2
Linguagens de Autoria Hipermídia e Linguagens de Descrição de Arquitetura de Software

Linguagens de descrição de arquitetura (***A**rchitecture **D**escription **L**anguage* – **ADL**) 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 (Shaw, 1996b).

Shaw e Garlan (Shaw, 1996b) apresentam seis propriedades que caracterizam o que uma linguagem de descrição de arquitetura ideal deveria fornecer:

- Composição – possibilidade de descrever um sistema como uma composição de elementos independentes.
- Abstração – possibilidade de descrever os elementos de uma arquitetura de software de tal forma que seus papéis abstratos em um sistema sejam prescritos claramente e explicitamente.
- Reuso – possibilidade de reutilizar elementos em descrições arquiteturais diferentes, mesmo que fossem desenvolvidos fora do contexto de um sistema arquitetural.
- Configuração – possibilidade de descrever a estrutura de um sistema, independente dos elementos sendo estruturados e, se possível, dar suporte à reconfiguração dinâmica. Para isso, uma ADL deveria separar a descrição de estruturas compostas dos elementos constituintes, facilitando o entendimento da composição como um todo.
- Heterogeneidade – possibilidade de combinar descrições arquiteturais múltiplas e heterogêneas.
- Análise – possibilidade de realizar uma análise rica e variada de descrições arquiteturais.

Analisando essas propriedades, pode-se identificar algumas semelhanças com as propriedades que uma linguagem de autoria hipermídia deve satisfazer:

- possibilidade de definir um documento através da junção de blocos de informação independentes (composição);
- possibilidade de descrever componentes e suas interações em um documento, representando a estrutura do documento, de forma independente do significado do conteúdo dos componentes (abstração);
- possibilidade de reutilizar elementos de um documento em outros documentos ou em composições diferentes de um mesmo documento (reuso);
- possibilidade de definir composições na estrutura de um documento e de separar a descrição de elementos compostos de seus elementos constituintes (configuração);
- possibilidade de combinar vários documentos heterogêneos (heterogeneidade);
- possibilidade de verificar a consistência (temporal, espacial etc.) de documentos (análise).

Baseando-se nessas propriedades comuns entre linguagens de autoria hipermídia (HAL) e linguagens de descrição de arquitetura (ADL), este capítulo realiza uma comparação detalhada entre estruturas de sistemas de software e estruturas de documentos hipermídia. Essa comparação serviu de base para identificar que facilidades poderiam ser trazidas do domínio de arquitetura de software para hipermídia introduzindo vantagens.

Como conseqüência da comparação, o capítulo também apresenta a descrição de um metamodelo estrutural, que pode ser usado tanto para especificar arquiteturas de software quanto estruturas de documentos hipermídia, que foi usado como base para o desenvolvimento deste trabalho de tese. Após a descrição do metamodelo estrutural, um resumo de como o metamodelo é usado para descrever estruturas hipermídia é apresentado, introduzindo o conceito de conector hipermídia. Além disso, é discutida a aplicação do conceito de estilos arquiteturais em hipermídia, criando os chamados templates de composição. Finalmente, uma última seção discute brevemente algumas contribuições que a área de hipermídia poderia levar para a área de arquitetura de software.

2.1
Linguagens de Descrição de Arquitetura

Uma ADL pode ser definida como uma linguagem para especificar a estrutura de alto nível de uma aplicação, ao invés de se preocupar com detalhes de implementação de um módulo de código específico. Ela modela a arquitetura conceitual de um sistema de software, distinguindo-a da implementação do sistema. A descrição arquitetural de um sistema é extremamente importante, pois serve como um esqueleto sobre o qual propriedades podem ser provadas e, por isso, tem um papel vital em expor a capacidade do sistema em satisfazer seus requisitos principais.

Na maioria das ADLs, os blocos principais de uma descrição arquitetural são (Medvidovic, 2000):

- componentes: unidades de computação ou armazenamento de dados que podem ser tão pequenos quanto um único procedimento ou tão grandes como uma aplicação inteira;
- conectores: usados para modelar interações entre componentes e regras que governam essas interações. Diferente dos componentes, conectores podem não corresponder a unidades compiladas em um sistema implementado. Eles podem se manifestar como variáveis compartilhadas, chamadas de procedimentos, protocolos cliente-servidor, *pipes* e assim por diante; e
- configurações arquiteturais: grafos conexos de componentes e conectores que descrevem a estrutura da arquitetura. Idealmente, a estrutura de um sistema deveria estar clara a partir de uma especificação de configuração somente, sem precisar estudar as especificações dos componentes e conectores. Composição hierárquica é desejável em ADLs, permitindo a descrição de um sistema em diferentes níveis de detalhe. Uma estrutura e um comportamento complexo poderiam, nesse caso, ser explicitamente representados ou poderiam ser abstraídos em um componente ou conector único que faz parte de outra arquitetura ainda mais complexa.

Para permitir a inferência de qualquer tipo de informação sobre uma arquitetura de software, interfaces de componentes e conectores também precisam ser modeladas. Sem essa informação, uma descrição arquitetural se torna uma coleção de elementos interconectados, similar a um diagrama de “caixas e linhas” sem semântica explícita por detrás. Essa informação é necessária para determinar se os componentes apropriados estão conectados, se suas interfaces casam, se conectores permitem a comunicação adequada e sua semântica combinada resulta no comportamento desejado.

A interface de um componente é seu conjunto de pontos de interação com o mundo exterior. A interface especifica serviços (mensagens, operações e variáveis) que um componente provê e serviços que ele requisita de outros componentes. Dessa forma, a interface define o dever computacional de um componente e suas restrições de uso.

A interface de um conector é um conjunto de pontos de interação que podem ser associados a componentes ou outros conectores, normalmente especificando os serviços que o conector espera desses elementos associados.

Para especificar o comportamento dos componentes e os protocolos de interação entre eles, normalmente as ADLs fornecem um descrição semântica do sistema, além da descrição estrutural de sua arquitetura, que pode ser feita em CSP, π-calculus, redes de Petri etc., ou mesmo através de uma metalinguagem para especificar a semântica de uma ADL (Wile, 1999). Contudo, o foco deste trabalho está nas definições estruturais de ADLs e HALs, sem considerar a especificação semântica, pois, como já foi dito, o interesse principal é identificar os pontos em comum da estrutura de um configuração arquitetural e a estrutura de um documento hipermídia.

Diversas propostas de ADLs podem ser encontradas na literatura, entre elas ACME (Garlan, 1997), Aesop (Garlan, 1995a), Armani (Monroe, 1999), CL (Paula, 1999), C2 (Taylor, 1996; Medvidovic, 1996), Darwin (Magee, 1996; Magee, 1995), Rapide (Luckham, 1995b; Luckham, 1995a), SADL (Moriconi, 1997), Unicon (Shaw, 1995; Shaw, 1996a) e Wright (Allen, 1997a; Allen, 1998). A referência (Muchaluat-Saade, 2000) apresenta um resumo sobre as principais características dessas linguagens.

A Tabela 1 resume características de algumas linguagens de descrição de arquitetura de acordo com os seguintes quesitos:

1. componente composto – especifica se a ADL fornece componentes compostos;
2. interface de componentes – especifica o nome que pontos de interface de componentes têm na ADL específica;
3. conector de primeira classe – especifica se a ADL trata conectores como entidades de primeira classe;
4. interface de conectores – especifica o nome que pontos de interface de conectores têm na ADL específica;
5. conector multiponto – especifica se a ADL permite que mais de dois elementos interajam através de um mesmo conector;
6. ligação direta entre conectores – especifica se a ADL permite que um conector seja diretamente ligado a outro conector;
7. conector composto – especifica se a ADL fornece conectores compostos;
8. restrições sobre mapeamentos entre interfaces de elementos aninhados – especifica se a ADL tem alguma restrição sobre como mapear pontos de interface de elementos compostos em pontos de interface de seus elementos constituintes;
9. definição de estilos arquiteturais – especifica se a ADL fornece a definição de estilos, usados para descrever famílias de sistemas de software ao invés de um sistema específico;
10. definição de restrições – especifica se a ADL permite a definição de restrições entre elementos arquiteturais;

Tabela 1. Resumo de características de algumas ADLs

	ACME	**Aesop**	**Armani**	**CL**	**C2**	**Darwin**	**Rapide**	**SADL**	**UniCon**	**Wright**
1	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
2	***port***	***input port output port***	***port***	***entry port exit port***	***provided port required port***	***provide service require service***	***provides requires in/out action services***	***iport oport***	***player***	***port***
3	ok	ok	ok		ok			ok	ok	ok
4	***role***	***role***	***role***	-	-	-	-	-	***role***	***role***
5	ok	ok	ok		ok				ok	ok
6					ok					
7	ok	ok	ok		ok					ok
8	ok	ok	ok	ok		ok	ok	ok	ok	ok
9		ok	ok					ok	ok	ok
10		ok	ok				ok	ok	ok	ok

2.2
Comparando ADLs e HALs

Estruturas de arquiteturas de software são similares a estruturas de documentos hipermídia. Fazendo uma análise superficial, pode parecer trivial fazer uma correspondência direta entre os elementos estruturais básicos de ADLs e os encontrados em linguagens de autoria hipermídia (HAL). Componentes são análogos a nós, conectores a elos e configurações a nós de composição.

A descrição hierárquica de um sistema de software é análoga à hierarquia de nós de composição em um documento hipermídia, com a diferença que em um sistema de software, o grafo de constituintes (componentes e conectores) deve ser conexo, e isso não é obrigatório em um documento hipermídia.

No entanto, analisando com mais detalhes, pode-se notar que a analogia não é tão direta e trivial. Existem algumas características interessantes em ADLs que não têm analogias em HALs e vice-versa (Muchaluat-Saade, 2001c). As próximas subseções discutem as diferenças mais relevantes no contexto deste trabalho.

2.2.1
Interface de Componentes

ADLs definem a interface de um componente como um conjunto de portas, geralmente chamadas *ports*. Além disso, algumas ADLs classificam os elementos desse conjunto, determinando um tipo para cada elemento de interface. Como exemplo, tem-se *entry* e *exit ports* em CL, *provided* e *required ports* em C2, *provide* e *require services* em Darwin e *iport* e *oport* em SADL. Além dessa classificação básica, que representa o sentido em que o componente realizará uma interação quando usar um determinado ponto de interface, existem outros critérios para especificar tipos. Como exemplo, em Rapide, os elementos de interface de um componente podem ser *provides, requires, in* e *out actions* e ainda *services*, dependendo do tipo de interação em que o ponto de interface específico pode participar.

Em HALs, a interface de um nó é dada por sua lista de âncoras, e geralmente uma âncora representa uma região marcada no conteúdo de um nó usada na definição de elos. Até agora, não foi necessário distinguir tipos diferentes de âncoras, já que qualquer uma delas pode ser origem ou destino de

elos e pode participar em qualquer tipo de relacionamento entre nós. Entretanto, para que um modelo hipermídia satisfaça a propriedade de composicionalidade, diferentes tipos de pontos de interface para nós de composição terão que ser definidos, como será discutido na Seção 2.4.

2.2.2 Conectores

Fazer uma correspondência entre conectores de software e elos em um hiperdocumento não é nada trivial. Conectores têm uma importância grande em descrições de sistemas de software e, em muitas ADLs, seu poder de abstração é comparado ao dos componentes. Com certeza, as principais diferenças entre ADLs e HALs estão relacionadas a conectores, sendo o ponto de partida fundamental para o trabalho que foi desenvolvido. As próximas seções discutem esse assunto com mais detalhes.

2.2.2.1 Cardinalidade de Conectores

Algumas ADLs, tais como ACME, Aesop, Armani, C2, UniCon e Wright, permitem que mais de dois componentes interajam usando o mesmo conector, enquanto outras, como CL e Darwin, não permitem. O mesmo cenário é encontrado considerando linguagens hipermídia. Algumas delas não permitem relacionamentos n-ários entre nós representados por elos, tais como XHTML, SMIL e Madeus (Jourdan, 1998), enquanto outras permitem, como por exemplo, NCL, XLink, e linguagens baseadas em modelos que usam redes de Petri, tais como Trellis/caT (Na, 2001) e HTSPN (Willrich, 1996).

2.2.2.2 Conectores como Entidades de Primeira Classe

Algumas ADLs modelam conectores explicitamente, distinguindo claramente tipos de conectores e instâncias de conectores. Nesse caso, conectores são considerados entidades de primeira classe (Medvidovic, 2000; Shaw, 1994), ou seja, eles podem ser nomeados, subtipados e reusados pelos projetistas de arquitetura. Outras ADLs só oferecem um tipo simples de conector usado para

interligar diretamente dois componentes, chamado de conector *in-line*. Nesse outro caso, os projetistas só podem criar instâncias desse tipo, tais como os *binds* de Darwin e os *links* de CL, onde as instâncias não podem ser manipuladas ou reusadas.

Um conector de primeira classe pode ser definido de forma independente de componentes e configurações onde será usado. Normalmente, uma configuração define instâncias de componentes e conectores, especificando também quais pontos de interface de uma instância de conector estão ligados a quais pontos de interface de instâncias de componentes.

ACME, Aesop, Armani, C2, SADL, UniCon e Wright são exemplos de ADLs que tratam conectores como entidades de primeira classe. Pode-se destacar algumas vantagens de dar esse tipo de tratamento aos conectores, tais como:

- Independência – definição de um tipo de conector independente de saber por quem ele é utilizado, ou seja, que configurações o contém e quais componentes interagem através dele.
- Reuso – já que tipos de conectores têm uma definição independente, configurações diferentes podem criar instâncias de um mesmo tipo de conector.
- Composição – conectores compostos representam grupos de conectores e componentes, modelando interações mais complexas entre componentes de uma arquitetura de software.

Conectores em ADLs podem ser comparados a relações hipermídia expressas por elos em HALs. Apesar de algumas linguagens/modelos hipermídia permitirem que elos possam representar diversos tipos de relação, normalmente apenas uma entidade existe, o elo propriamente dito, tanto para capturar a definição do tipo da relação, quanto a definição do relacionamento (instância da relação).

O tipo da relação capturada por um elo depende da expressividade da linguagem de autoria utilizada. No XHTML, por exemplo, elos são unidirecionais e ponto-a-ponto, representando o comportamento tradicional de navegação hipertexto. Elos SMIL (W3C, 2001d) também são unidirecionais e ponto-a-ponto, mas podem ser ativados pela interação do usuário ou outras condições de disparo, tais como eventos temporais. Nos modelos Dexter (Halasz, 1994) e AHM

(Hardman, 1998), elos podem ser bidirecionais e multiponto, mas são usados somente para capturar relações hipermídia que dependem da interação do usuário. O modelo Dexter não oferece suporte a sincronização e AHM fornece arcos de sincronização (*sync arcs*) ponto-a-ponto para capturar restrições temporais entre duas partes de uma apresentação. Os modelos NCM (Soares, 1995; Soares, 2000) e Labyrinth (Díaz, 2001) permitem a definição de relacionamentos multiponto complexos entre componentes baseados em conceitos distintos de evento. No NCM, um elo é definido como um relacionamento causal ou de restrição entre eventos que ocorrem sobre os nós, como por exemplo, a apresentação ou seleção de uma âncora. Em Labyrinth, um evento é tratado como uma entidade de modelo separada que pode ser anexada a um elo para descrever um relacionamento de causalidade. Um evento Labyrinth define uma condição e uma lista de ações que devem ser executadas quando a condição for satisfeita.

Modelos/linguagens hipermídia diferem na maneira como tratam a entidade elo. No XHTML, por exemplo, elos são embutidos no conteúdo dos nós, impedindo reuso do mesmo nó sem herdar os elos anteriormente definidos. Outros modelos/linguagens permitem a definição do elo como entidade independente, tais como Dexter, AHM, Microcosm (Carr, 1995; Lowe, 1999), XLink, Labyrinth e NCM. Nos modelos Dexter e AHM, elos são considerados um tipo de componente e, diferente de outros modelos, elos podem até ser usados para interligar outros elos (Hardman, 1998). No NCM, Microcosm e XLink, elos só podem interligar nós. No NCM, elos são contidos em nós de composição e devem relacionar nós recursivamente contidos na composição. Nós de composição NCM podem ser reusados, porém elos somente não podiam, na versão anterior do modelo (Soares, 2000). Microcosm e XLink fornecem repositórios de elos independentes, que podem ser reusados em vários documentos.

Apesar da maioria dos modelos/linguagens hipermídia fornecerem uma entidade única para capturar a definição de relacionamentos, alguns deles dividem a definição do relacionamento em duas partes, a primeira especificando a relação e a segunda, os participantes relacionados. Em XLink, por exemplo, as definições dos recursos participantes e das regras de navegação são feitas com tipos distintos de elementos-filho de um elo estendido. Na versão anterior do NCM, um elo era definido como um conjunto de pontos terminais de origem e de destino e outro

atributo chamado ponto de encontro, que especificava o comportamento de navegação. Entretanto, em ambos os casos, o comportamento de navegação está embutido na definição do elo e não pode ser manipulado de forma independente.

Outros modelos hipermídia fornecem conjuntos pré-definidos de tipos de relações que podem ser usados para criar relacionamentos em um documento. Esses modelos quebram a definição de um relacionamento em duas partes e tratam tipos e instâncias de relações de maneira separada. As restrições de alto nível de Madeus (Jourdan, 1998) e as transições tipadas de modelos baseados em redes de Petri, como HTSPN (Willrich, 1996) e Trellis/caT (Na, 2001), são exemplos. Todavia, diferente das ADLs, esses modelos não permitem que o usuário defina novos tipos de relação.

Em resumo, em HALs, as relações hipermídia representadas por elos não recebem o mesmo tratamento que conectores têm em ADLs. Uma das principais contribuições desta tese é introduzir o conceito de conector no domínio hipermídia, dando às relações expressas por elos um tratamento de primeira classe.

2.2.2.3 Interface de Conectores

A maioria das ADLs que tratam conectores como entidades de primeira classe permitem a definição da interface de um conector, especificando um conjunto de elementos que identificam papéis ou funções dos componentes que participarão da interação representada pelo conector. Esses elementos são denominados papéis (*roles*), tal como em ACME, Aesop, Armani, UniCon e Wright.

Quando uma configuração define quais conectores interligam quais elementos, na verdade, são especificadas as ligações entre pontos de interface de componentes (*ports*) e pontos de interface de conectores (*roles*). Essas ligações são chamadas de *attachments* em ACME e Wright, *communication links* em C2 e *ducts* em (Mehta, 2000). Em ADLs que fornecem conectores *in-line*, nenhuma interface de conector é especificada, e um conector liga diretamente dois pontos de interface de componentes.

ADLs não classificam pontos de interface de conectores usando tipos, de forma similar ao que é feito com a interface de componentes. Elas não definem que papéis são origem ou destino de uma relação, por exemplo. Se existir tal sentido na relação, isso será definido pelo tipo de ponto de interface do componente a que um papel é ligado.

Já que relações definidas por elos não têm sido tratadas como entidades hipermídia de primeira classe, nenhuma HAL define explicitamente a interface de uma relação. Em HALs que fornecem tipos pré-definidos de relações, a interface é implicitamente definida pelo tipo. Por exemplo, em HALs baseadas em redes de Petri, nós são representados por lugares e tipos de transições representam tipos de relações. Nesse caso, arcos chegando ou saindo de uma transição definem implicitamente o papel de cada nó que está interagindo através da transição.

Relações de sincronização hipermídia expressas por elos podem ter semântica causal ou de restrição. Quando relações causais são modeladas, condições relacionando pontos terminais de origem do elo devem ser satisfeitas para disparar a execução de ações relacionando pontos terminais de destino do elo. Nesse caso, definir o sentido da relação torna-se essencial. Como âncoras de nós (seus pontos de interface) podem ser origem ou destino de qualquer elo, pontos de interface de nós não devem ser tipados por esse motivo[5]. Desta forma, diferente das ADLs, pontos de interface de relações hipermídia certamente precisarão de tipos para expressar relações de causalidade, como será discutido no Capítulo 4.

2.2.2.4 Ligação Direta entre Conectores

Apesar de não ser uma facilidade comum em ADLs, C2 permite que um conector esteja diretamento conectado a outro conector (Medvidovic, 1996). O mesmo cenário é encontrado em hipermídia. Apesar de não ser um recurso utilizado para autoria de hiperdocumentos, a proposta inicial do padrão MHEG

5 Note que pontos de interface de nós podem ser tipados por outros motivos, como comentado na Seção 2.2.1.

(ISO, 1997), e os modelos Dexter (Halasz, 1994) e AHM (Hardman, 1998) permitem que um elo seja utilizado para interligar outros elos diretamente.

2.2.2.5 Conectores Compostos

Conectores compostos, chamados de *high-order connectors* por Garlan (Garlan, 1998) e bastante discutidos em (Mehta, 2000), representam composições de vários conectores e possivelmente componentes, modelando interações mais complexas entre componentes em uma arquitetura. Com isso, a descrição hierárquica de uma arquitetura de software pode ser feita encapsulando-se subsistemas como componentes ou como conectores. É importante mencionar a discussão encontrada na literatura de arquitetura de software sobre distinguir conectores de componentes. Existem ADLs que não fazem distinção, modelando conectores complexos como componentes, como Rapide (Luckham, 1995b) por exemplo, que fornece um componente chamado componente conector (*connector component*). Entretanto, esta abordagem é criticada por outros trabalhos (Allen, 1997b; Mehta, 2000). Eles argumentam que a função básica de uma linguagem é prover um veículo de expressão que case com a intuição e prática do usuário. Assim sendo, se abstrações têm semânticas distintas, elas devem ser representadas por elementos de linguagem distintos.

Vários trabalhos discutem e defendem a necessidade ou o uso de conectores compostos (Dean, 1995; Shaw, 1995; Garlan, 1995a; Allen, 1997a; Garlan, 1997; Garlan, 1998; Allen, 1998; Mehta, 2000). ACME, Aesop, Armani e Wright oferecem mecanismos para a definição de conectores compostos e os distinguem claramente de componentes compostos.

Alguns modelos hipermídia oferecem nós de composição, mas nenhum oferece elos representando relações cuja semântica seja dada por composições de outros nós e elos. O único sistema hipermídia que oferece um conceito parecido com o de relações compostas, é o sistema caT (*Context-Aware Trellis*) (Na, 2001; Furuta, 2002) que oferece sub-redes (*subnets*) de Petri representadas por transições compostas. Outra contribuição desta tese é introduzir a possibilidade de criar relações hipermídia compostas de nós e elos como um conector hipermídia composto.

2.2.3
Mapeamentos entre Interfaces de Elementos Aninhados

ADLs não permitem criar uma interação direta entre um elemento no interior de uma composição e um elemento fora da composição. Para permitir essa facilidade de forma indireta, um mapeamento entre pontos de interface de elementos aninhados é necessário, como ilustrado na Figura 2.

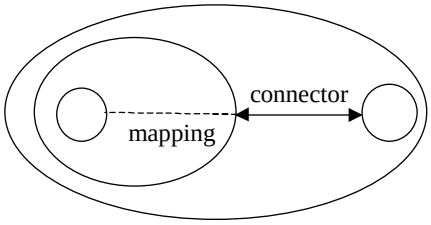
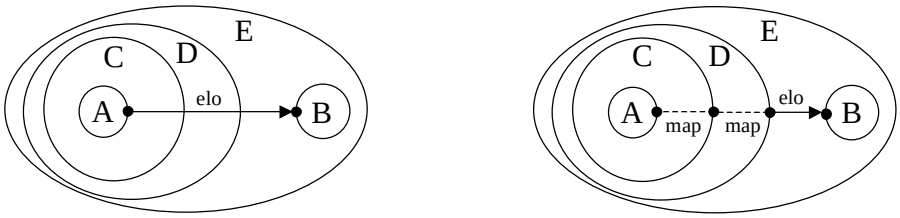


Figura 2. Definição de interações entre componentes em configurações distintas

Algumas ADLs fazem restrições quanto a esse mapeamento, exigindo que o tipo do elemento constituinte seja igual ao tipo do elemento composto. Exemplos de tais mapeamentos com restrições são os *rep-maps* de ACME e os *bindings* de Aesop, Armani e Wright. Outras linguagens, como C2, não fazem nenhuma restrição com relação a esse mapeamento.

Ainda em outras ADLs, a relação entre elementos aninhados é representada por um conector *in-line* (veja Seção 2.2.2.2) interligando os componentes pai e filho, como os *binds* de Darwin e os *links* de CL.

Algumas HALs permitem a conexão direta entre nós contidos em composições distintas, tal como linguagens baseadas no modelo NCM, de acordo com o ilustrado na Figura 3(a). Nesse caso, quando o reuso de nós é permitido, um elo deve distinguir a seqüência de nós aninhados a que se refere. Por exemplo, o elo mostrado na Figura 3(a), contido na composição E, toca o nó A dentro do aninhamento (D, C, A). Essa seqüência de aninhamento é análoga a uma seqüência de mapeamentos entre uma composição e seu constituinte na seqüência, como ilustrado na Figura 3(b). Entretanto, a versão anterior do modelo NCM tratava essa seqüência de aninhamento como parte da definição do ponto terminal do elo e não oferecia o conceito de mapeamento como em ADLs.



(a) elo ancorando diretamente em nós internos (b) elo ancorando indiretamente em nós internos

Figura 3. Elo entre nós contidos em composições distintas

A vantagem de oferecer a criação de mapeamentos e não permitir uma conexão direta entre elementos de composições distintas é satisfazer à propriedade de composicionalidade, facilitando mecanismos de verificação de consistência das estruturas criadas (Félix, 2002). Outra contribuição desta tese é incorporar essa facilidade à nova versão do modelo NCM e ,conseqüentemente, à nova versão da linguagem NCL, que será abordada no Capítulo 3.

2.2.4
Estilos Arquiteturais e Restrições

Estilos arquiteturais são úteis para representar estruturas bem conhecidas que são usadas freqüentemente em várias aplicações e projetos de software. Um estilo especifica uma família de sistemas de software, enquanto uma configuração define um sistema específico. Estilos em ADLs definem tipicamente quatro tipos de propriedades (Garlan, 1995a):

1. Eles fornecem um vocabulário de elementos de projeto – tipos de componentes e conectores tais como, *pipes*, filtros, clientes, servidores, *parsers*, bancos de dados etc.
2. Eles definem um conjunto de regras de configuração – ou restrições de topologia – que determinam as composições permitidas entre esses elementos. Por exemplo, as regras podem proibir ciclos em um estilo *pipes*-e-filtros, especificar que uma organização cliente-servidor deve ser um relacionamento n-para-1, ou definir um padrão de composição específico tal como uma decomposição *pipeline* de um compilador.
3. Eles definem uma interpretação semântica, onde composições de elementos de projeto, governados por certas regras de configuração, têm um significado bem definido.

4. Eles definem análises que podem ser realizadas em sistemas que seguem um estilo. Exemplos de análises incluem análises de escalonamento para estilos orientados a processamento em tempo real, ou detecção de *deadlocks* em transferências de mensagens em um sistema cliente-servidor.

A definição de restrições em ADLs normalmente está relacionada a definição de estilos, obrigando todos os sistemas que seguem um estilo a satisfazer as restrições que ele especifica. Exemplos são as regras de configuração (*configuration rules*) de Aesop, as restrições de Wright chamadas *constraints* e as restrições de Armani chamadas *invariants* e *heuristics*. Outras ADLs como SADL e Rapide permitem a definição de restrições dentro de qualquer configuração.

No que se refere à utilização de estilos arquiteturais, pode-se ressaltar algumas vantagens significativas:

1. Facilitar o reuso de projeto: soluções rotineiras com propriedades bem conhecidas podem ser reaplicadas a novos problemas.
2. Reutilizar o código: normalmente os aspectos invariantes de um estilo arquitetural permitem implementações compartilhadas.
3. Facilitar o entendimento da organização de sistemas através do uso de estruturas padronizadas.
4. Oferecer suporte a interoperabilidade através do uso de estilos padronizados.
5. Permitir análises especializadas e específicas a um estilo.
6. Fornecer visualizações específicas a um estilo, o que ajuda a representação textual ou gráfica a chegar mais perto da intuição do engenheiro.

O suporte para a definição de estruturas de documentos reusáveis similar à definição de estilos em ADLs não é oferecido em HALs. Outra importante contribuição desta tese é oferecer suporte similar em hipermídia, permitindo a criação dos chamados templates de composição, como será discutido na Seção 2.5.

2.3
O Metamodelo Estrutural

A comparação resumida na seção anterior, e discutida mais detalhadamente em (Muchaluat-Saade, 2001c), serviu como ponto de partida para a definição de um metamodelo estrutural que pode ser especializado para uso em ADLs ou HALs para representar estruturas de arquitetura de software ou de documentos hipermídia. A especialização do modelo para arquitetura de software ou hipermídia deve ser complementada com uma descrição semântica das entidades definidas pelo metamodelo. Essa descrição semântica é específica para cada domínio. Como dito anteriormente, esta tese só abordou a especificação estrutural de forma genérica.

O metamodelo estrutural é mais parecido com os modelos usados em ADLs, pelo fato de serem mais genéricos que os modelos de HALs, acrescentando pequenas diferenças que serão realçadas no texto.

O metamodelo define dois tipos de vértices, chamados componentes *(component)* e conectores *(connector)*, cada um contendo um conjunto de pontos de interface. O metamodelo também define um tipo básico de arco, chamado *bind*, conectando um ponto de interface de um componente a um ponto de interface de um conector. *Binds* não podem conectar diretamente componente a componente ou conector a conector (Muchaluat-Saade, 2001c). Se tal conexão for necessária, vértices intermediários *dummy* (componentes ou conectores) podem ser criados para representá-la. Note que, para as ADLs que oferecem conectores *in-line*, uma conexão direta entre dois componentes deve ser representada por um conector *dummy*. De forma análoga, para ligação direta entre dois conectores, um componente *dummy* deve ser utilizado.

Uma estrutura simples definida pelo metamodelo é ilustrada graficamente na Figura 4.

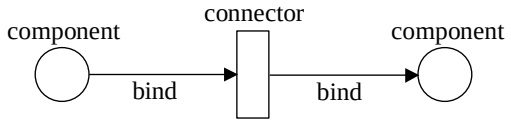


Figura 4. Exemplo de uma estrutura simples

O metamodelo admite que vértices possam representar subgrafos também compostos de componentes, conectores e *binds*, que ainda podem estar aninhados

em qualquer profundidade. Diferente de ADLs, um subgrafo que representa um vértice composto não precisa ser conexo.

Um componente composto pode ser diretamente conectado, através de um *bind*, a um conector constituinte. Essa é a diferença principal entre este metamodelo estrutural e o modelo usado em ADLs. Isso é necessário para modelar interações entre composições e seus componentes em documentos hipermídia. Por outro lado, um conector composto não poder ser conectado diretamente a um de seus componentes constituintes, pois esse tipo de interação não faz sentido e, portanto, não é permitida nem em ADLs e nem em HALs. A Figura 5 ilustra vértices compostos.

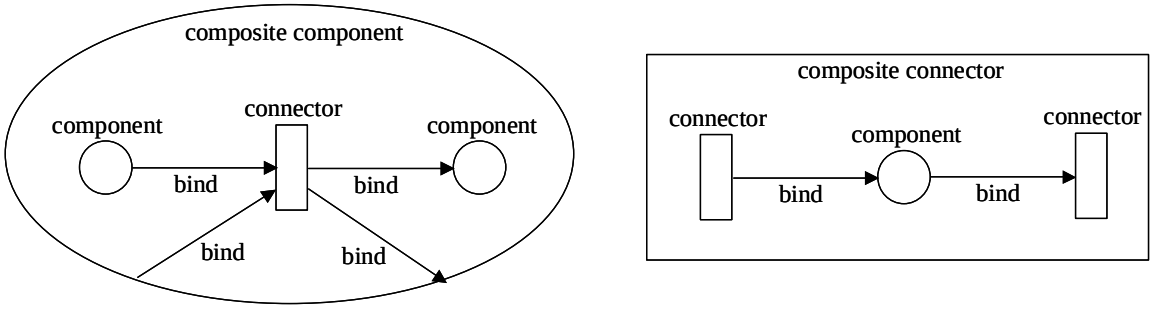


Figura 5. Exemplos de componente e conector compostos

ADLs e algumas HALs permitem que vértices dentro de uma composição sejam conectados a vértices fora dela. Essas interações são sempre descritas indiretamente em ADLs e podem ser definidas direta ou indiretamente em HALs. Sem perda de generalidade, conexões diretas não são permitidas no metamodelo, já que *binds* só podem ser definidos entre vértices tendo o mesmo pai ou entre vértices pai e filho. Isso garante a propriedade de composicionalidade no metamodelo estrutural. Para permitir conexões indiretas entre vértices dentro de uma composição e vértices fora dela, outro tipo de arco é necessário, chamado mapeamento (*map*). Um mapeamento conecta vértices pai e filho de um mesmo tipo, tal como um componente/conector pai a um componente/conector filho[6]. Conexões entre vértices dentro e fora de uma composição podem ser feitas através de caminhos de arcos formados por *binds* e *maps*. A Figura 6 ilustra o uso de mapeamentos. É importante notar, que diferente de *binds*, mapeamentos não representam interações entre componentes ou entre conectores. Eles servem

apenas como meio para exportar pontos de interface internos para a interface externa de um vértice composto.

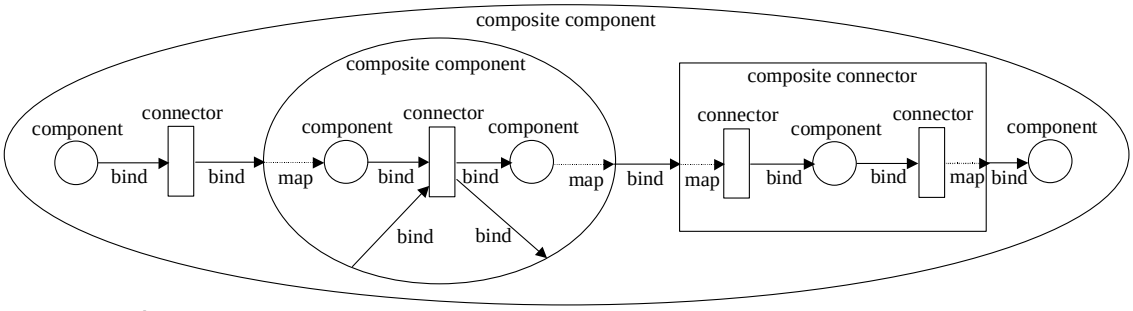


Figura 6. Vértices compostos, binds e maps

Mais precisamente, um vértice V é uma estrutura em grafo definida por uma tupla V = (IP, VM, VN, B, M), onde:

- IP é um conjunto $\{ip_1, ip_2, ..., ip_x\}$ de pontos de interface de V
- VM é um conjunto $\{vm_1, vm_2, ..., vm_n\}$ de vértices componentes, onde cada vértice vm_j tem um conjunto de pontos de interface IPj = $\{ipj_1, ipj_2, ..., ipj_z\}$
- VN é um conjunto $\{vn_1, vn_2, ..., vn_m\}$ de vértices conectores, onde cada vértice vn_j tem um conjunto de pontos de interface IPj = $\{ipj_1, ipj_2, ..., ipj_z\}$
- B é um conjunto $\{b_1, b_2, ..., b_k\}$ de *binds*, onde cada *bind* é uma tupla $(vm_a, ipa_k, vn_b, ipb_j)$ ou uma tupla (V, ip_i, vn_b, ipb_j), tal que $vm_a \in$ VM, $ipa_k \in \text{IP}_\text{a}$, $ip_i \in$ IP, $vn_b \in$ VN, $ipb_j \in \text{IP}_\text{b}$, e
- M é um conjunto $\{m_1, m_2, ..., m_r\}$ de mapeamentos, onde cada mapeamentos é uma tupla (V, ip_i, v_b, ipb_j), tal que $ip_i \in$ IP, e:
  - se o vértice V for um componente, $v_b \in$ VM, $ipb_j \in \text{IP}_\text{b}$,
  - se o vértice V for um conector, $v_b \in$ VN, $ipb_j \in \text{IP}_\text{b}$.

Se um vértice V não for composto, ele será representado por V = (IP, ∅, ∅, ∅, ∅).

Uma estrutura G é representada por uma tupla (IP, VM, VN, B, M), onde VM ou VN não são conjuntos vazios. Quando uma estrutura G for aplicada a um domínio particular, algumas restrições provavelmente precisarão ser definidas

[6] Tal como na maioria das ADLs, esse tipo de restrição foi mantido no metamodelo (veja

para garantir a sua consistência, entretanto, essas restrições não são definidas pelo metamodelo.

A próxima seção discute como representar estruturas hipermídia utilizando o metamodelo estrutural.

2.4 Representando Estruturas Hipermídia com o Metamodelo

Para aplicar o metamodelo estrutural ao domínio hipermídia, entidades de modelos hipermídia (Halasz, 1994), tais como objetos de mídia, nós de composição e elos, devem ser descritas em termos de entidades do metamodelo.

Objetos de mídia e nós de composição encontrados em HALs são tipos especiais de componentes. Pontos na interface de um objeto de mídia são âncoras ou atributos do nó. Pontos na interface de um nó de composição podem ser âncoras, atributos ou portas do nó, de acordo com o que segue.

Normalmente, uma âncora representa uma região marcada no conteúdo de um nó. No caso de um objeto de mídia, uma âncora pode ser qualquer conjunto de unidades de informação em seu conteúdo. Uma unidade de informação depende do tipo do objeto de mídia e pode ser um caractere em um objeto de mídia texto, um quadro em um objeto de mídia vídeo etc. No caso de um nó de composição, uma âncora pode ser qualquer conjunto de seus constituintes. É importante manter âncoras para nós de composição hipermídia, pois um elo pode ancorar em um nó de composição diretamente, para iniciar a apresentação de sua estrutura e não a apresentação de seus componentes, por exemplo.

Uma porta de um nó de composição é usada para criar pontos de entrada e saída no nó, permitindo que elos externos ancorem em nós contidos na composição, sem perder a composicionalidade. Isso permite, entre outras coisas, considerar a apresentação da composição como sendo a apresentação de seus componentes. Um nó de composição pode tornar um ponto de interface de um de seus nós constituintes visível para qualquer relação com qualquer outro nó fora da composição. Isso é feito através de um mapeamento, como será visto

---

Seção 2.2.3).

posteriormente, que associa um ponto de interface de um constituinte a uma porta da composição. Um exemplo de uso de mapeamentos em um nó de composição hipermídia será dado mais adiante nesta mesma seção.

Como discutido anteriormente, relações hipermídia expressas por elos não recebem tratamento de primeira classe pela maioria dos modelos hipermídia. Para dar status de primeira classe a essas relações, a definição da relação deve ser totalmente separada da definição do relacionamento. Com isso, a especificação de um elo deve ser dividida em dois elementos, o primeiro define a relação e o outro define as conexões entre o primeiro elemento e os componentes (nós do hiperdocumento) que participam do relacionamento. Para realizar a definição da relação de forma independente dos elos, uma nova entidade deve ser introduzida capturando o conceito de conector. Essa entidade foi chamada de **conector hipermídia** (Muchaluat-Saade, 2001a; Muchaluat-Saade, 2001b).

Um conector hipermídia representa uma relação que será usada para criar elos hipermídia entre nós de um documento. Ele especifica a semântica da relação, mas não especifica quais nós participarão do relacionamento. Os nós serão especificados pelos elos hipermídia que usarão essa relação, ou seja, que usarão o conector.

Um conector é definido por um conjunto de pontos de interface, chamados *roles* (papéis), e um atributo chamado *glue* (cola) (Allen, 1997a), similar à definição feita em ADLs. Cada papel de um conector especifica a função de um participante da interação, e o *glue* do conector descreve como os participantes interagem. O Capítulo 4, que discute a linguagem XConnector para criação de conectores hipermídia, dará a definição semântica completa de um papel e do *glue* baseada no conceito de evento.

Enquanto conectores representam relações, elos são usados para representar relacionamentos entre nós hipermídia. Assim sendo, um elo faz referência a um conector e ainda define um conjunto de *binds* relacionando papéis do conector a pontos de interface de nós, seguindo as definições do metamodelo.

Um exemplo simples de relação hipermídia é a relação de referência tradicional hipermídia, que causa a navegação para um nó de destino quando uma âncora de um nó de origem for selecionada pelo usuário. Nesse caso, um conector hipermídia definiria dois papéis, identificados por *selection_condition* e

presentation_action, por exemplo, e o *glue* do conector daria a semântica causal entre eles, especificando que se uma seleção acontecer no participante desempenhando o papel *selection_condition*, a apresentação do participante desempenhando o papel *presentation_action* deve ser executada. A Figura 7 ilustra um conector hipermídia *CON1* modelando a relação tradicional hipermídia e o documento *COMP1* que possui dois elos distintos referenciando o mesmo conector. O elo l_1 especifica que se a âncora *m* do nó *A* for selecionada, a âncora λ do nó *B* será apresentada, e o elo l_2 especifica que se a âncora *n* do nó *A* for selecionada, a âncora λ do nó *C* será apresentada.

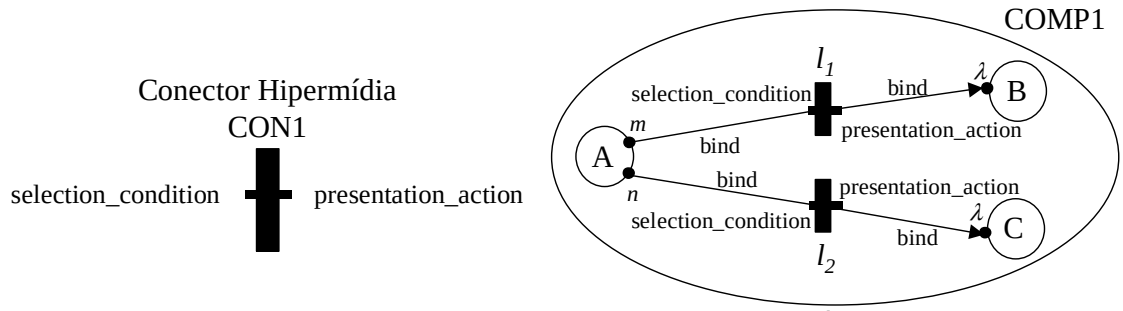


Figura 7. Exemplo de elos distintos usando o mesmo conector hipermídia

A definição completa de um elo é dada por:

- uma referência a um conector hipermídia;
- um conjunto de *binds* relacionando cada papel do conector a um ponto de interface de um nó.

Se o ponto de interface de um nó especificado em um *bind* for uma porta de uma composição, deve existir um caminho de mapeamentos relacionando aquela porta a pontos de interface de seus nós constituintes, até que uma âncora ou atributo seja encontrado. Contudo, vale ressaltar que os mapeamentos não são definidos por elos e sim por nós de composição relacionados, como será detalhado mais adiante.

Um *bind* especifica os seguintes atributos:

- nó *N;*
- ponto de interface de *N*[7]*;*

[7] Pontos de interface de nós ancorados por elos são portas ou âncoras definindo relacionamentos de apresentação ou seleção, ou ainda atributos, para o caso de relações de atribuição.

- especificação de apresentação do nó *N* (opcional);
- conector hipermídia;
- papel do conector hipermídia usado.

A especificação de apresentação de um nó pode ser definida por um elo ancorando no nó para adaptar as características de apresentação de acordo com a navegação feita, por exemplo, associando uma nova folha de estilo a uma página HTML que seja destino de um elo. Algumas HALs oferecem essa facilidade, como NCL, por exemplo.

Voltando ao exemplo da Figura 7, o conjunto de *binds* do elo l_1 é {(*A, m, CON1, selection_condition*), (*B, λ, CON1, presentation_action*)} e conjunto de *binds* do elo l_2 é {(*A, n, CON1, selection_condition*), (*C, λ, CON1, presentation_action*)}, onde a âncora λ representa todo o conteúdo de um nó.

Em HALs que permitem que um nó seja reutilizado em nós de composição distintos e que um elo atravesse uma composição, tal como ilustrado na Figura 8, um ponto terminal de um elo deve identificar a seqüência de nós aninhados usada para atingir a ocorrência do nó ancorado pelo elo.

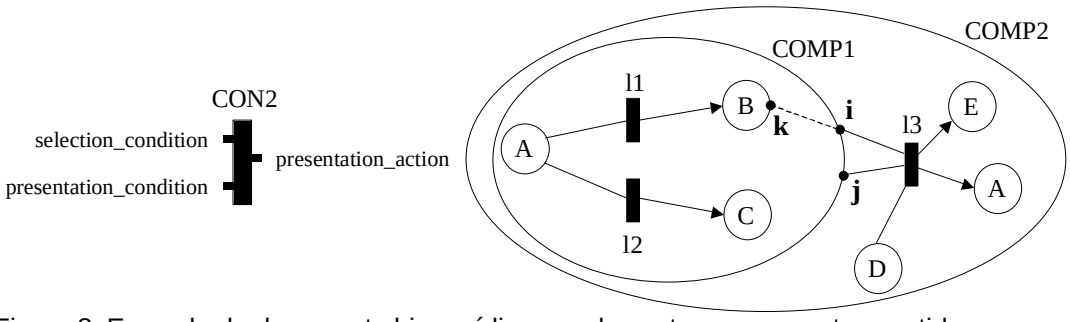


Figura 8. Exemplo de documento hipermídia com elos entre componentes contidos em composições distintas

Alguns modelos hipermídia, como o NCM em sua definição anterior (Soares, 1995; Soares, 2000), permitem que elos ancorem diretamente em nós contidos em composições distintas através da definição de uma seqüência de nós aninhados em cada ponto terminal do elo. Apesar de ser uma característica importante que deve ser oferecida por um modelo conceitual hipermídia, isso também impede a composicionalidade, pois elos podem atravessar uma composição sem notificá-la. Como salientado anteriormente, para permitir que elos externos possam ancorar indiretamente em nós dentro de uma composição e ainda satisfazer à propriedade de composicionalidade, a definição de portas em

nós de composição foi necessária. No entanto, portas somente não permitem que elos ancorem dentro de uma composição.

Seguindo o metamodelo, nós de composição devem ainda definir mapeamentos relacionando suas portas a pontos de interface de seus nós constituintes. Esses mapeamentos são definidos em um atributo de nó de composição, chamado conjunto de mapeamentos. Um mapeamento tem os seguintes atributos:

- nó de composição;
- porta do nó de composição;
- nó constituinte;
- ponto de interface do nó constituinte, que pode ser uma porta, âncora ou atributo.

Cada porta de nó de composição deve ter um mapeamento associado. Um elo pode ancorar indiretamente em um constituinte da composição através de uma porta e um caminho de mapeamentos, que o leva até o nó desejado. Voltando ao exemplo da Figura 8, o nó de composição *COMP1* define um mapeamento entre sua porta *i* e a âncora *k* do nó *B*, representado por (*COMP1, i, B, k*) e ilustrado por uma linha pontilhada na figura. O elo *l3* especifica que uma seleção na âncora *k* do nó *B*, durante a apresentação da âncora *j* do nó de composição *COMP1* e durante a apresentação do objeto de mídia *D*, provoca a execução da apresentação dos nós *E* e *A*. O conjunto de *binds* de *l3* é {(*COMP1, i, CON2, selection_condition*), (*COMP1, j, CON2, presentation_condition*), (*D, λ, CON2, presentation_condition*), (*E, λ, CON2, presentation_action*), (*A, λ, CON2, presentation_action*)}. O *bind* relacionado a porta *i* do nó de composição *COMP1* associado ao *map* (*COMP1, i, B, k*) faz a âncora *k* do nó *B* desempenhar o papel *selection_condition* do conector *CON2*.

Considerando a garantia de consistência de estruturas hipermídia baseadas no metamodelo, uma restrição necessária é que não deve existir conector isolado contido em um documento hipermídia, ou seja, cada elo deve definir pelo menos um *bind* associado a cada papel do conector utilizado. Além disso, como um mesmo conector pode ser reutilizado em elos distintos, se a definição do conector

mudar, todos os elos que o referenciam devem ser checados para verificar qualquer inconsistência.

Outra conseqüência de aplicar o conceito de conector a linguagens de autoria hipermídia é a possibilidade de se ter conectores hipermídia compostos, que é uma facilidade interessante não encontrada em nenhuma HAL. Um conector hipermídia composto tem os seguintes atributos:

- conjunto de *roles* (papéis), que possuem identificadores únicos no conjunto. A definição de um papel do conector composto é dada pelo papel de um conector constituinte, como definido a seguir;
- *glue*, que define:
  - conjunto de nós;
  - conjunto de elos;
  - conjunto de elos parcialmente definidos, que podem não ter *binds* para todos os papéis de seus conectores, mas que devem ter mapeamentos para estes papéis;
  - conjunto de mapeamentos, onde cada mapeamento relaciona um papel do conector composto a um papel de uma ocorrência de um conector constituinte, representada por um elo parcialmente definido. Cada papel do conector composto deve ter um mapeamento associado.

Diferente de um nó de composição, o grafo de nós, elos e elos parcialmente definidos contidos em um conector composto deve ser conexo, pois do ponto de vista conceitual, um conector deve representar uma única relação, sem definições alternativas.

Para ilustrar a utilidade de conectores hipermídia compostos, considere o conector *CON3* ilustrado na Figura 9.

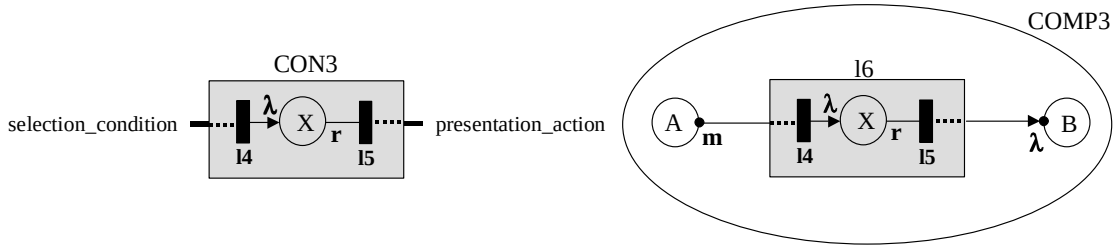


Figura 9. Exemplo de documento hipermídia usando um conector hipermídia composto

Na Figura 9, *CON3* representa uma relação hipermídia tradicional que requer uma confirmação do usuário antes de realizar a navegação ao nó de destino. Esse é um procedimento usual em browsers web comerciais, quando o usuário navega para uma página segura, por exemplo. Um documento *COMP3* pode definir um elo *l6*, que usa o conector *CON3*, e define um conjunto de binds {(*A, m, CON3, selection_condition*), (*B, λ, CON3, presentation_action*)}. No documento *COMP3*, os nós *A* e *B* desempenham os papéis da relação tradicional hipermídia, mas essa relação envolve a apresentação de um nó intermediário *X*. Quando a âncora *m* do nó *A* é selecionada, o elo *l4* provoca a apresentação do nó *X*. Então, quando a âncora *r* do nó *X* é selecionada, o elo *l5* provoca a apresentação do nó *B*. Considerando o exemplo dado inicialmente, o nó *X* seria a mensagem do browser que avisa ao usuário que a próxima página será transferida usando um protocolo de comunicação seguro.

Note que um elo que referencia um conector hipermídia composto é definido da mesma forma que um elo usando um conector simples. O elo deve sempre referenciar o conector hipermídia e definir o conjunto de *binds* relacionando papéis do conector a pontos de interface de nós.

Separar a especificação da relação e dos relacionamentos, ou seja, separar a definição do conector e dos elos que efetivamente o usarão, fornece algumas vantagens a linguagens de autoria hipermídia, tais como:

- Reuso da mesma relação (conector hipermídia) em elos distintos, que possuem a mesma semântica, mas especificam uma interação entre nós diferentes.
- Facilidade de definição de relações de alto-nível que uma HAL poderia fornecer. Agora uma linguagem não precisa oferecer somente um conjunto pré-definido de relações de alto-nível para facilitar o processo de autoria, mas ela também oferece meios para os usuários criarem suas próprias relações. Essas relações, representadas por conectores hipermídia, funcionam como templates de elos e podem ser reutilizadas por autores para criar elos em seus documentos. Essa característica fornece poder de expressão e facilidade de uso a uma linguagem.
- Tratamento das relações como entidades de primeira classe, que podem até ser compostas por outros nós e elos.

Nesta seção, foram apresentados alguns conceitos básicos que podem ser aplicados a qualquer linguagem de autoria hipermídia que deseja tratar relações como entidades de primeira classe, incorporando o conceito de conector hipermídia para definir elos entre componentes de um documento. A nova versão do modelo NCM (Soares, 2003) foi definida com base no metamodelo, utilizando as facilidades oferecidas por conectores hipermídia. Como conseqüência, a versão 2.0 da linguagem NCL também incorporou o uso de conectores, sendo seu desenvolvimento apresentado em detalhes nos Capítulos 3 e 4.

2.5 Aplicando o Conceito de Estilo Arquitetural em Linguagens de Autoria Hipermídia

A especificação de relações representadas por conectores, de forma independente dos relacionamentos representados por elos, pode ser vista como uma forma de definir templates de elos, que devem se interpretados por formatadores de documentos hipermídia. Pode-se generalizar essa idéia para estruturas de documentos, tornando possível a definição de **templates de composição hipermídia** (Muchaluat-Saade, 2002b; Muchaluat-Saade, 2002c), definindo estruturas hipermídia que podem ser reusadas em vários documentos distintos.

Como definido anteriormente, uma composição hipermídia é composta de componentes, representando objetos de mídia ou outras composições, e relacionamentos entre eles, possivelmente representados através de elos. Um template de composição especifica os tipos de componentes, tipos de relações, componentes e relacionamentos que uma composição possui, ou pode possuir, sem identificar quem são todos os componentes e relacionamentos propriamente ditos. Essa tarefa é responsabilidade das composições que usam o template.

A definição de templates é similar à definição de estilos arquiteturais em ADLs, adicionando a possibilidade de definir instâncias específicas de componentes e conectores, além do vocabulário de tipos, o que é bastante útil em documentos hipermídia. A definição de um template de composição é feita em duas partes, onde somente a primeira é obrigatória:

1. vocabulário, que define tipos de componentes, tipos de conectores e pontos de interface;
2. conjunto de restrições sobre elementos do vocabulário e inclusão de instâncias de componentes e conectores, representando relacionamentos entre componentes.

A vantagem principal do uso de templates de composição, além da reutilização da estrutura, é a possibilidade de definição de uma semântica para uma composição. Como mencionado anteriormente, alguns modelos/linguagens hipermídia utilizam composições para especificar relações de sincronização. Como exemplo, tem-se a linguagem SMIL, que fornece três tipos distintos de composição com semântica temporal, que são *par*, *seq* e *excl*. Nessas linguagens, relacionamentos mais complexos entre um conjunto de componentes têm que ser descritos por uma hierarquia de composições usando os tipos básicos. Isso nem sempre é desejável, pois obriga a estrutura lógica do documento a ser igual a sua estrutura de apresentação. Utilizando uma nova abordagem, esses tipos pré-definidos de composições podem ser vistos como templates, como ilustrado na Figura 10. A figura mostra um template de composição com semântica seqüencial (a mesma que o elemento *seq* do SMIL) usando elos para especificar o comportamento de sincronização entre seus componentes. A referência (Rodrigues, 2002a) discute a representação de composições paralelas e seqüenciais através de elos de sincronização com mais detalhes. O template mostrado na Figura 10 é usado por uma composição contendo os componentes *A* e *B*. O documento final gerado depois do processamento do template conterá *A, B* e ainda os elos de sincronização, dando a semântica seqüencial.

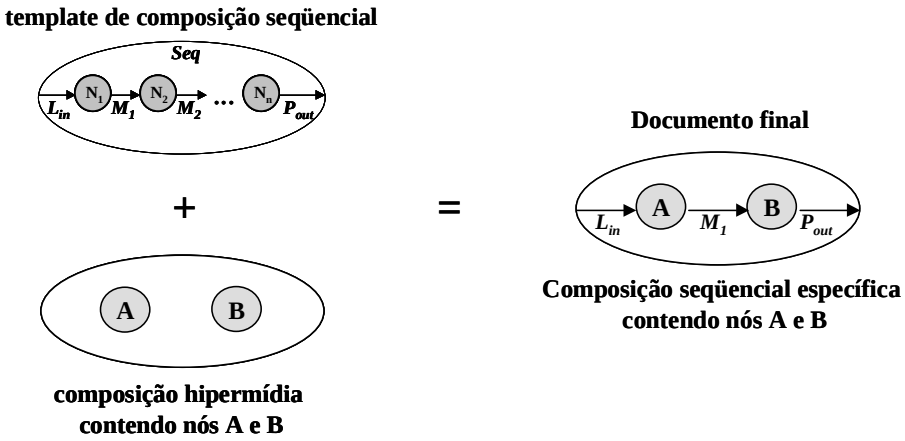


Figura 10. Uso de um template de composição com semântica temporal seqüencial

Se uma linguagem para criação de templates for oferecida, composições com diferentes semânticas embutidas podem ser criadas e os autores não precisam mais ficar limitados a um conjunto de composições pré-definido pela linguagem de autoria. Conceitualmente, um template de composição pode especificar qualquer tipo de relação entre seus componentes. Considerando o uso de templates para representar relacionamentos de sincronização temporal entre componentes, tal como no exemplo da Figura 10, fica mais natural a introdução de sincronização em linguagens já existentes (W3C, 2002; Ossenbruggen, 2000), pois qualquer elemento composto em uma linguagem pode assumir um comportamento de sincronização específico, se utilizar um template de composição com essa semântica.

Através do suporte para criação e uso de templates de composição, uma linguagem hipermídia pode reservar o uso de composições para a criação de relações de estruturação, o uso de elos para a criação de relações de referência e de sincronização e o uso de templates para embutir semântica, por exemplo temporal, em composições, facilitando a tarefa de autoria do usuário. Com isso, conseguimos atingir o cenário ideal que deveria ser oferecido por uma linguagem de autoria hipermídia, como discutido no Capítulo 1, dando ao autor a possibilidade de escolha de como estruturar seu documento e definir as relações entre seus componentes. Note que o modelo hipermídia em que essa linguagem de autoria se baseia deve seguir o metamodelo estrutural apresentado na Seção 2.3, ou seja, incorporar o conceito de conectores para definição de elos. Entretanto, não é necessário incorporar o conceito de template de composição dentro do modelo, essa é uma facilidade que a linguagem de autoria deve oferecer. Como exemplo prático, a entidade conector hipermídia precisou ser incorporada tanto na nova versão do modelo NCM (Soares, 2003) como na da linguagem NCL. Por outro lado, templates de composição são recursos só da linguagem NCL e não do modelo NCM. A linguagem para criação de templates de composição, batizada de XTemplate, será abordada em detalhes no Capítulo 5.

2.6
Possíveis Contribuições de HALs para ADLs

De forma análoga à introdução de facilidades de linguagens de descrição de arquitetura (ADL) em linguagens de autoria hipermídia (HAL), como discutido nas Seções 2.4 e 2.5, algumas facilidades disponíveis no domínio hipermídia também podem ser aplicadas ao domínio de arquitetura de software trazendo benefícios.

Recentemente, as linguagens de autoria hipermídia têm sido especificadas de uma forma modular, permitindo a união de subconjuntos de módulos de linguagens distintas, gerando os chamados perfis. Essa facilidade é bastante útil, pois permite o uso de recursos de uma linguagem em outra, promovendo o reuso de definições, facilitando extensões às linguagens e a utilização de especificações padronizadas. Como exemplo, tem-se a introdução de facilidades de sincronização do SMIL no XHTML através do perfil XHTML+SMIL (W3C, 2002). Essa idéia de especificação modular de linguagens certamente seria útil no domínio de arquitetura de software, onde novos recursos poderiam ser integrados a uma linguagem através da inclusão de novos módulos, permitindo um intercâmbio maior de facilidades entre as próprias ADLs.

Outra facilidade desenvolvida recentemente para linguagens baseadas em XML é a possibilidade de utilização de linguagens de transformação, que permitem transformar documentos criados por uma linguagem em documentos seguindo outras linguagens. Um exemplo de linguagem de transformação é o XSLT (XSL Transformations) (W3C, 1999c). Uma aplicação bastante útil do XSLT é a transformação de documentos XML em XHTML, permitindo sua apresentação em browsers web tradicionais. Outro exemplo de aplicação de XSLT é o processamento de templates de composição para a linguagem NCL 2.0, cujo desenvolvimento é abordado no Capítulo 6 desta tese. Provavelmente, o uso de linguagens de transformação seria útil no domínio de arquitetura de software, permitindo a conversão de especificações feitas em uma ADL para outra. Essa facilidade poderia permitir, por exemplo, a utilização de ferramentas de validação já desenvolvidas para uma ADL para validar especificações feitas em outra ADL, no entanto, isso exigiria uma metalinguagem e uma linguagem de transformação específica para ADLs.

Pelo fato de sistemas hipermídia terem que lidar com estruturas grandes e complexas representando conjuntos de hiperdocumentos interligados, bastante discussão sobre visualização de estruturas hipermídia pode ser encontrada na literatura. Alguns trabalhos (Muchaluat-Saade, 1996a; Muchaluat-Saade, 1996b) propõem a utilização de mecanismos de filtragem ao apresentar o grafo composto de nós e elos, contidos em uma base de documentos, a fim de tornar a visão mais legível para o usuário. Esses mecanismos para filtragem de informações podem ser aplicados diretamente às ferramentas de visualização gráfica de arquiteturas de software.

Uma sub-área recente de pesquisa em documentos hipermídia trata da possibilidade de adaptar documentos às características específicas de um autor ou de uma plataforma de exibição (Rodrigues, 2003). Isso é necessário quando, por exemplo, o autor de um documento especifica a utilização de objetos de mídia que precisam de certos recursos não disponíveis em uma plataforma de exibição, sejam eles recursos relacionados à infraestrutura de comunicação ou processamento. A mesma situação pode existir em sistemas de software, por isso, algumas ADLs já prevêem a possibilidade de reconfiguração dinâmica de uma arquitetura. Com certeza, um estudo mais detalhado poderia comparar as facilidades presentes em ADLs e HALs com relação à reconfiguração e adaptação, permitindo talvez identificar recursos de HALs que pudessem ser aplicados em ADLs, ou vice-versa.

Como mencionado no Capítulo 1, esta tese não tem como objetivo desenvolver esses possíveis tópicos em que a área de hipermídia poderia contribuir para a área de arquitetura de software.

Além dessas possíveis contribuições de HALs para ADLs, outros tópicos são de interesse comum às duas áreas. Pesquisas em controle de versões, uso de XML para especificar sintaxe, ambientes para autoria gráfica, provisão de qualidade de serviço etc. são certamente de interesse a ambas as áreas.