# 3
# Ferramentas para Edição de Arquiteturas de Sistemas Baseadas em Grafos Compostos

O sistema HyperProp (Soares et al., 2000) foi projetado inicialmente como um sistema para autoria e formatação de documentos hipermídia baseados no modelo NCM – *Nested Context Model* (Soares et al., 2003). Nos últimos anos, o sistema passou por diversas modificações, necessárias para incorporar novas características do modelo. Esta dissertação modificou o sistema de autoria, proporcionando a inclusão de uma nova visão, a visão textual. A visão espacial, apesar de ter sido definida anteriormente em (Costa, 1996) e (Moura, 2001), foi novamente implementada e incorporada à versão mais nova do sistema. Além disso, o ambiente de autoria foi remodelado para a edição de sistemas baseados em grafos compostos de uma forma geral (não apenas documentos hipermídia). Mais ainda, mecanismos de filtragem baseados na técnica olho-de-peixe foram implementados para as visões textual e espacial e atualizados para a visão estrutural.

Neste capítulo, o ambiente de autoria do sistema HyperProp será apresentado com suas quatro visões e, em seguida, o seu mecanismo de sincronização entre as visões será analisado.

## 3.1.
## Edição Textual

Na nova versão do sistema HyperProp foi acrescentada a visão declarativa ao ambiente de autoria, permitindo ao autor editar arquiteturas de sistemas na forma textual (arquivos baseados na meta-linguagem XML) e não apenas na forma gráfica, como anteriormente disponível (Pinto, 2000). A Figura 3-1 ilustra o editor textual integrado ao sistema exibindo um documento NCL.

A interface do editor textual é dividida em duas partes. A área esquerda apresenta a árvore XML (XML, 2000) do documento, na qual os vértices compostos podem ser

expandidos (exibindo todos os vértices diretamente pertencentes a ele) ou colapsados (ocultando todos os vértices diretamente pertencentes a ele). Já a área direita apresenta o documento na forma textual.

O texto apresentado no lado direito do editor pode ser exibido com o recurso de *highlight*, no qual o nome dos elementos ficam em negrito, enquanto os atributos do elemento e seus valores aparecem em cores diferentes. O uso do *highlight* é opcional, de acordo com a preferência do autor.

As alterações feitas em um dos lados do editor são refletidas no outro. Por exemplo, ao se criar um novo elemento na área textual (parte direita do editor), o mesmo será refletido como um novo nó na visão em árvore do documento (parte esquerda do editor). Além disso, ao clicar sobre um determinado nó da árvore na parte esquerda do editor, a linha textual referente ao nó é automaticamente destacada na área direita, como exemplificado com o elemento "*composite id=coisaPele*" na Figura 3-1.

Para sincronizar as alterações realizadas no lado textual com a árvore XML do documento o usuário pode pressionar a tecla "*F5*" do teclado ou clicar sobre o botão na barra de *menu* representado por duas setas.

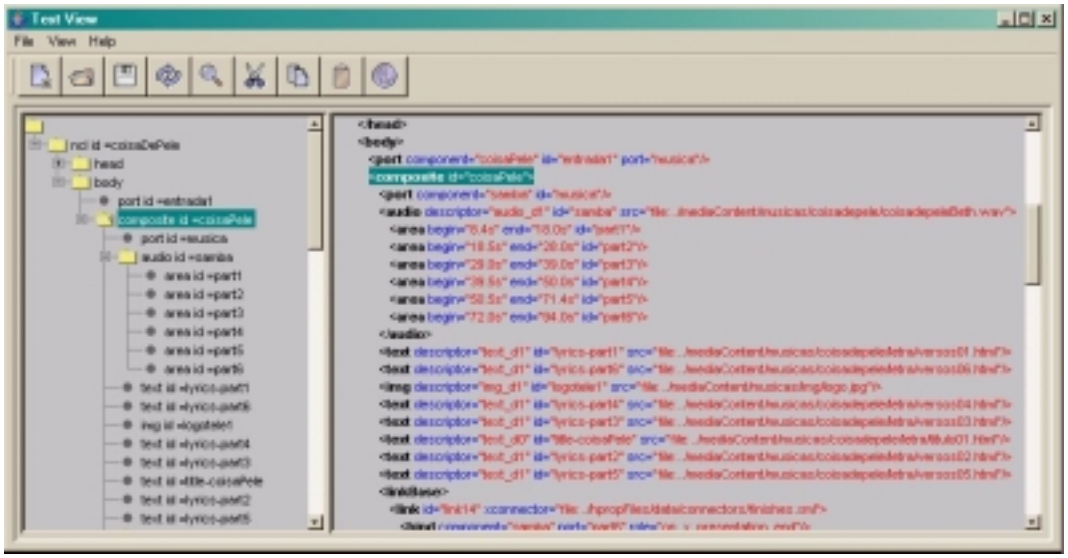


Figura 3-1 Visão declarativa do sistema HyperProp

É importante destacar que, no momento da sincronização da parte textual com a parte esquerda (visão em árvore), a especificação do documento textual é verificada de acordo com seu *Schema*, passado como parâmetro na declaração do documento e, caso nenhuma inconsistência seja detectada, a sincronização é realizada. Se, por acaso, uma

inconsistência for detectada, a sincronização é cancelada e uma mensagem de erro é enviada ao usuário.

Uma outra possibilidade de verificar se o documento XML editado pelo usuário está de acordo com a especificação do seu *Schema* é clicando sobre a opção de *menu View* e, em seguida, clicando sobre a opção de *menu Validate*. A Figura 3-2 ilustra um exemplo de validação, no qual o usuário definiu dois elementos *descriptor* com mesmo atributo *"id"*, obtendo uma mensagem de erro identificando o tipo de erro encontrado e a linha na qual o erro se encontra. Caso nenhum erro seja encontrado na especificação, uma mensagem também é exibida ao usuário informando que o documento está de acordo com sua especificação.

Se o usuário solicitar uma validação de documento e este não apresentar a localização do seu *Schema*, uma mensagem de erro será emitida informando que a validação não poderá ser realizada pois o documento não possuiu um *Schema* definido.

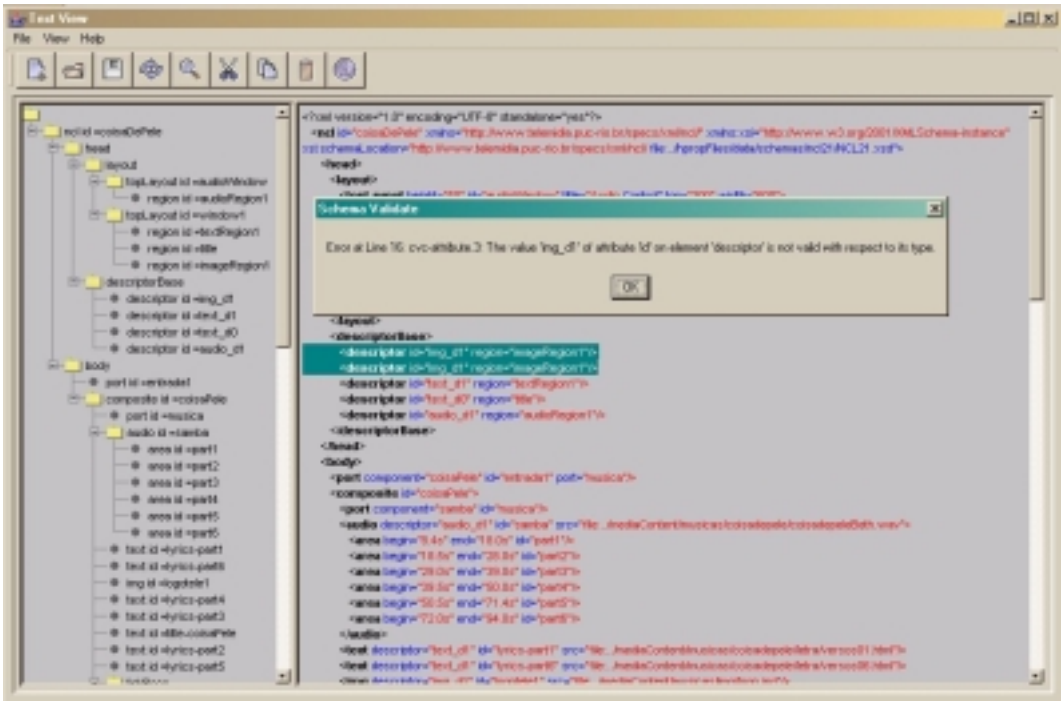


Figura 3-2 Validação de documentos XML

A validação do documento XML foi implementada usando o pacote JAXP (*Java API for XML Processing*) (JAXP, 2003) através da instanciação de um objeto da classe *DOMParser* que disponibiliza o método *parse* para validar documentos XML. O *Schema*

do documento a ser validado deve ser indicado dentro do próprio documento XML, no elemento raiz da árvore através do atributo *"xsi:schemaLocation"*.

Para que o método *parse* seja executado, uma *String* deve ser passada como parâmetro do método, indicando a localização do arquivo a ser validado. Neste momento, a parte textual do editor declarativo é salva em um arquivo temporário e sua localização passada como referência para o método *parse*. As informações de erros encontradas pelo método são tratadas e enviadas para o usuário.

Opções básicas como: copiar, colar e recortar, presentes nos editores tradicionais, também estão disponíveis na ferramenta. Caso algum elemento XML apresente o atributo *"id"*, este será refletido na árvore esquerda do editor. Observa-se na Figura 3-2 que os elementos *"head"*, *"layout"* e "body" da visão em árvore do editor não possuem o atributo *"id"*.

O editor declarativo do sistema HyperProp foi projetado para editar qualquer arquivo no formato XML e não apenas documentos NCL. Como exemplo da generalidade da ferramenta, a Figura 3-3 ilustra um documento GXL - *Graph eXchange Language*, uma linguagem declarativa para edição de grafos compostos (Winter et al., 2002) (Apêndice A), no qual o usuário está editando seu documento.

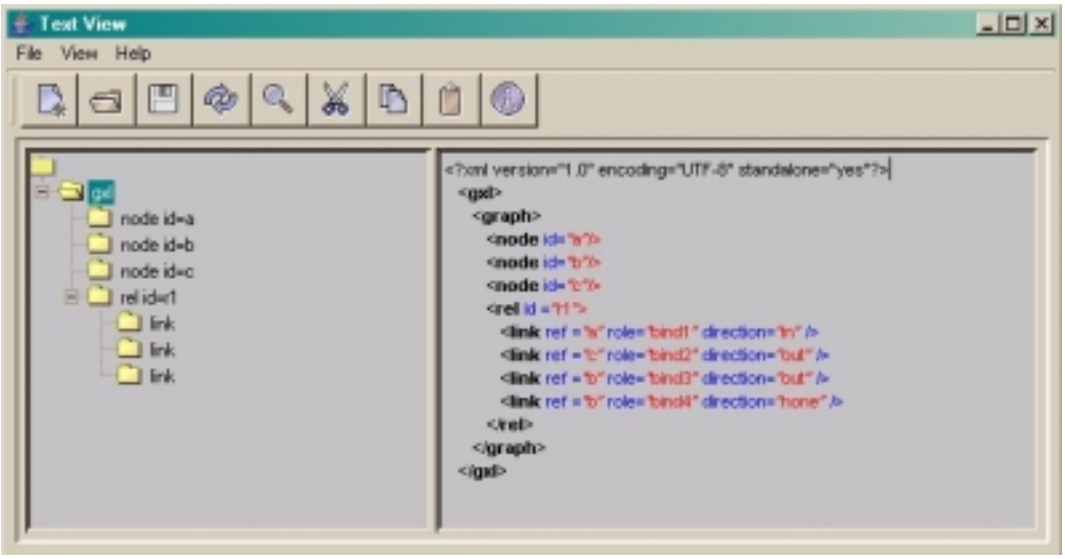


Figura 3-3 Editor declarativo com documento GXL

## 3.2. Edição Gráfica

Conforme comentado no Capítulo 1, uma outra alternativa para a autoria de arquiteturas de sistemas baseadas em grafos compostos é o uso de ferramentas gráficas. Cada uma das três ferramentas de edição gráfica disponíveis no sistema HyperProp será analisada separadamente nas próximas subseções.

### 3.2.1. Visão Gráfica Estrutural

A visão estrutural do ambiente de autoria do sistema HyperProp permite ao autor criar a estrutura lógica do grafo composto, ou seja, nela o autor pode criar, editar e apagar vértices (atômicos ou compostos) e arestas.

A interface da visão estrutural está dividida em duas partes, conforme ilustrado na Figura 3-4. O lado esquerdo da visão estrutural apresenta uma árvore com os vértices do grafo composto. O usuário pode escolher os tipos de vértices (atômico, composto ou ambos) a serem visualizados na árvore. No exemplo da Figura 3-4, apenas os vértices compostos são mostrados na visão em árvore do editor. Os vértices compostos podem ser expandidos, mostrando todos os vértices diretamente nele contido, ou colapsados, ocultando todos os vértices diretamente ligados a ele.

O lado direito da visão estrutural mostra todos os vértices (compostos e atômicos) contidos em um vértice composto selecionado na árvore (no exemplo da Figura 3-4, o vértice *coisaPele*). Além disso, o lado direito exibe as arestas que relacionam esses vértices.

Quando o vértice composto selecionado na árvore contém outros vértices compostos, esses últimos podem aparecer expandidos (por exemplo *MusicaeLetra)* ou colapsados (por exemplo, *Cantor*) na área da direita. Quando expandidos, os vértices internos e suas arestas são também exibidas. O processo de expandir vértices compostos pode se repetir nos vários níveis de aninhamento existentes na arquitetura representada.

Sempre que dois vértices são unidos por mais de uma aresta, a ferramenta exibe uma única aresta e coloca um rótulo identificando a quantidade de arestas entre estes vértices. Por exemplo, na Figura 3-4 pode-se observar a existência de uma aresta ligando

os vértices *samba* e *lyrics-part6* com o símbolo “(2)”. Neste caso, o símbolo “(2)” indica que existem duas arestas entre os vértices, apesar de apenas uma aresta ter sido desenhada. Já na aresta que liga os vértices *samba* e *photo*, não existe nenhum símbolo sobre a mesma, logo apenas uma aresta relaciona os vértices.

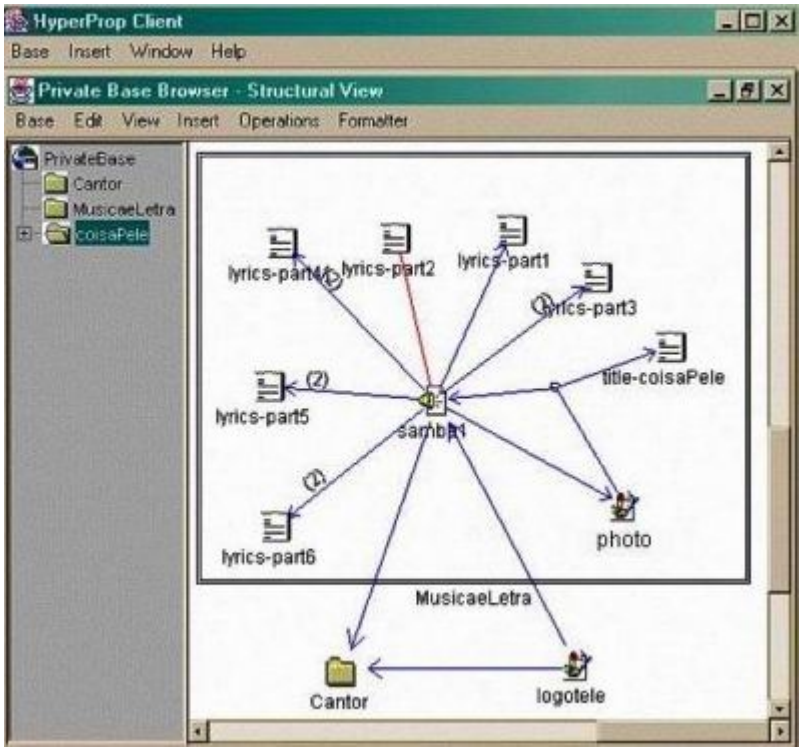


Figura 3-4 Visão estrutural

É importante destacar que a visão estrutural atual não ilustra a representação de portas existentes nos vértices compostos para exportar seus vértices internos. Observa-se, na Figura 3-4, que o vértice composto *MusicaeLetra* não ilustra nenhuma porta no seu desenho (retângulo) quando ocorre um relacionamento entre um vértice interno (*samba*) com um vértice externo (*Cantor*). Conforme a definição de composicionalidade da Seção 2.1, a aresta que relaciona os vértices *samba* e *Cantor* deveria ser ilustrada na Figura 3-4 pela união de duas arestas, uma pertencente ao conjunto M (Seção 2.1) relacionando o vértice composto *MusicaeLetra* com o vértice interno *samba* e outra aresta pertencente ao conjunto B (Seção 2.1) relacionando o vértice *Cantor* com o vértice *MúsicaeLetra*.

Apesar da visão estrutural ilustrada na Figura 3-4 apresentar algumas arestas cruzando a fronteira do vértice composto, na estrutura de dados da visão são definidas as

arestas do conjunto M e B (Seção 2.1) para relacionar vértices em diferentes níveis de aninhamento, preservando assim o conceito de composicionalidade.

No modelo NCM, cada elo criado referencia um conector (causal, de restrição etc.), podendo este conector ser reaproveitado na criação de novos elos. Para o desenho da visão estrutural (grafo composto), o reaproveitamento de conectores viola uma das restrições de grafos compostos - um mesmo vértice do grafo não pode estar diretamente contido em mais de um vértice composto (Noik, 1993). A solução adotada foi a definição do "vértice conector", utilizado para a representação do conector referenciado pelo elo. A identificação do "vértice conector", na visão estrutural, é formada pela identificação do elo que o referenciou mais a identificação do conector, garantindo, assim, sua unicidade de representação no grafo composto.

Quando o usuário deseja estabelecer um relacionamento entre apenas dois vértices, o "vértice conector" fica implícito no elo (aresta), não sendo exibido graficamente. No entanto, quando o autor especifica um relacionamento entre três ou mais vértices (vértices *samba*, *photo* e *title-coisaPele*), um "vértice conector" é desenhado e uma aresta é criada de cada um dos nós para o "vértice conector".

Um raciocínio semelhante pode ser aplicado no reaproveitamento de nós no modelo NCM. Quando um nó é reusado em diferentes perspectivas (níveis de aninhamento em vértices compostos), a visão estrutural cria um vértice para cada nó reaproveitado e a identificação desse vértice criado é feita pela junção da perspectiva na qual o nó se encontra mais a identificação do nó.

A visão estrutural também permite que as arestas sejam definidas como sendo direcionadas ou não. No caso do elo causal do modelo NCM, existem condições nos conectores relacionando vértices de origem do elo que devem ser satisfeitas para disparar a execução de ações sobre vértices de destino do elo. A navegação de páginas *Web* é um exemplo de relação causal clássico. Quando o usuário seleciona uma âncora em uma página *Web* (vértice de origem), ele é direcionado para outra página *Web* (vértice de destino). Nesse caso, definir a direção do elo torna-se interessante no desenho do grafo.

Já os elos de restrição do modelo NCM não apresentam nenhuma causalidade envolvida. Por exemplo, um elo de restrição pode especificar que um vértice (por exemplo, nó de áudio) deve terminar sua apresentação ao mesmo tempo que outro vértice

começa a dele. Nesse caso, a direção do elo relacionando os dois vértices não faz sentido. Um elo sem direção (elo de restrição no modelo NCM) pode ser visto entre os vértices *samba* e *lyrics- part2* na Figura 3-4.

Várias operações, tais como excluir, editar, expandir vértice composto, fechar vértice composto etc, podem ser realizadas sobre o desenho do grafo. As operações podem ser realizadas tanto com auxílio do *mouse* como através de caixas de diálogo (acionadas a partir da barra de *menu*). Para a criação de novas entidades (vértices compostos, vértices atômicos, arestas etc.), uma caixa de diálogo é aberta solicitando ao usuário informações específicas da entidade a ser inserida.

A visão estrutural do sistema HyperProp disponibiliza algumas características particulares para visualização de grafos. Por exemplo, os vértices podem ser especializados podendo associar ícones (imagens) aos vértices. No caso do uso do sistema HyperProp para autoria específica de documentos hipermídia baseado no modelo NCM, os ícones dos vértices atômicos que representam objetos de imagens apresentam a cor vermelha, já os vértices atômicos que representam objetos de áudio aparecem com uma caixa de som no ícone, os vértices compostos sempre são representados no formato de pastas amarelas etc.

A biblioteca anteriormente utilizada para representação de grafos na visão estrutural é a VGJ (*Visualizing Graphs with Java*) (VGJ, 1988). Ela foi inicialmente adotada para o desenvolvimento do sistema pois, de todas as bibliotecas estudadas, era a que provia recursos mais apropriados para o desenvolvimento da visão estrutural contemplando grafos compostos. Os principais recursos utilizados da biblioteca VGJ na implementação foi o algoritmo de *Spring* (Kamada & Kawai, 1989), utilizado para desenho dos grafos, e a representação de aninhamento de vértices (vértices compostos), através da qual um subgrafo pode ser representado por um único vértice com o uso da VGJ (Pinto, 2000). A Figura 3-4 foi desenhada usando essa biblioteca.

Atualmente, existem novas bibliotecas para desenhos de grafos que disponibilizam mais recursos para edição de grafos compostos que a biblioteca VGJ. A biblioteca JGraph (Alder, 2003), que trabalha com o padrão de projeto MVC (Model-View-Controller) (Gamma et al., 2002). Em JGraph, o “modelo” são os objetos utilizados para representar as entidades existentes no grafo, a “visão” são objetos responsáveis pela apresentação do

grafo para o usuário (características do leiaute de exibição) e o "controlador" são objetos que definem a maneira como a interface do usuário interage com usuário. Tal característica de implementação na biblioteca JGraph facilita a manutenção do código, pois os objetos estão bem divididos e definidos, diferentemente da biblioteca VGJ, onde as características de exibição das entidades estão representadas como atributo das mesmas. Nos próximos parágrafos, algumas diferenças encontradas nas bibliotecas JGraph e VGJ serão apresentadas.

A biblioteca JGraph disponibiliza as principais entidades de grafos presentes na biblioteca VGJ (*graph*, *nodes* e *edge*) e algumas outras entidades interessantes para representação de grafos compostos, tal como a entidade *Port* –objetos pertencentes aos vértices (compostos ou atômicos) que são referenciados pelas arestas (*edge*) para relacionar os vértices. Ou seja, uma aresta não referencia diretamente um vértice e sim uma de suas portas.

A entidade *Port* presente na biblioteca JGraph pode ser facilmente ilustrada nos desenhos de grafos, pois a biblioteca disponibiliza métodos para sua representação nos vértices, diferentemente da biblioteca VGJ que não apresenta a entidade porta como uma de suas entidades, sendo necessário sua extensão para a representação de portas.

A biblioteca VGJ disponibiliza apenas três algoritmos para desenhos de Grafos: *Tree Algorithm, Spring Embedder e Directed Graphs* (VGJ, 1988). Já a biblioteca JGraph disponibiliza oito algoritmos para desenhos de grafos: *Radial Tree, Circle, Moen's Algorithm, Simulated Annealing* e *Sugiyama* além dos mesmos três fornecidos pela biblioteca VGJ.

Uma nova versão da visão estrutural está sendo implementada de forma a torná-la mais flexível e portável na representação de grafos compostos. Para isso, a biblioteca JGraph (Alder, 2003) foi adotada.

A nova visão estrutural que está em desenvolvimento já fornece ao usuário diferentes algoritmos para o desenho de grafos, tais como: o algoritmo de *Spring*, Sugiyama (Sugiyama et al., 1981) e Circular (Wills, 1997). No entanto, alguns pontos ainda se encontram pendentes no desenvolvimentos e serão apresentados nos trabalhos futuros.

### 3.2.2. Visão Gráfica Temporal

A visão temporal é responsável pela especificação dos relacionamentos temporais entre vértices de um grafo composto, definindo suas posições relativas no tempo. Os vértices de um grafo composto na visão temporal são representados por retângulos, cujos comprimentos indicam suas durações de exibição/execução, conforme ilustrado na Figura 3-5.

A interface desenvolvida é composta por dois eixos: o horizontal, que representa a escala temporal à qual os vértices estão associados, e o vertical, responsável por alocar os vértices a recursos da arquitetura sendo especificada. Um exemplo de utilização da ferramenta para definir arquiteturas de sistemas, seria aquele em que os recursos poderiam representar os processadores de um sistema distribuído, com os componentes alinhados no tempo representando o revezamento na utilização dos recursos.

Para o caso particular da autoria hipermídia, os recursos representam dispositivos (monitor, placa de som etc.) utilizados pelos nós durante a apresentação. No modelo NCM e na linguagem NCL, esse relacionamento entre nós (vértices) e os dispositivos de saída é realizado através dos descritores dos nós, conforme explicado anteriormente na Seção 2.3.1.

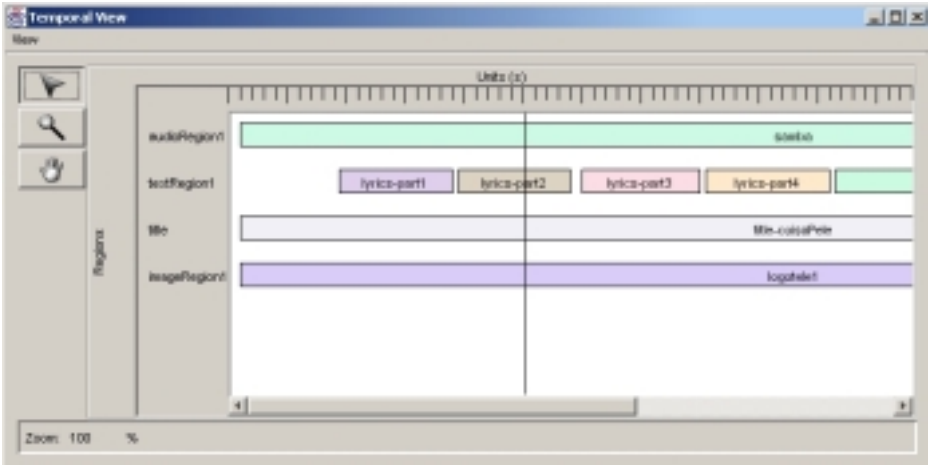


Figura 3-5 Visão Temporal

Na visão temporal, é possível que o autor aplique Zoom (*in* e *out*) no desenho, possibilitando uma visão geral de todo o sistema ou uma visão específica em um determinado ponto da visão. Além disso, a visão temporal possui um eixo vertical móvel,

que indica quais vértices estão visíveis no instante de tempo por ele demarcado, refletindo esses mesmos vértices na visão espacial do sistema, conforme será visto na Seção 3.3 (Moura, 2001).

Assim como a visão estrutural, uma nova visão temporal, ilustrada na Figura 3-5, está em desenvolvimento. Entretanto, alguns problemas como: *i)* representação de vértices compostos; *ii)* desenho das arestas entre os vértices; *iii)* a edição dos vértices; e *iv)* integração do filtro olho-de-peixe, ainda não foram resolvidos.

### 3.2.3. Visão Gráfica Espacial

A visão espacial possibilita ao autor determinar graficamente a disposição dos vértices de um grafo composto com relação aos recursos utilizados na arquitetura de sistema, e como as características espaciais de exibição dos vértices devem variar durante a apresentação de uma arquitetura (Moura, 2001). Ou seja, a visão espacial pode ser definida em dois momentos: o primeiro momento ocorre durante a definição do leiaute espacial da apresentação de uma arquitetura de acordo com os recursos disponíveis e o segundo momento ocorre na especificação de como os vértices são alterados (posicionamentos, tamanhos etc) ao longo da apresentação da arquitetura.

Os recursos disponíveis podem ser analisados como um grafo composto. Por exemplo, um dispositivo de saída “monitor” pode ter sua área de visualização dividida em diferentes regiões, onde cada região pode apresentar diferentes tipos de vértices da arquitetura.

No caso da linguagem NCL, a especificação do leiaute da apresentação do documento hipermídia pode ser representada por um grafo composto onde os vértices compostos são representados pelos elementos *layout*, *topLayout* e *region*. Já os vértices atômicos são representados apenas pelo elemento *region*. Observe que o elemento *region* pode ser representado tanto como um vértice atômico quanto como um vértice composto. Isso ocorre porque a linguagem permite que uma região contenha outras regiões. Maiores detalhes sobre a linguagem podem ser obtidos em (Muchaluat et al., 2003).

Um exemplo de edição de um leiaute espacial para apresentação de um documento hipermídia baseado na NCL pode ser visualizado na Figura 3-6. Nessa figura, o autor

selecionou a região *r7* e, em seguida, solicitou a visualização das propriedades da região selecionada. Todas as propriedades são exibidas numa caixa de diálogo. Caso o autor altere o valor de algum parâmetro, por exemplo o parâmetro *Height*, a figura desenhada no editor é automaticamente redesenhada com o novo valor atribuído.

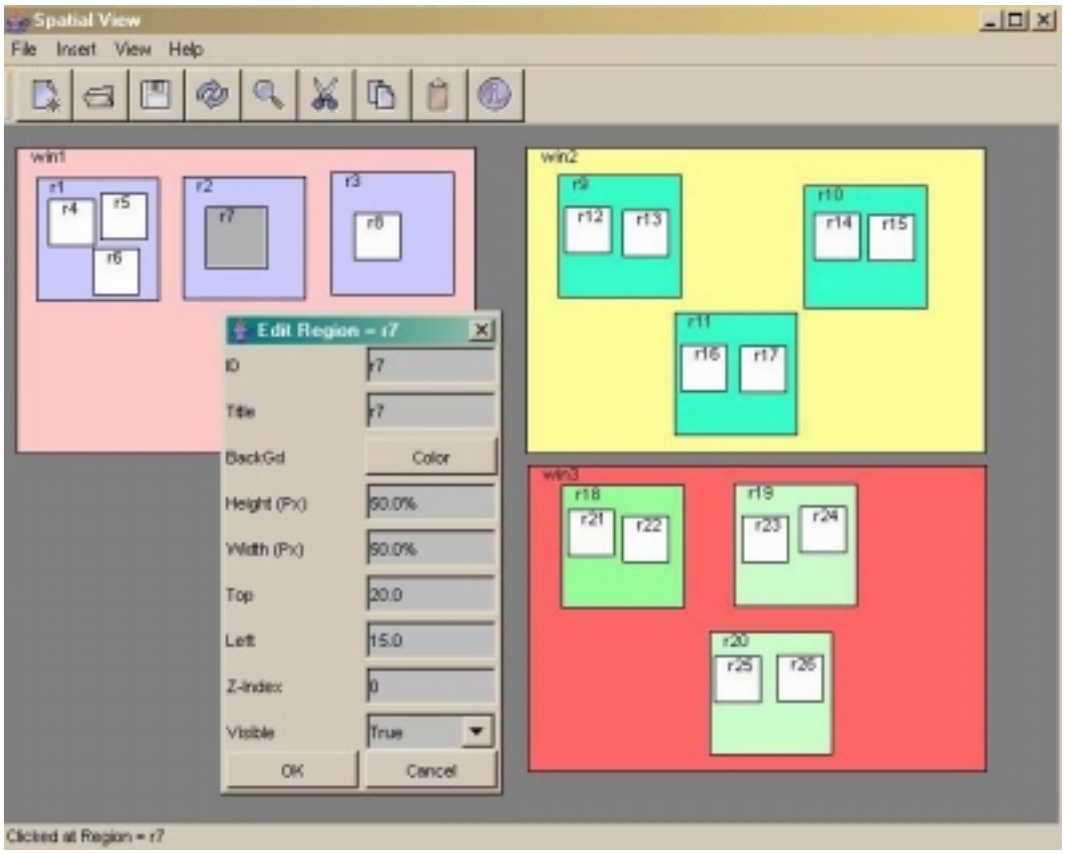


Figura 3-6 Visão Espacial para edição de *Layout* de apresentação.

Quando o símbolo “*%*” aparece em alguma das propriedades *Height*, *Width*, *Top* e *Left* significa que o valor dessa propriedade é proporcional ao valor do elemento espacial que contém a região. Por exemplo, no caso da região *r7*, a propriedade *Height* equivale a “5*0%*” da propriedade *Height* de seu vértice pai (a região *r2*).

A validação dos dados passados pelo usuário é sempre realizada e, caso alguma inconsistência seja detectada, uma mensagem de erro é enviada ao usuário. Por exemplo, se o usuário escrever o valor de um número por extenso ao invés de digitar apenas os números, ou até mesmo, digitar duas vezes o símbolo “**,**” no campo cuja entrada de dados deve ser apenas um valor numérico, uma mensagem de erro será enviada para o usuário informando a incompatibilidade dos dados.

Quando um vértice composto é movimentado, todos os vértices internos a ele são movimentados também, mantendo as posições relativas dos vértices filhos ao vértice pai. Se, durante a edição do leiaute, o usuário movimentar um vértice para fora da área delimitada por seu vértice pai, uma mensagem é enviada para o usuário informando que o vértice está fora da área delimitada por seu pai. Por exemplo, se o usuário posicionar o vértice *r7* para uma área fora do vértice *r2*, uma mensagem de alerta é enviada.

É importante destacar que, durante a apresentação de uma arquitetura, se um vértice estiver posicionado fora da área delimitada por seu vértice pai, todos os objetos relacionados a esse vértice (recurso) não serão exibidos. Por exemplo, na apresentação de documentos hipermídia, o usuário pode posicionar uma determinada região (destinada para exibição de vídeos) para fora da janela (monitor) impossibilitando sua visualização.

O autor pode, através da interface gráfica, criar, editar e apagar os elementos do grafo composto destinados à apresentação do leiaute da arquitetura, assim como alterar as posições dos vértices com relação a seu vértice pai apenas com o auxílio do mouse.

Em (Moura, 2001), os relacionamentos espaço-temporais dos vértices ao longo da apresentação são analisados. Os relacionamentos espaço-temporais envolvem, além das relações de atributos espaciais, os atributos temporais e/ou ações associadas à apresentação dos vértices do grafo (inicia, interrompe ou termina exibição).

Um exemplo de relacionamento espaço-temporal seria determinar que quando o vértice *"x"* (por exemplo, um nó de imagem) estiver posicionado na posição (20, 30), sua exibição deve ser encerrada. Outro exemplo, seria que, durante a apresentação dos vértices *"x"* e *"y"*, eles devem permanecer alinhados à direita enquanto o terceiro vértice *"z"* estiver sendo apresentado.

Neste trabalho, apenas a especificação do leiaute foi desenvolvido no editor espacial, deixando-se para trabalhos futuros a edição de relacionamentos espaço-temporais.

## 3.3. Sincronização entre as visões

Para permitir a sincronização entre as várias formas de autoria gráfica, e também o funcionamento das visões gráficas com a edição declarativa de maneira integrada, adotou-se a arquitetura ilustrada na Figura 3-7.

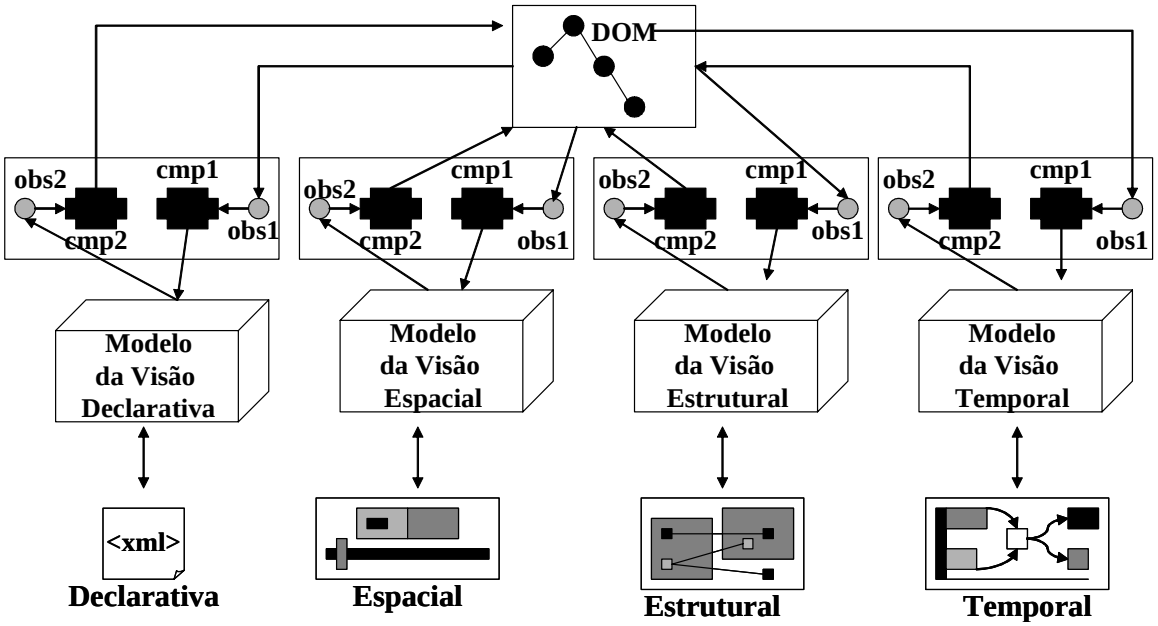


Figura 3-7 – Arquitetura de integração das visões

O modelo de dados utilizado para integração das visões é o DOM – *Document Object Model* (DOM, 2004), padronizado pelo W3C para armazenar e manipular árvores XML, no caso, a árvore que descreve a estrutura do grafo.

Para cada uma das visões, existe um par de compiladores responsáveis, respectivamente, por traduzir a árvore DOM para a estrutura de dados utilizada para exibição do grafo, e por converter essa estrutura de dados na representação do modelo de integração, no caso o DOM.

Alguns compiladores foram desenvolvidos e integrados ao sistema HyperProp neste trabalho, tais como os compiladores que convertem objetos Java do modelo NCM em diferentes visões na estrutura DOM. É importante destacar que o sistema HyperProp já disponibilizava um compilador capaz de converter arquivos NCL (formato declarativo) em objetos Java NCM (Silva et al., 2004). No entanto, somente com o desenvolvimento

dos novos compiladores foi possível disponibilizar um ambiente de autoria sincronizado entre as formas de edição gráfica e declarativa no ambiente HyperProp.

Além dos compiladores, cada visão tem um par de observadores (um para cada compilador). Sempre que a árvore DOM é modificada, os *observadores1*, na Figura 3-7, são notificados e acionam os respectivos compiladores para solicitarem as atualizações dos modelos das visões.

De forma análoga, os *observadores2* recebem eventos de mudança do modelo da respectiva visão (resultantes de ações do usuário sobre a ferramenta de edição) e requisitam que os compiladores correspondentes reflitam essas mudanças no modelo de integração (DOM).

No exemplo do sistema HyperProp, os compiladores de documentos XML foram implementados usando o pacote JAXP (*Java API for XML Processing*) (JAXP, 2003).

Para a visão textual, os compiladores são triviais, uma vez que o modelo da visão textual é também baseado na árvore DOM de documentos XML. Para as visões gráficas, foi necessário desenvolver compiladores apropriados para tradução dos modelos (Silva et al., 2004), uma vez que suas estruturas de dados são baseadas na implementação Java do modelo NCM.

Os módulos que formam os compiladores devem ser específicos para cada arquitetura. Por exemplo, para adaptar o sistema para trabalhar com especificações de arquiteturas de sistemas usando os conceitos de ADL (Capítulo 2), os modelos das visões e os compiladores devem ser adaptados. Evidentemente, como a base da edição são as entidades elementares de grafos compostos, tal adaptação modifica apenas os atributos das entidades básicas (vértices compostos, atômicos, arestas etc.).

A sincronização das diversas visões da ferramenta se dá não apenas pela integração das estruturas de dados dos diversos modelos, mas também pelas suas exibições. Por exemplo, se um vértice é escolhido como foco de exibição em qualquer das visões, ele automaticamente vira o foco em todas as outras visões. Particularmente, se um vértice é escolhido como foco na visão estrutural, ele vira o foco para a exibição de toda a cadeia de acontecimentos, dele derivada, na visão temporal (Costa, 1996).

A Figura 3-8 ilustra um exemplo de sincronização entre a visão espacial atual e a declarativa, onde a região *textRegion1* foi selecionada pelo usuário e automaticamente a

linha referente a essa região foi destacada no editor declarativo. Conforme o usuário altera o posicionamento da região *textRegion1*, selecionada no editor espacial, os valores dos atributos referentes ao posicionamento da região no editor textual são imediatamente alterados.

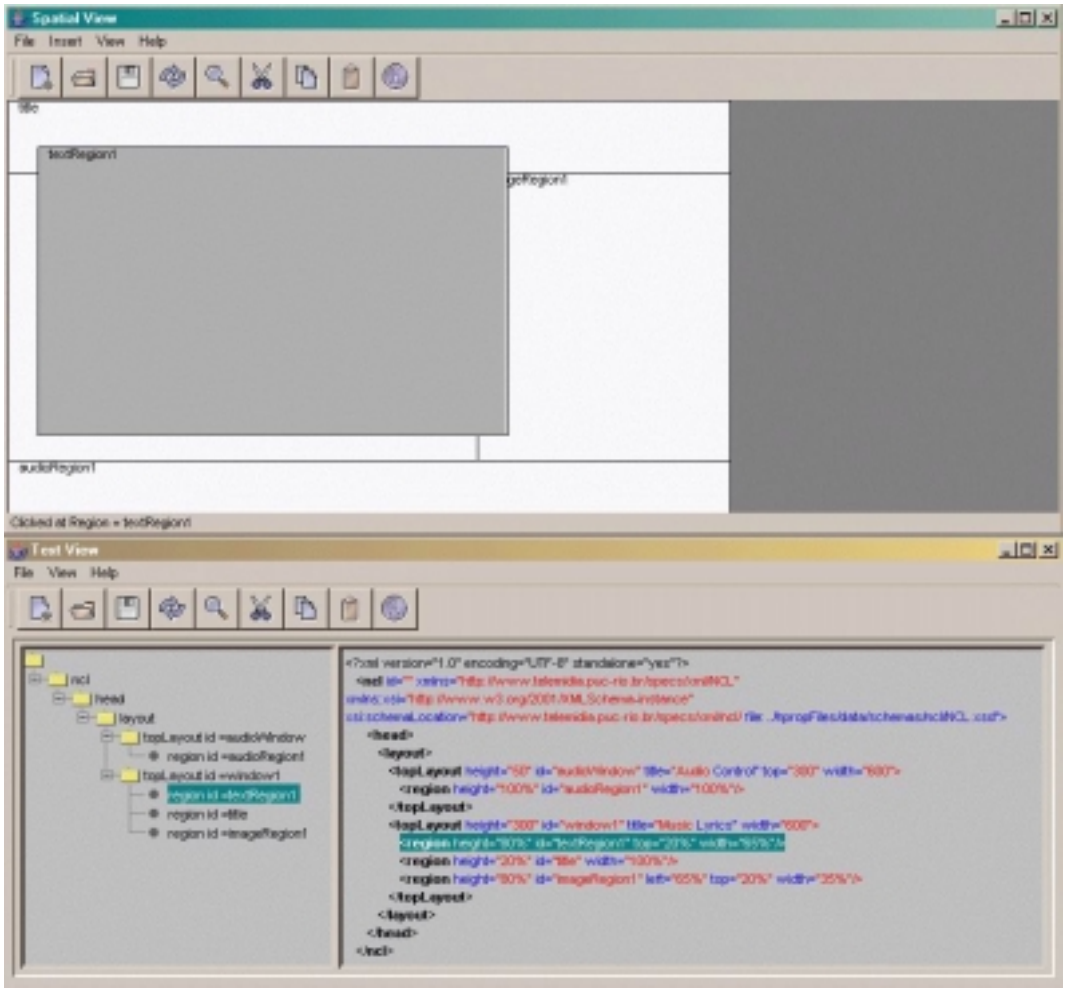


Figura 3-8 Visão Espacial sincronizada com Visão Textual

Um outro exemplo de exibição sincronizada se dá entre as visões temporal e espacial. A visão temporal possui uma barra vertical móvel (vide Figura 3-5), que permite ao autor fixar um instante de tempo para estabelecer alguma análise do sistema sendo especificado. Esse eixo móvel pode ser útil para analisar o comportamento espacial da apresentação em um dado instante (Moura, 2001). Por exemplo, quando o eixo vertical está sobre determinados vértices no instante de tempo *"t1"*, esses vértices são exibidos em seus respectivos dispositivos de saída na visão espacial.

Ainda outro exemplo de visões sincronizadas é discutido no próximo capítulo, quando filtragens na exibição de vértices em uma visão implicam em filtragens nas demais visões.

Uma dificuldade encontrada na sincronização das visões gráficas com a declarativa foi com relação ao texto gerado no editor declarativo, pois os elementos do documento XML exibidos no editor não apresentam nenhuma ordem de classificação (por exemplo, alfabética). Vale destacar que a estrutura hierárquica dos nós da árvore é mantida. Por exemplo, o usuário pode organizar o documento XML no editor declarativo seguindo a ordem alfabética dos nomes dos elementos. No entanto, quando um evento de sincronização for disparado por uma das visões gráficas, essa organização criada pelo usuário é perdida. Isso ocorre pois a forma na qual o DOM organiza seus os objetos na sua árvore é refletida diretamente na visão textual.