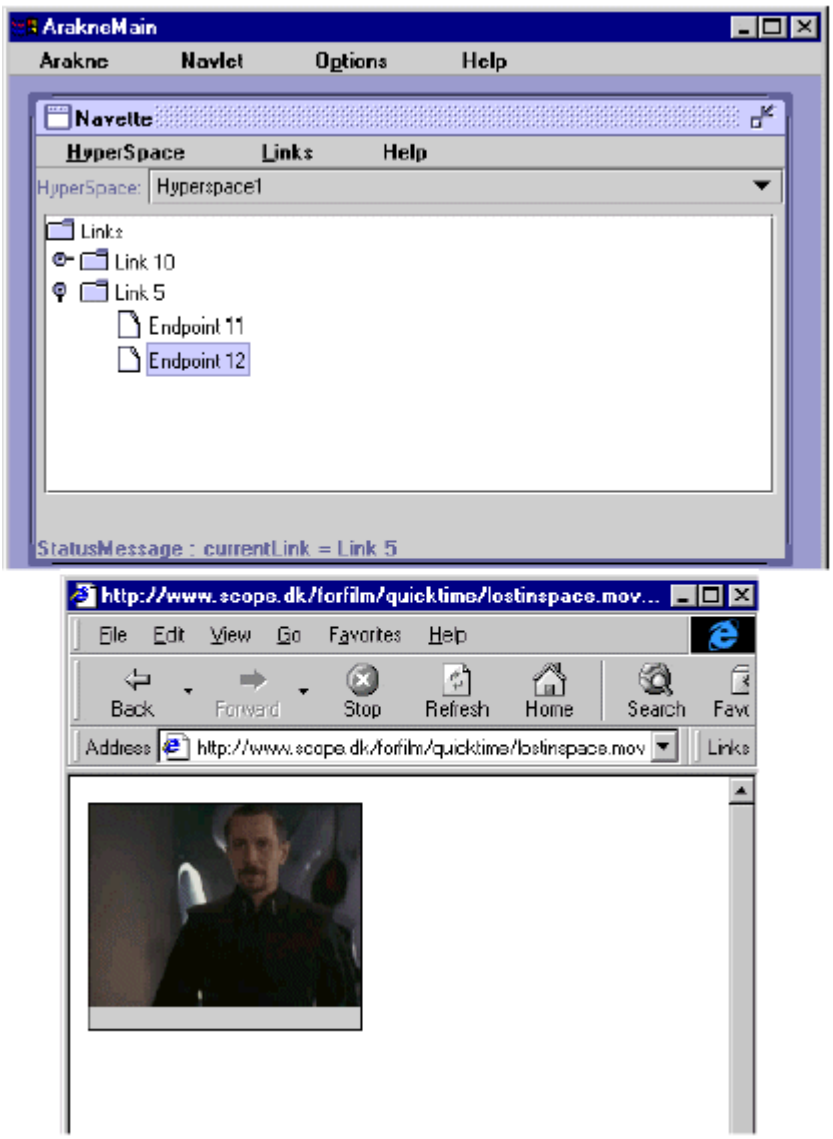


Figura 5-15 Ambiente Arakne