2
Conceitos Preliminares

O uso de grafos para modelagem de problemas encontra-se na literatura clássica desde a época de Leonhard Euler (1707-1783), quando o mesmo solucionou o conhecido problema das “Pontes de Königsberg” (Cormen et al., 2001).

A modelagem de problemas em grafos pode ser bastante útil, pois através delas é possível formalizar diferentes topologias em uma linguagem de representação padrão, para a qual já existem diversos teoremas que podem auxiliar na solução dos problemas. Além disso, sua representação gráfica pode ajudar na análise do problema. Porém, nem sempre essa visualização é plausível de ser utilizada, devido à sua dimensão e à sua complexidade. Conforme comentado no Capítulo 1, nesses casos, técnicas de filtragem podem ser utilizadas para reduzir tal complexidade.

Grafos podem ser usados em diversas áreas do conhecimento. Certos sistemas, como redes de computadores, sistemas hipermídia, mapas geográficos, arquiteturas de software, entre outros, apresentam características de estruturação que podem ser representadas através de grafos compostos.

Este capítulo está organizado da seguinte forma. A Seção 2.1 apresenta a definição de grafos compostos. A Seção 2.2 descreve algumas arquiteturas de sistemas modeladas através de grafos compostos. Por fim, na Seção 2.3, o conceito de grafos com composicionalidade é analisado em modelos hipermídia, quando então são resumidamente apresentados o modelo conceitual NCM – *Nested Context Model* (Soares et al., 2003) e a linguagem NCL – *Nested Context Language* (Muchaluat-Saade et al., 2003).

2.1.
Definição de Grafos Compostos

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 entre dois elementos de *V* (Noik, 1993). 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 um subgrafo de *G*. Os vértices pertencentes a *AV* são vértices que não podem conter outros vértices, sendo, por isso, chamados de vértices atômicos.

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

$V = CV \cup AV$
$A \subseteq (V \times V)$
$I \subset CV \times V$ é um conjunto de relações 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[1] que o contenha (direta ou recursivamente); e um mesmo vértice do grafo não pode estar diretamente contido em mais de um vértice composto (Noik, 1993).

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 desta dissertação, um *grafo com composicionalidade*, um grafo composto em que o conjunto de arestas *A* é formado pela união dos conjuntos *B* e *M*, tais que:

a) $B \subseteq (V \times V)$ e, para qualquer par de vértices (*v*,*c*) pertencentes a *B*, ou $\exists\ cv \in CV$ tal que $v \in cv$ e $c \in cv$, ou $\forall\ cv \in CV$, $v \notin cv$ e $c \notin cv$. 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.

[1] Diz-se que um vértice composto *c* contém um vértice *v* recursivamente, se existe ao menos um vértice composto *r* contido em *c* tal que *r* contém *v* diretamente ou contém *v* recursivamente.

b) M $\subseteq$ (CV x V) e, para qualquer par de vértices (*cv*,*v*) pertencentes a *M*, $v \in cv$. Em outras palavras, *M* constitui um conjunto de arestas de pais para filhos, sendo, portanto, um subconjunto de *I*.

A Figura 2-1 ilustra um grafo composto com composicionalidade. Nessa 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* (an_1 e c_2, por exemplo).

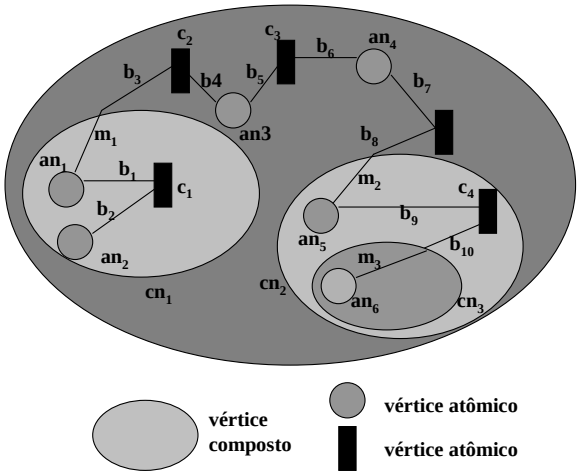

Figura 2-1 Grafo composto com composicionalidade

2.2. Sistemas Modelados através de Grafos Compostos

Conforme já comentado no Capítulo 1 e no início deste capítulo, a estrutura de dados de um grafo composto pode ser usada na modelagem de sistemas que apresentam o conceito de composição. Mais ainda, quando esses sistemas oferecem composicionalidade, grafos com composicionalidade podem ser aplicados na modelagem. Nas próximas subseções, alguns sistemas são descritos como exemplos.

2.2.1. ADLs (*Architecture Description Languages*)

Linguagens de descrição de arquitetura (ADLs) são linguagens formais que podem ser usadas para representar a arquitetura de um sistema de software, definindo seus componentes, como eles se comportam e os padrões e mecanismos para interações entre esses componentes (Clements, 1996).

Uma ADL pode também ser entendida como uma linguagem voltada para especificar a estrutura de alto nível de uma aplicação (ou seja, a arquitetura conceitual), ao invés de se preocupar com detalhes de implementação de um módulo de código específico.

Uma descrição arquitetural é composta de elementos básicos (ou blocos principais), que são os componentes, conectores e configurações arquiteturais.

- componentes são unidades de computação ou armazenamento de dados que, dependendo do nível de abstração, podem ser tão pequenos quanto um único procedimento ou tão grandes como uma aplicação inteira;
- conectores são usados para modelar interações entre componentes e definir as regras que governam essas interações;
- configurações arquiteturais são grafos conexos de componentes e conectores que descrevem a estrutura da arquitetura.

Diversas propostas de ADLs podem ser encontradas na literatura, entre elas ACME (Garlan et al., 1997), Aesop (Garlan, 1995), Armani (Monroe, 1999), CL (Paula, 1999), Darwin (Magee et al., 1996) e Wright (Allen, 1997).

Muitas ADLs admitem a existência de elementos (componentes ou conectores) compostos e garantem a composicionalidade da arquitetura. Assim, uma configuração em ADL pode ser representada por um grafo com composicionalidade, onde os vértices do grafo representam componentes e conectores. Pela definição da Seção 2.1, as arestas de *B* representam as ligações (*binds*) entre portas de componentes e papéis de conectores (seguindo a terminologia da ADL *Wright* (Allen, 1997)). 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*). A Figura 2-2 apresenta uma arquitetura de ADL representada como um grafo composto com composicionalidade.

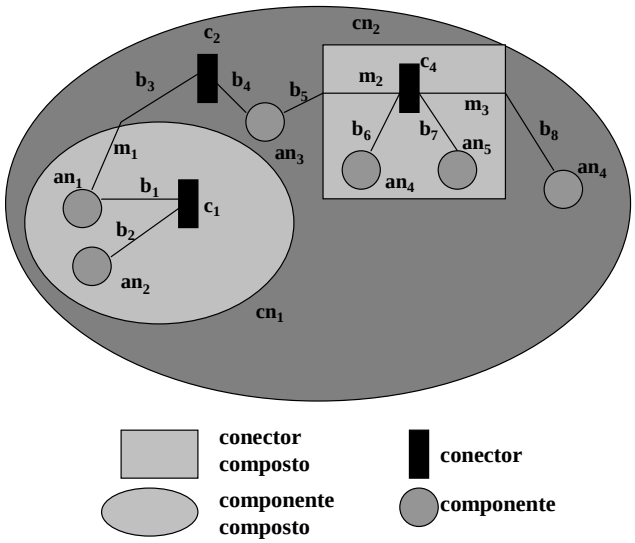

Figura 2-2 Representação de uma arquitetura de software descrita através de uma ADL utilizando grafos compostos

Uma forma de descrever ADLs textualmente é através da linguagem xADL (Dashofy et al., 2001), conforme explicado a seguir.

2.2.1.1. xADL - *XML-Based Architecture Description Language*

Em (Dashofy et al., 2001), é apresentada uma linguagem para a descrição de arquiteturas, denominada xADL - *XML-Based Architecture Description Language*. A linguagem xADL foi desenvolvida na Universidade da Califórnia e é extensível, pois todos os seus elementos e atributos são mapeados em módulos individuais, implementados através de XML *Schema* (XML, 2001), padrão W3C. A xADL possibilita, entre outras facilidades, a especificação dos relacionamentos dos elementos que compõem uma arquitetura.

A linguagem xADL adotou as entidades tradicionais, encontradas nas principais ADLs, para sua constituição, sendo formada por componentes, conectores, *links*,

interfaces (pontos de entrada e saídas para conectores e componentes) e configurações (topologia da organização dos componentes e conectores).

Os componentes em xADL são identificados pelo elemento "*component*". Cada componente pode conter: *i)* interfaces, representadas pelo elemento "*interface*"; *ii)* mapeamentos, representados pelo elemento "*signatureInterfaceMapping*"; *iii)* conectores, representados pelo elemento "*conector*"; e *iv)* outros componentes.

As interfaces podem ser de três tipos: *i)* "*in*", fluxo de comunicação apenas de entrada; *ii)* "*out*", fluxo de comunicação apenas de saída ; e *iii)* "*inout*", para representar tanto fluxo de saída como de entrada.

O elemento "*signatureInterfaceMapping*" (mapeamento) é constituído por dois outros elementos internos a ele: "*outerSignature*", para referenciar uma das interfaces do elemento no qual o mapeamento está sendo definido (vértice composto) e "*innerInterface*", para referenciar uma interface de um dos elementos filhos no qual o mapeamento está sendo definido.

O elemento "*link*" relaciona elementos (componentes e conectores) que tenham o mesmo pai ou dois elementos independentes (ausência de pai), respeitando assim o conceito de composicionalidade. Os "*links*" são formados por um conjunto de elementos "*point*", que referenciam interfaces de componentes ou interfaces de conectores.

Os conectores, assim como os componentes, são constituídos por um conjunto de interfaces, mapeamentos, componentes e conectores internos a ele. É importante no entanto destacar que, em xADL, conectores só exportam interfaces de conectores filhos e componentes só exportam interfaces de componentes filhos.

A Figura 2-3 apresenta um documento HTML - *HyperText Markup Language* (HTML, 1999) com seu respectivo código mapeado em xADL na Figura 2-4.

```
<html>
  <head>
    <script language="JavaScript">
        function MsgBox (textstring) { alert (textstring) }
    </script>
  </head>
  <body>
    <form>
      <input name="text1" type= “text”>
      <input name="Go!" type= “button” value="Go!" onClick="MsgBox (form.text1.value)">
```

```
    </form>
  </body> </html>
```

Figura 2-3 Documento HTML

```
<component id = "headComp" type = "Component">
    <description type = "Description">Head</description>
    <interface id = "headCompRight" type = "Interface">
        <description type = "Description">Head.Right </description>
        <direction type = "Direction"> inout </direction>
    </interface>
    <component id = "javaComp" type = "Component">
        <description type = "Description">function</description>
        <interface id = "javaCompRight" type = "Interface">
            <description type = "Description">function.Right </description>
            <direction type = "Direction" >in</direction>
        </interface>
        <interface id = "javaCompLeft" type = "Interface">
            <description type = "Description">function.Left</description>
                <direction type = "Direction" >out</direction>
        </interface>
    </component>
    <signatureInterfaceMapping>
        <outerSignature href = "headCompRight"/>
        <innerInterface href= "javaCompLeft" />
        <outerSignature href = "headCompRight"/>
        <innerInterface href= "javaCompRight" />
     </signatureInterfaceMapping>
</component>
<component id = "bodyComp" type = "Component">
    <description type = "Description">Body</description>
    <interface id = "bodyCompLeft" type = "Interface">
        <description type = "Description"> Body.Left </description>
        <direction type = "Direction" >inout </direction>
    </interface>
</component>
<connector id = "con1" type = "Connector">
    <description type = "Description" >Connector</description>
```

```
        <interface id = "con1Left" type = "Interface">
            <description type = "Description">Con1.Left</description>
            <direction type = "Direction:>inout</direction>
        </interface>
        <interface id = "con1Right" type = "Interface">
            <description type = "Description">Con1.Right</description>
            <direction type = "Direction:>inout</direction>
        </interface>
</connector>
<link id = "link1" type = "Link">
    <description type = "Description">Head-Con1</description>
    <point type = "Point">
        <anchorOnInterface href = "con1Left" type = "XMLLink" />
    </point>
    <point type = "Point">
        <anchorOnInterface href = "headCompRight" type = "XMLLink"/>
    </point>
</link>
<link id = "link2" type = "Link">
    <description type = "Description">Body-Con1</description>
    <point type = "Point">
        <anchorOnInterface href = "con1Right" type = "XMLLink"/>
    </point>
    <point type = "Point">
        <anchorOnInterface href = "bodyCompLeft" type = "XMLLink"/>
    </point>
</link>
```

Figura 2-4 Documento xADL

Inicialmente é criado um componente "*headComp*", para representar o elemento "*head*" da página HTML, contendo uma interface "*headCompRight*" do tipo "*inout*" (o fluxo pode ser de entrada e saída), um componente filho denominado de "*javaComp*", e mapeamentos "*signatureInterfaceMapping*" relacionando a interface de *headComp* com as interfaces de *javaComp*.

O componente "*javaComp*" representa o elemento "*script*" da página HTML e contém duas interfaces: "*javaCompRight*" do tipo "*in*" e "*javaCompLeft*" do tipo "*out*".

Os mapeamentos entre os componentes "*headComp*" (vértice composto) e "*javaComp*" (vértice atômico interno a "*headComp*") são feitos através do elemento "*signatureInterfaceMapping*", onde a interface do elemento externo é referenciada no elemento "*outerSignature*", e a interface do elemento interno é referenciada através do elemento "*innerInterface*".

O componente "*bodyComp*" (vértice atômico), representado pelo elemento "*body*" da página HTML, é constituído pela interface "*bodyCompLeft*" do tipo "*inout*".

O conector "*con1*" foi criado para possibilitar a comunicação entre dois componentes. Para isso, duas interfaces do tipo "*inout*" foram criadas no conector: "*con1Left*" e "*con1Right*". Nesse documento xADL, a finalidade do conector é permitir a comunicação entre os componentes "*bodyComp*" e "*headComp*".

Os "*links*" fazem associações entre a interface de um conector e a interface de um componente. É importante destacar que quando um "*link*" referencia a interface de um nó, e este nó apresenta mapeamentos exportando interfaces de seus elementos internos, o "*link*" juntamente com esses mapeamentos permitem uma conexão entre elementos de níveis diferentes, respeitando assim o conceito de composicionalidade.

2.2.2. Forma

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 na subseção anterior) e os modelos hipermídia (expostos na Seção 2.3). Forma (Felix, 2004), 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 utilizando a definição da Seção 2.1, 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*). A Figura 2-5 apresenta uma arquitetura segundo Felix (Felix, 2004).

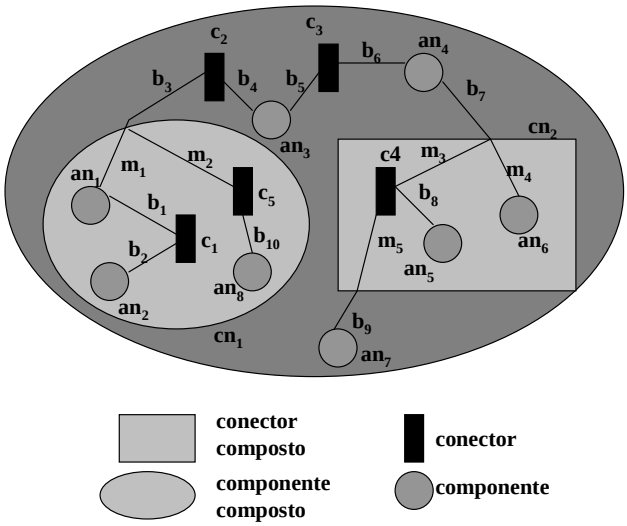

Figura 2-5 Arquitetura de forma em grafos compostos

2.2.3. Workflow

De acordo com Hollingsworth (Hollingsworth, 2004), *Workflow* é a automação de um processo de negócio, em parte ou como um todo, onde documentos, informações e tarefas são passadas de um participante ao outro, de acordo com um conjunto de regras definidas.

Em outras palavras, *Workflow* é fluxo de trabalho que descreve o que deve ser executado e como as atividades são encadeadas. As características de um *Workflow* são, basicamente, possuir um fluxo de atividades com um início e um fim definidos e, em alguns casos, envolver processos repetitivos.

O fluxo de trabalho é composto de atividades encadeadas e baseado em regras de processo de negócio. Conforme as atividades são realizadas, regras de negócio são aplicadas sobre essas atividades, direcionando-as para cada grupo de trabalho com base nas regras pré-determinadas, ou seja, na lógica de processo. As regras são atributos que definem de que forma os dados que trafegam pelo fluxo de trabalho devem ser transformados, direcionados e monitorados.

Segundo (Cruz, 1998), rotas são caminhos lógicos que, definidas sobre regras específicas, têm a função de transferir a informação dentro do processo, ligando as atividades do fluxo de trabalho. Os tipos de rotas são:

- serial - a atividade possui apenas uma atividade posterior. Cada atividade deve ser completada para ocorrer o início da seguinte;
- paralelo - grupo de atividades que ocorrem ao mesmo tempo onde as atividades são independentes entre si. As atividades de um grupo paralelo possuem a mesma atividade anterior que dispara o início das mesmas. Além disso, as atividades de um grupo paralelo possuem a mesma atividade posterior que aglutina os seus finais; e
- caminho condicional: a escolha de uma rota é determinada por uma regra.

A representação gráfica de um *Workflow* pode ser mapeada em um grafo composto, onde cada grupo de atividades pode ser representado como um vértice (composto) e os relacionamentos entre eles podem ser representados como arestas do grafo (*binds* e mapeamentos).

A Figura 2-6 ilustra a representação em grafo de um *Workflow* onde podem-se perceber seis vértices atômicos (an_x) e um vértice composto (cn_1), com lógica paralela na execução de três atividades. Nesse caso, as atividades são modeladas como vértices atômicos e os grupos de atividades como vértices compostos.

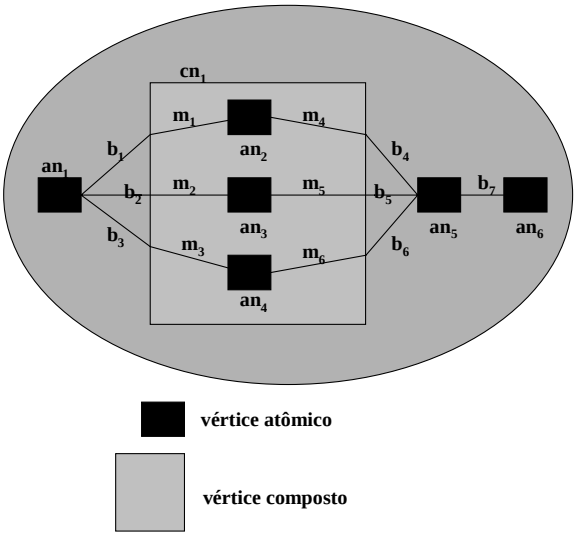

Figura 2-6 *Workflow* mapeado em grafos compostos

Existem diversas linguagens baseadas em XML para representação de sistemas de gerência de *Workflow*. Três dessas propostas vêm da indústria: XPDL, Wf-XML e WSFL. Outras são acadêmicas, tais como: XRL e WQM (Zschornack, 2003).

Na próxima subseção, a linguagem XRL é explicada como exemplo de representação declarativa de *Workflow*.

2.2.3.1. XRL - *Exchangeable Routing Language*

A linguagem XRL - *Exchangeable Routing Language* - (Aslst & Kumar, 2000), criada na Universidade do Colorado, tem seu formalismo semântico baseado nas Redes de Petri.

XRL possui elementos definidos na linguagem com semânticas específicas para a definição dos trajetos a serem percorridos para a conclusão de um determinado processo, sendo eles:

- tarefa (vértice atômico): ação a ser realizada, possuindo uma série de atributos, como nome, documentos, tempo de início e fim da tarefa etc. O elemento tarefa é representado na linguagem através da *tag* "*task"*;
- seqüência (vértice composto): os elementos internos ao elemento seqüência devem ser executados na ordem em que estão dispostos no documento, ou seja, os primeiros elementos definidos deverão ser os primeiros a serem processados, O elemento seqüência é representado na linguagem através da *tag* "*sequence"*;
- paralelismo (vértice composto): os elementos definidos dentro de um elemento paralelo **são iniciados** ao mesmo tempo. Três tipos de paralelismo são definidos na linguagem com relação **ao término** do processamento dos elementos: *i)* paralelismo com sincronismo total: todos os elementos em paralelo devem terminar suas execuções para que o fluxo continue, representado na linguagem pela *tag* "*Parallel_sync"*; *ii)* paralelismo com sincronismo parcial: o fluxo pode continuar quando alguns elementos terminam seus processamentos, representado na linguagem pela *tag* "*Parallel_part_sync"*; e *iii)* paralelismo sem sincronismo: não existe espera

entre o término de processamentos dos elementos para que o fluxo continue, basta que um elemento termine a tarefa para o fluxo continuar seu processamento, representado na linguagem pela *tag* "*Parallel_no_sync"*;

- condicional (vértice composto): somente os elementos que satisfizerem determinada condição serão executados, representado na linguagem pelo elemento "*condition*"; e
- repetição (vértice composto): construção que permite a execução repetida de seus elementos internos. Nesse caso, uma condição para o laço deve ser testada, representada na linguagem pela *tag* "*repetition*".

A fim de explicar a funcionalidade dos principais elementos presentes na linguagem XRL, a Figura 2-7 ilustra um exemplo de uso da linguagem.

```
<route>
 <sequence>
   <task name = "an1" … >
       <parallel_sync name = "cn1" …>
     <task name = "b" name = "an2"… >
      <task name = "c" name = "an3" …>
     <task name = "d" name = "an4"… >
    </parallel_sync>
     <task name = "an5" … >
     <task name = "an6" … >
  </sequence>
</route>
```

Figura 2-7 Um exemplo de W*orkflow* em XRL

2.3. Modelos Hipermídia

De forma análoga às ADLs e às ferramentas de especificação formal, alguns modelos hipermídia preservam a noção de composicionalidade, embora quase sempre limitada a apenas uma de suas entidades (um subconjunto de seus vértices): os nós. Mesmo modelos bastante semelhantes às ADLs, como o modelo NCM (*Nested Context Model*) - e conseqüentemente a linguagem NCL (*Nested Context Language*) (Muchaluat-

Saade et al., 2003) baseada nesse modelo - 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 (SMIL, 2001), 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. Essa exportação é realizada através de mapeamentos (representados pelas arestas de *M*, conforme as definições da Seção 2.1), mantendo assim a propriedade de composicionalidade para o documento. 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). A Figura 2-8 apresenta um exemplo de documento, seguindo o modelo NCM, descrito através de um 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.

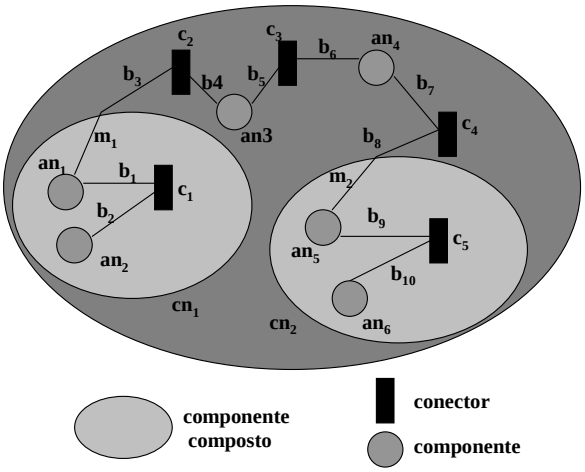

Figura 2-8 Modelo NCM representado em grafos compostos

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. A Figura 2-9 ilustra o tratamento de elos multipontos sem conectores em grafos compostos.

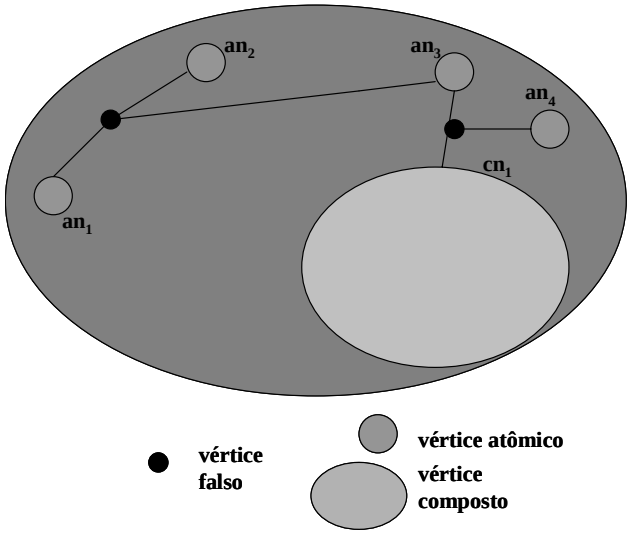

Figura 2-9 Elo multiponto em grafos compostos

Conforme já mencionado no Capítulo 1, os exemplos de documentos hipermídia usados nesta dissertação são, em sua maioria, baseados no modelo NCM e na linguagem NCL (focos iniciais da ferramenta apresentada nesta dissertação). Sendo assim, as próximas duas seções descrevem resumidamente esses conceitos (modelo e linguagem), a fim de oferecer os subsídios básicos para a leitura do restante da dissertação. Maiores

detalhes do modelo (Soares et al., 2003) e da linguagem (Muchaluat-Saade, 2003) podem ser obtidos nas referências citadas.

2.3.1. Modelo Conceitual NCM (*Nested Context Model*)

O modelo conceitual NCM, *Nested Context Model* (Soares et al., 2003), representa um documento hipermídia através de um nó. O nó NCM pode ser um objeto de mídia (nó terminal ou de conteúdo) ou um nó de composição. Um nó de composição contém um conjunto de nós, podendo esses nós internos serem objetos de mídia (nós terminais) ou outros nós de composição, recursivamente. Os nós de composição NCM podem conter também elos relacionando esses nós. Uma restrição do modelo impede que um nó contenha, recursivamente, a si próprio, similar à restrição de grafos compostos apresentada no início deste capítulo.

Para permitir o relacionamento entre partes internas do conteúdo de um nó, o modelo define pontos de interface, que podem, por sua vez, ser uma âncora ou uma porta. Âncoras são pontos de interface que podem ser definidos para qualquer tipo de nó, representando intuitivamente um subconjunto marcado de unidades de informação do conteúdo. Portas são pontos de interface que só podem existir em nós de composição. Uma porta de um nó de composição especifica um mapeamento para um ponto de interface de um dos nós internos da composição.

Os tipos de âncoras variam de acordo com o tipo de mídia do nó. No caso de um nó de texto, uma âncora poderia ser uma palavra, já em um nó de áudio contendo uma música poderia ser um determinado intervalo de tempo da música. Uma âncora também pode referenciar um atributo do nó, permitindo que relacionamentos também sejam estabelecidos com base nas características do nó.

Para estabelecer relacionamentos entre nós, é necessária a criação de elos que são agrupados nas bases de elos pertencentes aos nós de composições, conforme comentado anteriormente.

Um elo faz referência a um conector hipermídia (Muchaluat-Saade, 2003) e a um conjunto de associações (*binds*). Um conector representa uma relação sem definir quais nós fazem parte do relacionamento. As relações podem ser de vários tipos, por exemplo:

relações de sincronização, referência, derivação, relações entre tarefas de um trabalho cooperativo etc. O conector exporta, através de *papéis*, as interfaces para que os objetos tomem partido na relação, identificando, como o próprio nome sugere, o papel de cada objeto no relacionamento. Dessa forma, os *binds* têm como finalidade associar papéis de um conector a pontos de interface de nós do documento.

A especificação de apresentação de um nó no modelo NCM é feita independente da definição do nó, sendo representada por outro objeto do modelo, chamado descritor (Soares et al., 2000) (Rodrigues, 2003). Os descritores reúnem as informações referentes às características de exibição dos objetos do documento. Os principais atributos de um descritor são: *i)* ferramenta utilizada na exibição do nó; *ii)* região espacial de apresentação, que identifica onde o objeto vai ser exibido; *iii)* informações para ajuste temporal da apresentação do objeto; *iv)* parâmetros diversos que possam ser úteis para caracterizar a exibição (por exemplo, volume de um áudio, sotaque a ser aplicado a uma voz sintetizada etc.).

2.3.2. Linguagem NCL (Nested Context Language)

A NCL (*Nested Context Language*), atualmente na versão 2.0 (Muchaluat-Saade, 2003), é uma linguagem declarativa para especificação de documentos hipermídia.

Devido ao fato de a linguagem estar baseada no modelo NCM, suas características são similares às do modelo, tais como: o uso de composições (nós de composição) para a estruturação lógica do documento, especificação de relacionamentos temporais através de elos e a especificação de relacionamentos espaciais por meio de descritores.

Além disso, a linguagem introduz o conceito de *template* de composição hipermídia (Muchaluat-Saade, 2003), que tem como objetivo facilitar a autoria de documentos NCL, pois possibilita ao autor reutilizar uma estrutura já definida anteriormente. Dessa forma, tipos pré-definidos de composições podem ser tratados como *templates* NCL.

Vale destacar que a linguagem NCL 2.0 foi especificada através de XML *Schema* (XML, 2001) e dividida em módulos, assim como a linguagem xADL e a linguagem SMIL em sua versão 2.0. Tal abordagem facilita a interoperabilidade entre linguagens, permitindo que módulos de uma linguagem sejam incorporados a outra. Por exemplo, o

módulo *BasicMedia* da linguagem SMIL é diretamente utilizado pela linguagem NCL. Esse módulo tem como finalidade definir os tipos básicos de mídia que existem em um documento.

A Figura 2-10 ilustra o código de um documento NCL para um melhor entendimento da linguagem.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ncl id="foodNcl" ...>
 <head>
  <layout>
        <topLayout id="w1" ...>
          <region id="image" top=”0” left=”0” .../>
         ...
        </topLayout>
  </layout>
  <descriptorBase>
   <descriptor id=" imagtextD2" region=" image "/>
   ...
  </descriptorBase>
 </head>
 <body>
    <composition id="food">
      <port id="entradaFood" .../>
        <video descriptor="imagtextD2" id="foodVideo" ...">
        <audio descriptor="audio" id="ingredientesAud" ...">
          <area id="part01Aud" begin="2.0s" end="3.2s"/>
           ...
        </audio>
        <img descriptor="logoDescriptor" id="logoTelemidia" .../>
        ...
        <text descriptor="textD1" id="ovoTxt" .../>
        ...
        <linkBase>
          <link id="link01-01" xconnector="starts.xml" >
            <bind component="ingredientesAud" role="on_x_presentation_begin" />
            <bind component="logoTelemidia" role="start_y"/>
          </link>
           ...
        </linkBase>
   </composition>
 </body>
</ncl>
```

Figura 2-10 Exemplo de documento NCL 2.0

A linguagem NCL 2.0 oferece um elemento *composition*, que permite criar nós de composição (representados por vértices compostos em uma modelagem em grafo). As composições podem conter nós de mídia (vértices atômicos), como por exemplo textos (*text*), imagens (*img*), vídeos (*video*), áudio (*audio*), outros nós de composição (*composition*) e bases de elos (*linkBase*).

As bases de elos contêm os elos (*link*), que, por sua vez, são formados por conectores, identificados na linguagem pelo atributo *xconnector* do elemento *link*, e pela associação entre nós e conectores (*bind*). Cabe comentar que, quando referenciado dessa forma, o conector encontra-se descrito em um outro arquivo.

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 (*area*) dos nós de mídia, ou para portas de nós de composição internos.

Para manter a composicionalidade, as associações dos elos (*binds*) devem referenciar exclusivamente portas/âncoras de nós diretamente contidos na composição que contém o elo.

A linguagem NCL também define as características de apresentação dos nós em uma entidade separada, representada pelo elemento *descriptor*. O *descriptor* pode especificar tanto os parâmetros temporais de apresentação como o posicionamento espacial do nó, por exemplo associando um nó a um dispositivo de saída (*region)*.

Com base nos elementos da linguagem NCL definidos anteriormente, as visões temporal, espacial e estrutural dos documento, mencionadas no Capítulo 1, podem ser geradas.

Cabe comentar que, para cada visão gráfica, diferentes elementos da linguagem NCL ganham destaque. Por exemplo, os elementos *topLayout* (conjunto de *regions*), *region* e *descriptor* são os elementos fundamentais utilizados na representação espacial do documento. Já na visão temporal, os elos juntamente com as âncoras presentes nos nós de mídias (por exemplo, nó de áudio) são entidades-chave para a visualização do documento no eixo do tempo. No caso da visão estrutural, os nós de composição ganham destaque, pois são responsáveis pelo agrupamento lógico das entidades presentes no documento.