Integração de Ferramentas Gráficas e Declarativas na Autoria de Arquiteturas Modeladas através de Grafos Compostos*

Rogério Miguel Coelho, Rogério Ferreira Rodrigues, Luiz Fernando Gomes Soares
Departamento de Informática, PUC-Rio
rmc@inf.puc-rio.br, rogerio@telemidia.puc-rio.br, lfgs@inf.puc-rio.br

Abstract

This paper presents a set of tools to help the implementation of applications for authoring system architectures based on composite graphs. The tools are divided in four graph views: structural, temporal, spatial, and declarative. The views are synchronized in order to offer an integrated authoring environment. Moreover, the views include filtering mechanisms to simplify the specification of more complex architectures. The tools were specialized to hypermedia authoring, but they can be used in other domains, such as software architecture specifications, formal specification tools, workflow projects, etc.

1. Motivação e Conceitos Preliminares

Um *grafo composto G* é definido por um conjunto de vértices *V* e um conjunto de arestas *A*, onde cada elemento de *A* define uma relação binária entre dois elementos de *V* [9]. O conjunto de vértices *V*, por sua vez, pode ser particionado em dois subconjuntos: *CV* e *AV*. Os vértices pertencentes a *CV* são chamados de vértices compostos e identificam, cada um, um subconjunto de vértices do grafo que são por ele contidos, formando assim um subgrafo de *G*. Os vértices pertencentes a *AV* são chamados de *vértices atômicos*.

Mais precisamente, um grafo composto é definido por uma tupla G = (V, A, I), onde:

V = CV ⋃ AV

A ⊆ (V x V)

I ⊂ CV x V é um conjunto de arcos de inclusão.

Duas restrições ainda se fazem necessárias: um vértice composto *cv* pertencente a *CV* não pode conter a si mesmo e nem recursivamente conter algum vértice que o contenha; e um mesmo vértice do grafo não pode estar diretamente contido em mais de um vértice composto.

Como a propriedade de *composicionalidade* é importante no tratamento de alguns grafos compostos, pode-se restringir a definição do conjunto de arestas *A*. Dessa forma, define-se, no contexto deste artigo, um *grafo com composicionalidade*, grafos compostos em que o conjunto de arestas *A* é formado pela união dos conjuntos *B* e *M*, tais que:

a) B ⊆ (V x V) e, para qualquer par de vértices (*v,c*) pertencentes a *B*, ou ∃ *cv* ∈ *CV* tal que *v* ∈ *cv* e *c* ∈ *cv*, ou ∀ *cv* ∈ *CV*, *v* ∉ *cv* e *c* ∉ *cv*. Em resumo, os vértices de uma aresta ou estão contidos em uma mesma composição ou não estão contidos em nenhuma composição. Essa restrição faz com que uma aresta definida em *B* não possa cruzar a fronteira de um vértice composto.

b) M ⊆ (CV x V) e, para qualquer par de vértices (c*v,v*) pertencentes a *M, v* ∈ *cv*. Em resumo, *M* constitui um conjunto de arestas de pai para filhos, sendo, portanto, um subconjunto de *I*.

A Figura 1 ilustra um grafo composto com composicionalidade. Na figura, vértices cn_i são vértices compostos, vértices an_i e c_i representam vértices atômicos, as arestas b_i pertencem ao subconjunto *B* de arestas, e as arestas m_i pertencem ao subconjunto *M*. Note que relacionamentos entre vértices que não estejam diretamente contidos em um mesmo vértice composto são construídos através de concatenações de arestas definidas em *B* e em *M*. Na figura, os arcos de *I* são representados pela inclusão de um vértice em outro.

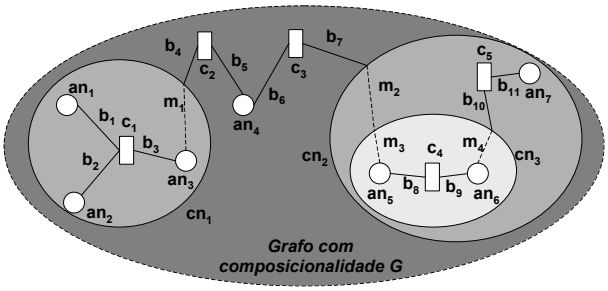

Figura 1 - Grafos compostos com composicionalidade

A estrutura de dados de um grafo composto é bastante útil na modelagem de sistemas que apresentam composicionalidade ou, no mínimo, relações de aninhamento de componentes. Exemplos de tais sistemas podem ser encontrados nas configurações de software modeladas por ADLs (*Architecture Description Languages*) [1], em ferramentas de descrição formal de arquiteturas [4], ou ainda na especificação de documentos hipermídia [11].

* Este trabalho foi realizado com apoio do Fundo Setorial para o Desenvolvimento Tecnológico das Telecomunicações (FUNTTEL), através do contrato 0594/02; e do CNPq.

Muitas ADLs preservam a noção de composicionalidade em suas entidades de primeira classe: componentes e conectores. Uma configuração em ADL pode ser representada por um grafo com composicionalidade, onde os vértices do grafo representam componentes e conectores. As arestas de *B* representam as ligações (*binds*) entre portas de componentes e papéis de conectores (seguindo a terminologia da ADL *Wright* [1]). Componentes e conectores compostos são vértices compostos que contêm componentes e conectores como elementos internos. Portas de um componente composto podem exportar interfaces (portas) de seus componentes internos, através de mapeamentos entre portas (representados por arestas definidas em *M*). Da mesma forma, papéis de um conector composto podem exportar papéis de seus conectores internos, através de mapeamentos entre papéis (também representados por arestas definidas em *M*).

Ferramentas de especificação formal de arquiteturas têm como uma de suas bases a composicionalidade. Algumas dessas ferramentas apresentam um conceito de composicionalidade mais amplo do que as ADLs (descritas no parágrafo anterior) e os modelos hipermídia (expostos no próximo parágrafo). Forma [4], por exemplo, define composições como contendo conectores, componentes e outras composições. No entanto, ao contrário das ADLs, que restringem os mapeamentos apenas entre componentes ou entre conectores, interfaces de uma composição em Forma podem exportar interfaces de qualquer um de seus componentes filhos (exportações que, novamente, podem ser representadas por arestas de *M*). Uma arquitetura pode ser representada por um grafo composto tendo como vértices composições, componentes e conectores. Ligações entre os vértices preservam a composicionalidade do grafo (representadas por arestas de *B*).

De forma análoga às ADLs e às ferramentas de especificação formal, muitos modelos de linguagens hipermídia preservam a noção de composicionalidade, embora quase sempre limitada a apenas uma de suas entidades (um subconjunto de seus vértices): os nós. Mesmo linguagens com modelos bastante semelhantes às ADLs, como o modelo NCM (*Nested Context Model*) da linguagem NCL (*Nested Context Language*) [11], embora apresentem a noção de conector como entidade de primeira classe, não permitem em suas versões atuais conectores compostos. A maioria dos modelos, entretanto, como o da linguagem SMIL [13], não modela os relacionamentos entre nós como entidades de primeira classe. Desse modo, não existe a noção de conectores, mas apenas de elos interligando nós. Independente dessas diferenças, em todos os casos, um documento hipermídia pode ser representado por um grafo composto, onde seus nós (composições ou não) são representados pelos vértices do grafo.

Em modelos hipermídia com conectores, os conectores também são representados por vértices no grafo. Similar às ADLs, portas de um nó de composição podem exportar interfaces (no caso âncoras ou portas de composições filhas) de seus componentes internos através de mapeamentos (representados pelas arestas de *M*), mantendo a noção de composicionalidade. Nesses modelos, também similar às ADLs, ligações (*binds*) são estabelecidas entre os papéis de um conector e interfaces dos nós (representadas pelas arestas de *B* no grafo composto). Elos hipermídia são definidos por um conector juntamente com todas as arestas que o tocam. Um elo, dessa forma, pode ser multiponto, ou seja, ligar mais de duas interfaces de nós. Quando as interfaces que se ligam a um conector são portas de um nó composto, os mapeamentos dessas portas em outras interfaces de nós internos, recursivamente, caracterizam os pontos terminais de um elo.

Em modelos hipermídia onde não existe a noção de conector, mas apenas a de elo (podendo esse elo ser multiponto), um documento pode ainda ser representado por um grafo composto, onde é criado, para cada elo, um vértice "falso" (como se o vértice representasse um conector inexistente), convergindo para ele todas as arestas ligadas a vértices (representando nós) definidos pelos pontos terminais das extremidades do elo.

Arquiteturas de software, de documentos, de modelos formais, enfim, arquiteturas de sistemas, podem ser especificadas através de uma linguagem textual ou através de uma linguagem gráfica. Nesse último caso, permite-se a especificação através de operações sobre o desenho do grafo composto que representa a arquitetura, tal como o ilustrado na Figura 1. Existem várias razões tanto para se usar uma linguagem textual como para se usar uma linguagem gráfica, que serão discutidas ao longo deste artigo. Por não haver uma solução ótima e única, o ideal é se trabalhar com os dois paradigmas. Essa junção só terá real valor, no entanto, se as visões atuarem de forma sincronizada, ou seja, se alterações em uma visão se refletirem na outra, em uma edição de fato integrada.

Em um grafo composto, tanto as arestas quanto as composições aninhadas de vértices podem representar diversos tipos de relações. Em todas as arquiteturas de sistemas citadas neste texto, duas categorias de relações têm destaque especial: as relações temporais e as espaciais. Uma relação temporal entre dois vértices diz respeito às suas ordenações no tempo. De forma análoga, uma relação espacial entre dois vértices diz respeito aos seus posicionamentos no espaço de recursos do sistema. Ambas as relações podem ser representadas pelo

aninhamento de composições (por exemplo, uma composição seqüencial em SMIL representa o ordenamento temporal das apresentações de seus nós filhos em seqüência), ou por arestas no grafo (por exemplo, um conector em ADL pode especificar que a computação de um componente deve seguir a de outro, ambos ligados ao conector).

A importância das relações temporais e espaciais torna interessante também poder especificar uma arquitetura pelo posicionamento dos vértices de seu grafo composto no tempo e no espaço, gerando duas novas visões (temporal e espacial) de edição, além das visões estrutural e textual do grafo. Mesmo quando toda a estrutura de aninhamento de um grafo composto (dada pela visão estrutural) se restringe ao relacionamento temporal (ou ao espacial), a edição através das visões temporal e espacial pode ser útil, trazendo informações complementares às existentes na visão estrutural (por exemplo, a duração dos vértices). Sendo assim, além da integração das visões estrutural (gráfica) e textual já mencionada, é desejável que também as visões espacial e temporal estejam a elas sincronizadas, e evidentemente sincronizadas entre si.

Este artigo trata do desenvolvimento de uma ferramenta para especificação de arquiteturas de sistemas (passíveis de serem modeladas por grafos com composicionalidade), baseada na integração de visões textual, estrutural, temporal e espacial da arquitetura. O artigo estende trabalhos anteriormente realizados dentro do âmbito de ferramentas gráficas para especificação de documentos hipermídia no sistema Hyperprop [12], um sistema para autoria e formatação de documentos hipermídia baseados no modelo NCM. As extensões envolvem tanto a incorporação das visões textual e espacial às antigas ferramentas, quanto à remodelagem para aplicação em outros domínios de arquitetura. Mecanismos de filtragem baseados em técnicas "olho-de-peixe" [5] são apresentados neste artigo de forma integrada, adequando a ferramenta de edição para a especificação desde as arquiteturas mais simples até as mais complexas.

O artigo encontra-se organizado como a seguir. A Seção 2 trata da descrição textual de arquiteturas de sistemas baseadas em grafos com composicionalidade. A Seção 3 apresenta a ferramenta de edição, com suas quatro visões. Nessa seção, o mecanismo de sincronização para a integração das várias visões é também descrito. A Seção 4 destaca as técnicas de filtragem implementadas. Para descrição da ferramenta, de suas visões e das técnicas de filtragem, a arquitetura de documentos hipermídia foi escolhida como exemplo, uma vez que o alvo primeiro da ferramenta foi a especificação de documentos segundo o modelo NCM e a linguagem NCL. A Seção 5 compara o editor desenvolvido com algumas ferramentas para autoria de documentos hipermídia. Por fim, a Seção 6 tece as conclusões e descreve os trabalhos futuros.

2. Especificação Declarativa de Arquiteturas de Sistemas com Composicionalidade

A definição de uma linguagem textual para descrição de grafos compostos traz vários benefícios em relação a uma descrição puramente gráfica. Um primeiro ponto é a possibilidade de criar e manipular grafos com editores de texto convencionais, desobrigando que o autor faça uso de uma ferramenta gráfica, muitas vezes proprietária. Outro aspecto diz respeito à expressividade na descrição da arquitetura do sistema (hiperdocumento, software, configuração etc.), pois, por mais simples que seja o editor textual, o autor sempre terá, a seu dispor, todos os comandos para a descrição da arquitetura. Uma terceira vantagem é estabelecer um formato padrão e aberto para intercâmbio dos grafos entre as mais diversas ferramentas. O formato textual também pode favorecer a depuração, pelo próprio ser humano, de uma especificação incorreta ou corrompida. Por fim, uma descrição textual permite que o autor edite o grafo apenas com o teclado (criação, cópia e colagem etc.), o que pode, em certas circunstâncias, trazer maior agilidade no processo de autoria. Evidentemente, também existem vantagens da edição gráfica sobre a edição textual, que serão comentadas na próxima seção.

Um padrão adequado para descrição declarativa de grafos compostos é a meta-linguagem XML [14]. XML favorece a especificação estruturada, podendo ser usada na construção de linguagens para descrição das mais variadas arquiteturas.

Para dar suporte à especificação de grafos com composicionalidade, conforme definidos na Seção 1, uma linguagem XML deve fornecer elementos básicos para especificação de vértices atômicos, vértices compostos, definição do conteúdo dos vértices compostos, especificação dos mapeamentos entre vértices compostos e seus componentes internos, e a definição das associações entre os vértices.

A partir da especificação em XML, um módulo de *parser* pode gerar a árvore do documento, permitindo que uma aplicação percorra e manipule o grafo composto.

Como exemplo, a linguagem NCL 2.0 [11] é uma linguagem declarativa que permite a especificação textual de um documento hipermídia respeitando a propriedade de composicionalidade (Seção 1).

Como pode ser observado na Figura 2, a linguagem oferece um elemento *composition*, que permite criar nós de composição (vértices compostos). As composições podem conter nós de mídia (vértices atômicos, como por

exemplo, textos, imagens, vídeos etc.) outros nós de composição e elos. Elos (elemento *link*), por sua vez, são formados por conectores (vértices atômicos) e pela associação (arestas) entre nós e conectores, especificadas pelo elemento *bind* da linguagem. A linguagem permite que sejam definidos mapeamentos entre as composições e os nós contidos na composição, através do elemento *port*. Os mapeamentos são feitos de portas de uma composição para âncoras dos nós de mídia, ou para portas de nós de composição internos. Para manter a composicionalidade, as associações dos elos devem referenciar exclusivamente portas/âncoras de nós diretamente contidos na composição que contém o elo.

```
<?xml version="1.0" encoding="UTF-8"?  …>
<ncl id="coisaDePele" ... >
<head>
...
 <descriptorBase>
  <descriptor id="audio_d1" .../>
  <descriptor id="img_d1" .../>
 </descriptorBase>
</head>
<body>
<composition id="coisaPele">
 <port id="musica" component="samba"/>
...
 <audio descriptor="audio_d1" id="samba" src="smb.wav">
  <area id="part1" begin="8.4s" end="18.0s"/>
  ...
 </audio>
...
 <img descriptor="img_d1" id="foto" src=" foto.jpg"/>
 ...
 <linkBase>
  <link id="link1" xconnector="starts.xml">
   <bind component="samba"
role="on_x_presentation_begin"/>
   <bind component="foto" role="start_y"/>
  </link>
  ...
  </linkBase>
 </composition>
 </body>
</ncl>
```

Figura 2 – Exemplo de documento NCL.

A linguagem NCL define as características de apresentação dos nós em uma entidade separada, representada pelo elemento *descriptor*. O descritor pode especificar tanto a duração como o posicionamento espacial do nó. Essas informações são a base da construção das visões temporal e espacial do documento, mencionadas na seção anterior. Além dos descritores, os conectores hipermídia contêm as expressões de relacionamento entre os nós, complementando as informações espaço-temporais do documento necessárias à construção das visões mencionadas.

NCL permite que um nó esteja ao mesmo tempo contido em mais de uma composição. Na realidade, são elementos XML distintos no documento, onde um dos elementos é referenciado pelos demais para reuso de seus atributos. Essa característica é bastante comum em outras arquiteturas de sistemas e facilmente modelada na estrutura de grafo composto. Em termos de grafo, cada elemento XML é considerado como um vértice distinto, mas com a facilidade de que as modificações nos valores dos atributos em um vértice possam automaticamente refletir na alteração dos atributos em outros vértices do grafo.

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

3.1 Edição Textual

A maioria dos programas para edição de documentos textuais (*NotePad*, *WordPad*, *vi* etc.) apresenta algumas limitações no que tange à autoria de documentos XML, tais como: i) inexistência de um depurador, que permita ao usuário localizar erros no documento com base nas DTDs (ou Schemas) que o especificam; ii) ausência de destaque das informações estruturais (elementos, atributos etc.) do documento em relação ao seu conteúdo; iii) dificuldade de sincronização do editor com outras ferramentas de autoria.

Ferramentas projetadas para autoria XML (Salix [2], por exemplo) provêem parte dos recursos mencionados no parágrafo anterior (i e ii). Porém, o fato da ferramenta analisada não ser "aberta" inviabiliza sua integração a outros sistemas. O mesmo ocorre com outros editores XML (*UltraEdit32*, *XMLSpy* etc.) Um outro ponto negativo em ferramentas proprietárias é a dificuldade, ou mesmo impossibilidade, de acrescentar novas funcionalidades às ferramentas, se necessário.

A fim de obter uma ferramenta para edição de arquiteturas descritas conforme a proposta apresentada na Seção 2 e com as características levantadas no início desta seção, optou-se por desenvolver um editor, conforme ilustrado na Figura 3.

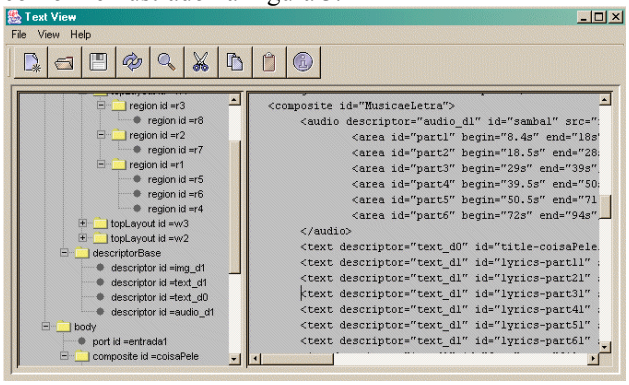

Figura 3 – Visão declarativa para autoria

A interface do editor é dividida em duas partes. A área esquerda apresenta a árvore XML do documento, na qual os vértices compostos podem ser abertos (exibindo todos os vértices diretamente pertencentes a ele) ou fechados. Já a direita apresenta o documento na forma textual. As alterações feitas em um dos lados do editor

refletem automaticamente no outro. Além disso, ao clicar sobre um determinado vértice na parte esquerda do editor, a linha textual referente a ele é automaticamente destacada.

O texto apresentado no lado direito do editor apresenta os elementos e atributos XML em outras cores, para facilitar a identificação dos mesmos. Sempre que uma especificação de documento encontra-se inconsistente com as definições das DTDs (ou Schemas) do documento XML, passadas como parâmetro na declaração do documento XML, o módulo de validação da ferramenta emite uma mensagem identificando os problemas ao autor.

Apesar das vantagens comentadas na seção anterior, a edição puramente declarativa pode apresentar algumas limitações: dificuldade no entendimento da estrutura do documento, conforme a quantidade de informação cresce; necessidade de conhecimento da sintaxe e semântica da linguagem usada na autoria que, dependendo da complexidade, pode envolver um tempo longo de aprendizado; e, para certos usuários, o tempo gasto especificando o documento na forma puramente textual torna o processo de autoria trabalhoso.

3.2 Edição Gráfica

Uma alternativa para a autoria de arquiteturas de sistemas baseadas em grafos compostos é o uso de ferramentas gráficas. Três pontos positivos podem ser destacados nesse paradigma comparado à autoria declarativa. O primeiro é a visão global e prévia do documento (WYSIWYG – *What You See Is What You Get*), na qual o autor pode ter um retorno visual mais imediato durante a edição [10]; o segundo é o uso de ícones para rotular as ações de edição – o autor precisa somente estabelecer a associação entre as imagens de componentes da interface gráfica com eventos de edição, ao invés de guardar nomes de comandos; e o terceiro é a facilidade de reuso – pequenas partes dos documentos podem ser facilmente identificadas (através de figuras) e reaproveitadas na elaboração de um novo documento.

Alguns pontos negativos podem, porém, ser apontados na autoria gráfica, como, por exemplo, a necessidade de mais recursos (memória, CPU, tamanho de tela etc.) para oferecer suporte à edição.

Com o intuito de oferecer ao autor um ambiente para autoria de arquiteturas baseadas em grafos compostos que agregasse as vantagens de ambos os paradigmas (textual e gráfico), o editor foi estendido e sua ferramenta de edição textual (Figura 3) foi integrada a outras três ferramentas de edição gráfica: estrutural, temporal e espacial.

3.2.1 Visão Gráfica Estrutural

A visão estrutural 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 (*binds* ou mapeamentos), além de realizar operações de inclusão.

A interface da visão estrutural está dividida em duas partes, conforme ilustrado na Figura 4. O lado esquerdo apresenta uma árvore com todos os vértices de composição do grafo composto, possibilitando ao usuário abri-los ou fechá-los. A visão em árvore mostra, na realidade, as arestas definidas no conjunto *I* do grafo (Seção 1). Para não tornar a visão poluída, filtros podem ser aplicados à visão, conforme será melhor comentado na Seção 4. No exemplo da Figura 4, o autor optou por visualizar na árvore apenas os vértices compostos do grafo.

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

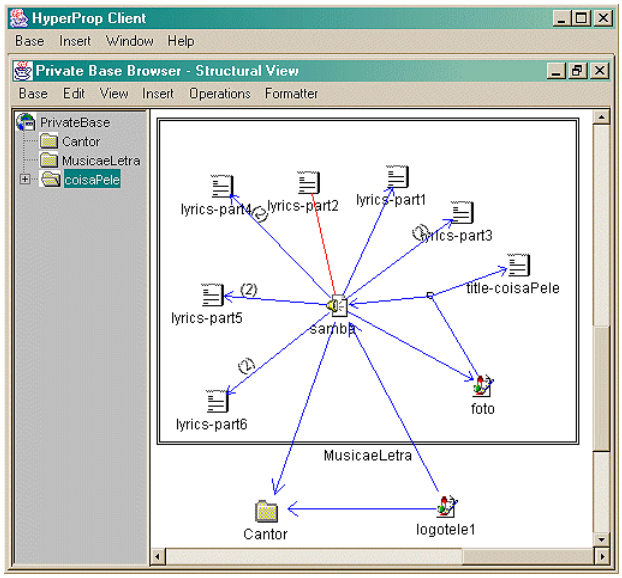

Figura 4 - Visão estrutural

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

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 relacionando aqueles vértices.

A ferramenta desenvolvida define um tipo especial de vértice denominado conector. Quando o autor deseja estabelecer uma relação entre apenas dois vértices, o vértice conector fica implícito na aresta, não sendo exibido graficamente. No entanto, quando o autor especifica uma relação entre três ou mais vértices (vértices *samba*, *foto* e *title-coisaPele*), um conector é desenhado, e uma aresta ponto-a-ponto é criada de cada um dos vértices para o conector. A visão estrutural também permite que as arestas sejam definidas como sendo direcionadas ou não (aresta de restrição NCM que une os vértices *samba* e *lyrics-part2*).

Várias operações sobre o grafo podem ser feitas com manipulação direta do mouse sobre os desenhos ou através de caixas de diálogo que podem ser chamadas pelo menu da visão.

Apesar da visão estrutural ilustrada na Figura 4 apresentar algumas arestas cruzando a fronteira dos vértices compostos, na estrutura de dados da visão são definidos os mapeamentos entre vértices compostos e seus componentes, preservando a composicionalidade dos documentos.

A ferramenta desenvolvida foi especializada para a autoria de documentos hipermídia, tendo por base o modelo NCM [12]. No grafo, os vértices compostos representam nós de composição NCM e os vértices atômicos os nós de mídia (áudio, imagem, texto etc). Vértices atômicos são especializados e exibidos com ícones diferentes[1]. As arestas exibidas no grafo representam elos entre os nós, contendo um conector hipermídia e a associação das interfaces dos nós aos papéis desse conector.

Para guardar as várias informações do modelo, os vértices e as arestas são estendidos com atributos, tais como âncoras, descritor para apresentação (reunindo as características de exibição) etc. Alguns desses atributos são fundamentais na construção das visões temporal e espacial.

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 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. Por exemplo, usando a ferramenta para definir arquiteturas de sistemas, os recursos podem definir processadores de um sistema distribuído, e os componentes alinhados no tempo conforme o revezamento na utilização dos recursos.

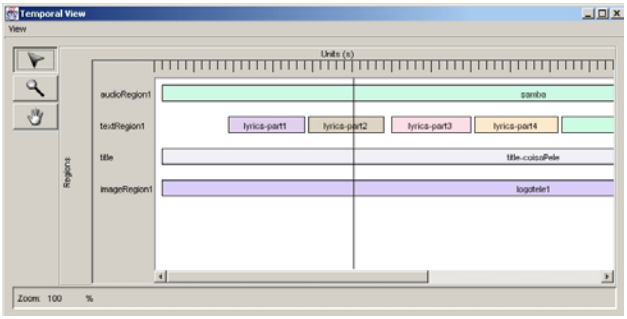

Figura 5 - Visão temporal

Para o caso particular da autoria hipermídia, os recursos representam dispositivos (regiões de janelas, placa de som etc.) utilizados pelos nós durante a apresentação. No modelo NCM e na linguagem NCL, esse relacionamento entre nós (vértices) e as regiões de exibição é realizado através dos descritores dos nós, conforme explicado anteriormente. Já a semântica das relações temporais existentes entre os nós (vértices) são dadas pelos elos através de referências a conectores. Os conectores definem as funções de cada participante (vértices) da relação e como eles interagem [11].

3.2.3 Visão Gráfica Espacial

A visão espacial possibilita ao autor determinar graficamente a disposição dos vértices de um grafo composto em relação aos recursos utilizados na arquitetura de sistema sendo especificada. A Figura 6 mostra um exemplo da visão espacial desenvolvida baseada na autoria de documentos hipermídia.

Na visão espacial ilustrada na janela superior da figura, o autor criou um *layout* de apresentação de um documento hipermídia formado por um conjunto de janelas que, por sua vez, é formado por um conjunto de regiões. É interessante destacar que a própria topologia dos recursos, permitindo a definição de aninhamentos, dá origem a um novo grafo composto na arquitetura do sistema. A diferença é que o desenho da estrutura não é livre e deve obedecer à topologia dos recursos. No caso, cada janela é um vértice composto e cada região é um

[1] A adaptação da ferramenta para o sistema HyperProp exigiu também a especialização dos vértices compostos. Foram definidos vértices compostos para estruturação do documento (exibidos no exemplo da Figura 4), para controle de versões, para definição de trilhas e para suporte à autoria cooperativa. Nesse último caso, um vértice composto importante é a *base privada* do usuário (raiz na árvore do lado esquerdo da figura). O modelo NCM define que qualquer nó que esteja contido em uma composição, deve também encontrar-se diretamente contido na base.

vértice atômico ou composto (uma vez que uma região pode ser dividida em sub-regiões).

O autor pode, através da interface gráfica, criar, editar e apagar os elementos do grafo composto, assim como alterar as posições relativas dos vértices apenas com auxílio do mouse.[2] As duas janelas inferiores à janela do editor espacial mostram a apresentação do documento hipermídia usando o *layout* especificado.

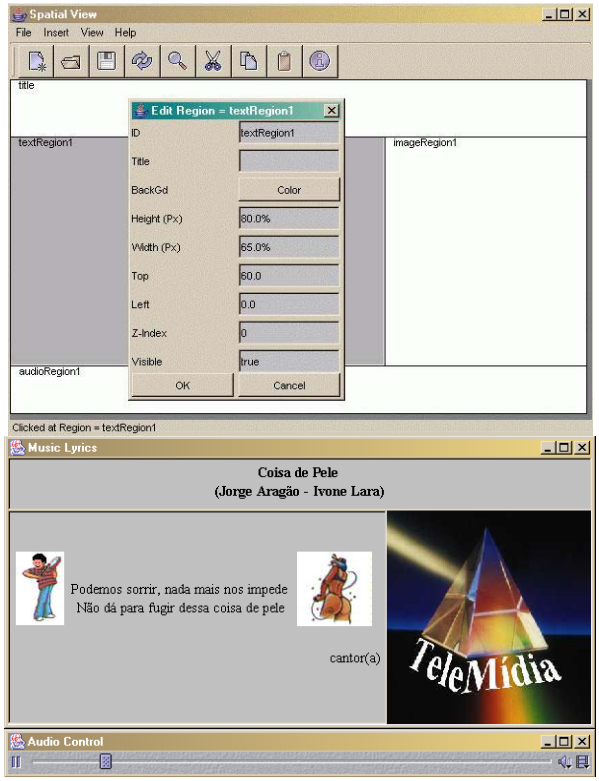

Figura 6 - Visão espacial

3.4 Sincronização entre as Visões

Para permitir o chaveamento 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 7.

O modelo utilizado para integração das visões é o DOM – *Document Object Model* [3], 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 estrutura de dados utilizada para exibição do grafo, e por converter essa estrutura de dados na representação do modelo de integração. Além dos compiladores, cada visão tem um par de observadores (um para cada compilador). Sempre que a árvore DOM é modificada, os observadores 1, na figura, são notificados e acionam os respectivos compiladores para solicitarem as atualizações dos modelos das visões. De forma análoga, os observadores 2 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).

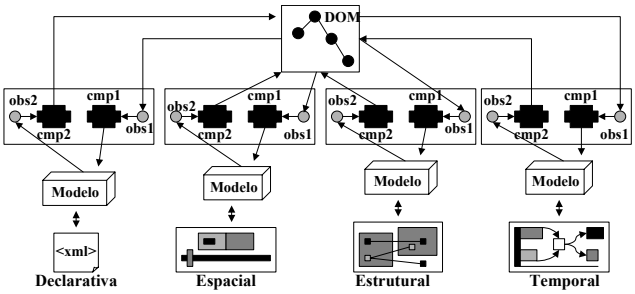

Figura 7 – Arquitetura de integração das visões

No exemplo do sistema HyperProp, os *parsers* de documentos XML foram implementados usando o pacote JAXP (*Java API for XML Processing*) [6]. Para a visão textual, os compiladores são triviais, uma vez que o modelo da visão textual é também baseado na árvore DOM de um documento NCL (Seção 2). Já para as visões gráficas, foi necessário desenvolver compiladores apropriados para tradução dos modelos [11], uma vez que suas estruturas de dados são baseadas na implementação Java do modelo NCM [12].

Os módulos que formam os compiladores são específicos para cada utilização da ferramenta. Por exemplo, para adaptar o editor para trabalhar com especificações de arquiteturas de sistemas usando os conceitos de ADL, 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 temporal, dele derivada, na visão temporal. Um outro exemplo de exibição sincronizada se dá entre as visões temporal e espacial. A visão temporal possui uma barra vertical

[2] Cabe salientar que a ferramenta espacial desenvolvida para o sistema HyperProp pode ser facilmente utilizada na especificação de *layouts* espaciais de documentos que sigam outras linguagens, como SMIL e XMT-O, uma vez que os modelos de recursos são bastante similares.

móvel (vide Figura 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. Ainda outro exemplo de visões sincronizadas é discutido na próxima seção, quando filtragens na exibição de vértices em uma visão implicam em filtragens nas demais visões.

4. Técnicas de Filtragens Aplicadas às Visões da Ferramenta de Edição

Um problema enfrentado pelos usuários que trabalham com estruturas de dados grandes é a desorientação na busca por determinada informação. Em grafos compostos, essa desorientação é causada pelo número elevado de vértices e arestas.

Dentre as técnicas mais comuns para a apresentação das estruturas, podem ser citadas o uso de recursos de *zoom* e *scroll* em partes da estrutura. Entretanto, a técnica que se mostra mais eficaz, embora mais complexa, é a filtragem de partes da estrutura não relevantes para o usuário. A dificuldade é identificar quais partes da estrutura realmente são relevantes.

A visão olho-de-peixe, originalmente proposta por Furnas [5] para estruturas hierárquicas, atua como uma lente, preservando os detalhes próximos a um ponto escolhido (vértice em foco) e, à medida que se afasta desse ponto, exibindo menos informação (apenas as mais importantes segundo critérios que serão explicados no próximo parágrafo). Em [8], o método é estendido para grafos compostos.

A estratégia básica do olho-de-peixe é definir uma função de grau de interesse, que atribui a cada vértice do grafo um valor, representando o grau de interesse do usuário em relação ao vértice em foco. A idéia principal é que essa função (DOI) aumente com a importância a priori (API) e diminua com a distância (D), por arestas e aninhamento, ao foco. Assim tem-se: DOI(x,y) = API(x) – D(x,y), onde DOI é o grau de interesse do usuário para o elemento x em relação ao elemento em foco y.

A filtragem dos vértices é realizada escolhendo um determinado valor K para o corte, de modo a exibir somente os elementos x que possuírem um DOI(x,y) ≥ K. Variando o valor de K, que pode ser interpretado como o nível de detalhe desejado, obtém-se diferentes visões olho-de-peixe para o mesmo grafo. O editor descrito neste artigo possui o algoritmo olho-de-peixe implementado em todas as visões, podendo o usuário ativar ou desativar a filtragem a qualquer momento. Ao selecionar um foco numa determinada visão, ou um novo ajuste no parâmetro K (grau de interesse), os elementos pertencentes à visão são automaticamente filtrados. O sistema também permite que o usuário escolha estabelecer um ponto de corte unificado para as visões, ou pontos de corte diferenciados. Nesses casos, o filtro realizado em uma visão reflete em todas as outras. de forma sincronizada. A Figura 8 mostra a visão espacial com o filtro olho-de-peixe aplicado (janela superior) e com a filtragem refletida na visão declarativa (janela inferior).

[8] apresenta maiores detalhes da técnica de filtragem olho-de-peixe em grafos compostos.

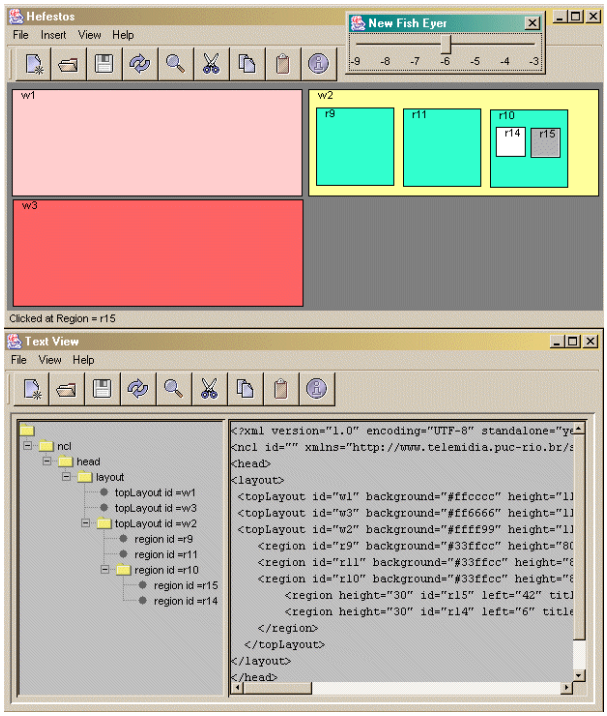

Figura 8 - Olho-de-peixe nas visões espacial e declarativa

5. Trabalhos Relacionados

A ferramenta Salix [2] trata especificamente da autoria de documentos XML. Salix apresenta um depurador de documentos XML, que permite ao usuário localizar o erro depois da validação do documento XML, fazendo uso das DTDs associadas ao arquivo. A mesma idéia foi aplicada ao sistema HyperProp, no qual um depurador da linguagem NCL informa ao usuário os números das linhas onde os erros foram localizados no arquivo NCL. Além disso, a visão declarativa do sistema pode ser diretamente utilizada na edição de arquivos especificados em outras linguagens XML. A facilidade de depuração, no entanto, precisa ser adaptada pelo programador da ferramenta, em função da especificação da linguagem (DTD, Schemas etc.).

Kaomi é um *toolkit* para projetar ambientes de autoria de documentos multimídia [7]. O sistema é baseado em três visões gráficas (apresentação, estrutural e temporal)

e uma declarativa do documento XML, que são sincronizadas entre si pela seleção de um objeto. Para a utilização do Kaomi, o projetista do ambiente deve fornecer um compilador entre a linguagem utilizada pelo documento (XML) e a estrutura de dados utilizada pelo sistema, analogamente ao que foi proposto no ambiente descrito neste artigo. Apesar de similares, o Kaomi é voltado para autoria de documentos multimídia e não de grafos compostos de uma forma geral. Além disso, o *toolkit* não menciona as facilidades de filtragem descritas neste artigo.

O GRiNS ProEditor[3] é um ambiente para autoria de documentos hipermídia que enfatiza o uso da visão temporal, embora disponibilize ao usuário outras visões, como a espacial, de apresentação e textual. No entanto, a ferramenta é especificamente voltada para autoria de documentos SMIL e também não oferece as facilidades de filtragem dos documentos.

6. Conclusões e Trabalhos Futuros

Este artigo discute as características de um editor para arquiteturas de sistemas modeladas através de grafos compostos com suporte à composicionalidade. O sistema é baseado na integração de quatro visões que se complementam: declarativa (ou textual), estrutural, temporal e espacial. Para auxiliar autores na construção de grafos maiores, técnicas de filtragem são integradas ao sistema.

Embora voltado para autoria, as ferramentas implementadas podem ser úteis também no auxílio à execução da própria arquitetura. No caso de sistemas hipermídia, por exemplo, as visões estrutural e temporal, juntamente com os mecanismos de filtragem, podem ser úteis na navegação de um leitor através de um documento mais complexo (com muitos nós e elos).

Uma adaptação do editor foi feita para a implementação de um ambiente de autoria de documentos hipermídia. Nesse sentido, a linguagem declarativa utilizada para autoria é a NCL e o modelo conceitual utilizado para representação dos documentos o NCM.

A linguagem NCL define um módulo, denominado XTemplate, para especificação de *templates* de composições hipermídia [11]. *Templates* permitem testar restrições e aplicar transformações sobre o conteúdos das composições. Embora orientada à autoria de hiperdocumentos, pretende-se, como trabalho futuro, investigar o uso de extensões do módulo XTemplate para a definição de *templates* de vértices compostos de um modo geral.

Outro trabalho futuro diz respeito à possibilidade do usuário editar graficamente os mapeamentos entre vértices compostos e seus componentes. Atualmente, essa edição só pode ser feita através das caixas de diálogo das ferramentas de edição.

Por fim, pretende-se explorar a generalidade do editor para a implementação de ferramentas de autoria de grafos voltados para outros domínios, como os mencionados no início deste artigo. Pretende-se também tornar disponíveis as bibliotecas e APIs compondo as várias ferramentas, para que terceiros possam construir aplicações que façam uso, ou estendam, o editor genérico descrito neste trabalho.

Referências

[1] Allen R. J. "A Formal Approach to Software Architecture". Tese de Doutorado, Carnegie Mellon University, EUA, 1997.

[2] Aoki E. H, Nakasone T. L., Seraphim E. "Um ambiente de autoria de documentos XML". IX Simpósio Brasileiro de Sistemas Multimídia e Hipermídia – WEBMídia03, Salvador, Brasil, 2003.

[3] "Document Object Model (DOM) Level 3", W3C Recommendation, Abril 2004.

[4] Felix M. F, "Análise formal de modelos de software orientada por abstrações arquiteturais". Tese de Doutorado, PUC-Rio, Brasil, 2004.

[5] Furnas G. "Generalized Fisheye Views". Proceedings of ACM SIGCHI'86 Conference on Human Factors in Computing Systems, Boston, 1986.

[6] JAXP – Java API for XML Processing. http://java.sun.com/xml/jaxp.

[7] Jourdan M., Roisin C., Tardif L. "A Scalable Toolkit for Designing Multimedia Authoring Environments". Multimedia Tools and Applications Journal, Kluwer Academic Publishers, 1999.

[8] Muchaluat D.C., Rodrigues R.F., Soares, L.F.G. "WWW Fisheye View Graphical Browsers". V Multimedia Modeling Conference, Lausanne, 1998.

[9] Noik E. G., "Layout-independent Fisheye Views of Nested Graphs", IEEE Symp. Visual Languages, 1993.

[10] Shneiderman B. "Designing the user interface: strategies for human-computer interaction", Reading, Addison-Wesley, 3 ed. 1998.

[11] Silva H.V., Muchaluat-Saade D.C., Rodrigues R.F., Soares L.F.G. "NCL 2.0: Integrating New Concepts to XML Modular Languages", ACM Symposium on Document Engineering, Milwaukee, 2004.

[12] Soares L.F.G., Rodrigues R.F., Muchaluat-Saade D.C. "Modeling, Authoring and Formatting Hypermedia Documents in the HyperProp System", ACM Multimedia Systems Journal, 8(2), 2000.

[13] "Synchronized Multimedia Integration Language (SMIL 2.0)", W3C Recommendation, 2001.

[14] "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, 2000.

[3] http://www.oratrix.com