6
Implementação no Sistema HyperProp

O Sistema HyperProp (Soares, 2000) é um sistema para autoria e formatação de documentos hipermídia baseados no modelo NCM. Ao longo dos últimos anos, a implementação do sistema passou por diversas modificações, que foram necessárias para incorporar ao sistema os refinamentos do modelo NCM e acoplar novas funcionalidades.

A primeira implementação foi baseada em um sistema de arquivos (Lima, 1992). Em seguida, foi feita uma implementação centralizada do sistema, que incorporava mecanismos para controle de versões (Soares, 1994; Batista, 1994). Mais tarde, o sistema tornou-se distribuído (Colcher, 1993), oferecendo os módulos cliente e servidor. Posteriormente, incorporou funcionalidades para autoria gráfica de estruturas aninhadas (Muchaluat-Saade, 1996a) e de relações de sincronização temporal e espacial (Costa, 1996a), onde o autor interagia com várias visões integradas (Costa, 1996b; Muchaluat-Saade, 1997). Logo após, foi implementada a primeira versão do formatador HyperProp, permitindo, finalmente, a apresentação de documentos hipermídia baseados no modelo (Rodrigues, 1997; Souza, 1997). Nessa primeira etapa, as várias implementações do sistema HyperProp foram desenvolvidas nas linguagens C e C++, onde a última implementação oferecia recursos para autoria gráfica e formatação de documentos.

Em uma etapa seguinte, partiu-se para uma nova implementação, visando a integração do sistema HyperProp à Web (Rodrigues, 1998a; Rodrigues, 1998b). Nessa segunda etapa, o sistema foi desenvolvido na linguagem Java, e o formatador reimplementado como um applet. O browser gráfico do HyperProp foi disponibilizado para a visualização de sites web (Muchaluat-Saade, 1998a; Muchaluat-Saade, 1998b), onde documentos HTML eram convertidos para documentos NCM. O editor gráfico de estruturas foi refinado (Pinto, 2000), o mecanismo de controle de versões foi revisto e o mecanismo para notificações

incorporado (Muniz, 2000). Dado o novo enfoque de integração com a Web, a primeira versão da linguagem declarativa baseada no modelo NCM, NCL 1.0, foi desenvolvida e integrada ao sistema de autoria (Antonacci, 2000a; Antonacci, 2000b). A integração de documentos SMIL 1.0 ao sistema HyperProp foi implementada (Rodrigues, 1999a; Rodrigues, 1999b; Rodrigues, 2002a) e controladores de execução para diferentes tipos de objetos de mídia foram integrados ao novo formatador (Rodrigues, 2000; Rodrigues, 2001). Posteriormente, um novo modelo para layout espacial de componentes de documentos foi desenvolvido (Moura, 2001) e o suporte para autoria cooperativa foi refinado, introduzindo mecanismos para controle de acesso e especificação de tarefas dos autores (Gorini, 2001). Em paralelo, um novo protocolo para comunicação entre os módulos cliente e servidor do HyperProp foi desenvolvido, contemplando mecanismos para provisão de QoS (Qualidade de Serviço) (Coimbra, 2001).

Em uma terceira etapa, fez-se uma reavaliação do sistema e chegou-se à conclusão de que a implementação corrente estava restringindo algumas facilidades previstas pelo modelo NCM, principalmente no que diz respeito à customização de nós, através da criação de outros atributos (chamados de propriedades), além dos predefinidos pelo modelo. Então, decidiu-se por reimplementar o sistema HyperProp, flexibilizando a criação de propriedades para as entidades do modelo NCM (Soares, 2000a). Essa terceira etapa é a que vem sendo desenvolvida atualmente. A implementação também está sendo feita em Java, só que deixando um pouco de lado a questão de integração com browsers web tradicionais, que trazia algumas limitações pelo fato de ter que lidar com applets Java (Rodrigues, 1998a).

Na implementação atual, apenas o cliente HyperProp está operacional, oferecendo funções para autoria e formatação de documentos (o servidor utilizado continua sendo o da versão anterior). A autoria pode ser realizada de forma gráfica ou de forma declarativa, seguindo a nova versão da linguagem NCL, descrita no Capítulo 3, e oferecendo as facilidades salientadas nos Capítulos 4 e 5 para definição de conectores hipermídia e templates de composição. O formatador HyperProp foi bastante refinado, oferecendo mecanismos para pré-busca de

conteúdo e adaptação de documentos durante a execução (Rodrigues, 2002b; Rodrigues, 2002c; Rodrigues, 2003).

Este capítulo se limita a discutir a implementação de conectores no sistema HyperProp, os parsers para documentos NCL 2.0 e o processador de templates de composição, o que é feito nas próximas seções. Para maiores informações sobre o restante da implementação, deve-se consultar (Muchaluat-Saade, 2003).

6.1 Conectores Hipermídia no Sistema HyperProp

Com a introdução de conectores hipermídia como novas entidades do modelo NCM, a hierarquia de classes Java que implementam as entidades do modelo no sistema HyperProp foi bastante modificada, principalmente no que diz respeito à definição de elos NCM. A Figura 34, a Figura 35, a Figura 36, e a Figura 37 ilustram os diagramas de classes para a implementação de conectores hipermídia e a Figura 38 ilustra o diagrama de classes para a implementação de elos NCM. Seguindo a modelagem orientada a objetos, usada na implementação do todo o sistema HyperProp, utilizou-se a notação UML (Rumbaugh, 1999) para representar os diagramas. Os diagramas de classes apresentados refletem os elementos e atributos dos módulos *XConnector* e *Linking* de NCL 2.0.

A Figura 34 exibe o diagrama de classes das entidades conector hipermídia e base de conectores. Ambas são entidades do modelo NCM, possuindo um identificador único. Uma base de conectores contém um ou mais conectores e um conector pertence a apenas uma base. Um conector hipermídia é constituído de um conjunto de papéis, que são detalhados na Figura 35, e um atributo *glue*. A classe conector hipermídia é especializada em conector causal e conector de restrição, sendo que o *glue* de um conector causal deve ser do tipo causal e o *glue* de um conector de restrição é do tipo restrição.

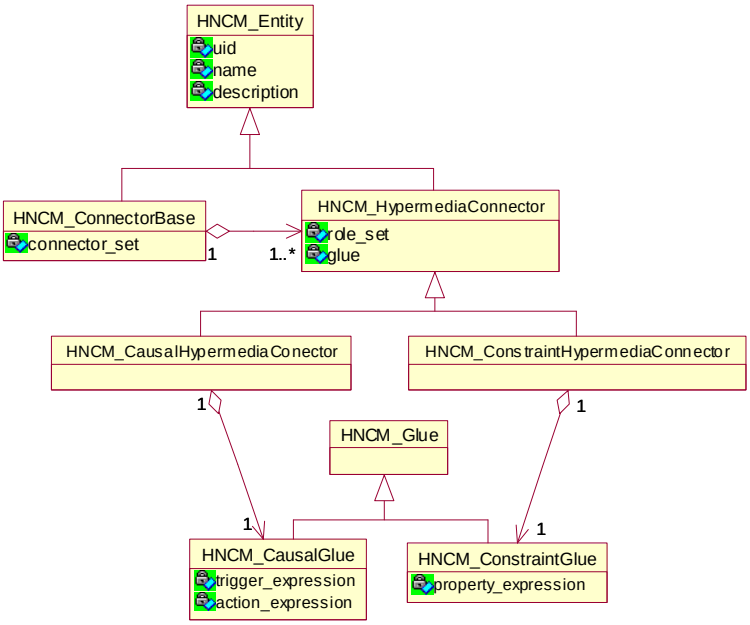

Figura 34. Diagrama de classes para entidade conector hipermídia

A Figura 35 exibe o diagrama de classes para papéis de um conector hipermídia. Um conector hipermídia causal deve possuir, no mínimo, um papel do tipo condição e um do tipo ação e um conector de restrição deve possuir, no mínimo, dois papéis do tipo propriedade. Além disso, um conector do tipo restrição só pode possuir papéis do tipo propriedade e conectores causais podem possuir papéis de qualquer tipo. Essas restrições não foram detalhadas na figura para não aumentar sua complexidade.

Um papel (classe *HNCM_Role*) possui um identificador e especifica um tipo de evento, sua cardinalidade, indicando o número mínimo e máximo de participantes que podem desempenhar o papel, e ainda um nome de atributo, usado somente para eventos de atribuição. A classe *HNCM_Role* é especializada nas classes *HNCM_ActionRole* (papel do tipo ação), *HNCM_ConditionRole* (papel do tipo condição) e *HNCM_PropertyRole* (papel do tipo propriedade).

Um papel do tipo ação indica o tipo da ação a ser executada (veja a máquina de estados de um evento exibida na Figura 12 do Capítulo 4), e atributos relacionados a execução da ação, como por exemplo, retardos, número de repetições e os valores inicial e final para atribuições. Para um detalhamento maior do significado desses atributos, deve-se consultar o Capítulo 4.

Um papel do tipo condição deve especificar uma condição, que pode ser simples ou composta. Uma condição simples avalia transições, estados e valores

de atributos do evento especificado pelo papel ou o valor do atributo especificado pelo papel. Uma condição composta é uma expressão lógica binária, baseada nos operadores AND, OR e NOT, envolvendo outras condições simples ou compostas. Quando uma condição é avaliada, durante a execução de um documento pelo formatador, ela sempre retorna um valor booleano.

Um papel do tipo propriedade possui um atributo propriedade (*role_property*) que especifica o valor do estado do evento especificado no papel; ou o instante de tempo que uma determinada transição do evento acontece; ou o valor do atributo ocorrências (*occurrences*) ou repetições (*repeat*) do evento especificado pelo papel; ou ainda o valor do atributo especificado pelo papel. Diferente de uma condição, uma propriedade pode retornar qualquer tipo de valor.

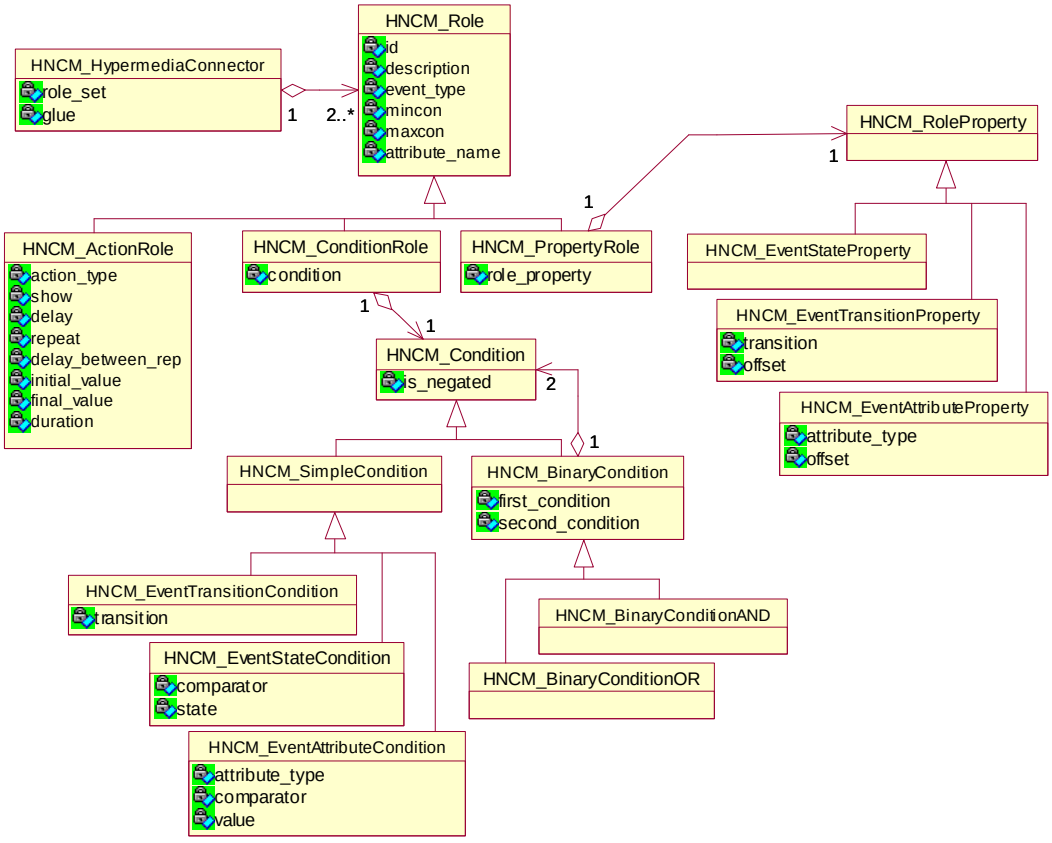

Figura 35. Diagrama de classes para papéis do tipo ação, condição e propriedade

Além do conjunto de papéis, um conector hipermídia possui um atributo *glue*, especializado de acordo com o tipo do conector. A Figura 36 exibe o diagrama de classes para um *glue* do tipo restrição. Esse tipo de *glue* possui uma expressão de propriedades, que pode ser simples ou composta. Uma expressão de

propriedades simples compara o valor retornado por um papel do tipo propriedade com outro valor do mesmo tipo ou compara os valores retornados por dois papéis do tipo propriedade distintos. Uma expressão de propriedades composta é uma expressão lógica binária, baseada nos operadores AND ou OR, envolvendo outras expressões de propriedades simples ou compostas. Note que quando uma expressão de propriedades for avaliada durante a execução de um documento pelo formatador, ela sempre retorna um valor booleano. Quando um papel do tipo propriedade, que especifica cardinalidade máxima maior que um, for referenciado pela expressão de propriedades do *glue* de restrição, ele deve ser associado a um qualificador. Os possíveis valores para qualificiadores de papéis são indicados na Tabela 13 do Capítulo 4.

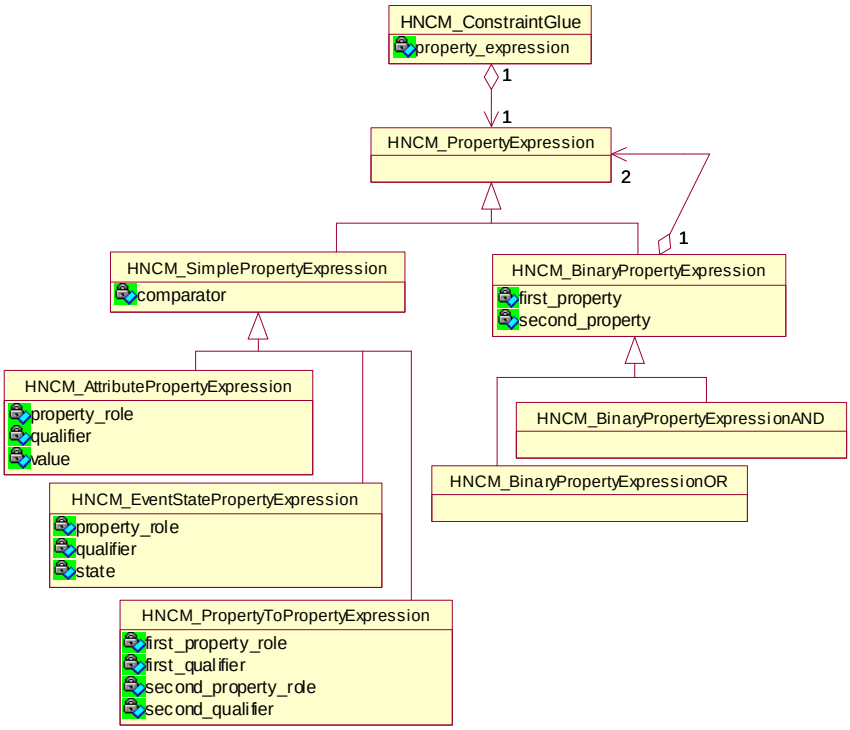

Figura 36. Diagrama de classes para *glue* do tipo restrição

A Figura 37 exibe o diagrama de classes para um *glue* do tipo causal, usado em conectores causais. Um *glue* do tipo causal possui uma expressão de disparo (*trigger_expression*) e uma expressão de ações (*action_expression*), que especificam a relação de causalidade entre os papéis do conector.

Uma expressão de disparo pode ser simples ou composta. Uma expressão de disparo simples se refere a um papel do tipo condição e especifica um qualificador, quando necessário. Uma expressão de disparo composta é uma

expressão binária, baseada nos operadores AND, OR e NOT, envolvendo duas expressões de disparo, ou uma expressão de disparo e uma expressão de propriedades (veja Figura 36) simples ou compostas. Uma expressão de disparo pode especificar um intervalo de tempo (*delay_interval*) para sua avaliação, conforme comentado no Capítulo 4. Quando uma expressão de disparo é avaliada, durante a execução de um documento pelo formatador, ela sempre retorna um valor booleano. Caso o resultado da expressão de disparo for verdadeiro, a expressão de ações deve ser executada.

Uma expressão de ações também pode ser simples ou composta. Uma expressão de ações simples se refere a um papel do tipo ação e especifica um qualificador, quando necessário. Uma expressão de ações composta é uma expressão binária, baseada nos operadores PAR, SEQ ou EXCL, envolvendo duas expressões de ações simples ou compostas. Uma expressão de ações pode especificar um retardo (*delay*) que deve ser aguardado antes de sua execução.

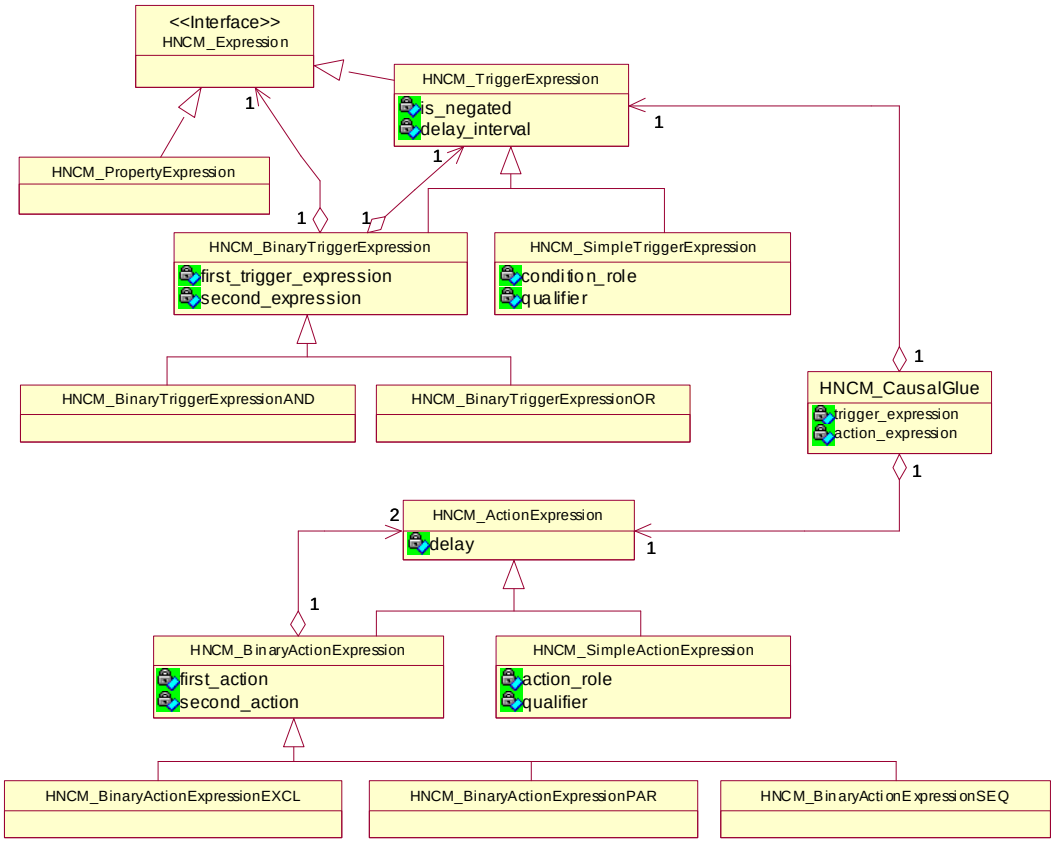

Figura 37. Diagrama de classes para *glue* do tipo causal

A Figura 38 exibe o diagrama de classes da nova definição da entidade elo no modelo NCM. Um elo é uma entidade NCM com um identificador único, que faz referência a um conector hipermídia e possui um conjunto de *binds*. A classe elo é especializada em elo causal e elo de restrição, dependendo do tipo do conector referenciado. A classe *HNCM_Bind* define as extremidades do elo, completando a definição do relacionamento. Cada *bind* associa um papel do conector referenciado, a um nó e a um de seus pontos de interface. Um *bind* pode especificar, ainda, um conjunto de descritores alternativos para o nó, permitindo que as características de apresentação de um nó sejam modificadas pelos elos que tocam o nó, conforme discutido no Capítulo 3.

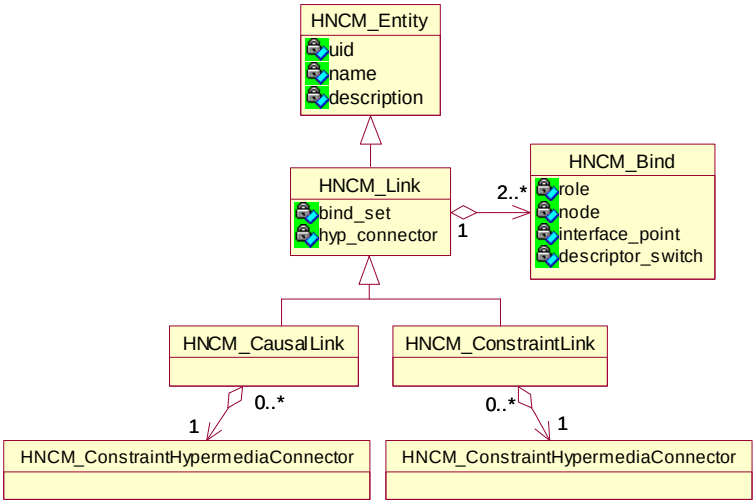

Figura 38. Diagrama de classes para entidade elo

Na implementação atual, a parametrização de valores utilizados em atributos de conectores, já disponível na linguagem XConnector, ainda não está contemplada nas classes do sistema HyperProp, sendo deixada para trabalho futuro.

A estrutura de classes apresentada é utilizada para representar os elos definidos em documentos hipermídia manipulados no subsistema de autoria do HyperProp, não importando se os documentos foram criados graficamente, ou gerados pelo procedimento declarativo e importados para o sistema através dos parsers NCL. Na implementação atual, a definição de conectores só pode ser feita de forma declarativa e a definição de elos está disponível em ambos os procedimentos.

Já no subsistema de formatação, que prepara os documentos para exibição e controla sua apresentação ao usuário final, a representação dos elos é feita de acordo com o modelo de execução, utilizado pelo formatador HyperProp, que não usa o conceito de conector. Dessa forma, ao preparar um documento para exibição, o formatador necessita convertê-lo para o modelo de execução, onde os elos NCM, que fazem referência a conectores, são traduzidos para elos do modelo de execução. A formatação de documentos no Sistema HyperProp é discutida em detalhes por (Rodrigues, 2003).

6.2
Parsers para Documentos NCL 2.0

Para permitir a autoria declarativa de documentos no sistema HyperProp, foram implementados parsers, que possibilitam a importação de documentos NCL, gerando objetos Java correspondentes às entidades NCM definidas no documento. Na implementação atual, apenas a importação de documentos NCL foi implementada, ficando para trabalho futuro a exportação de objetos NCM em Java para a representação textual em XML. O processo declarativo de autoria no sistema HyperProp está ilustrado na Figura 39.

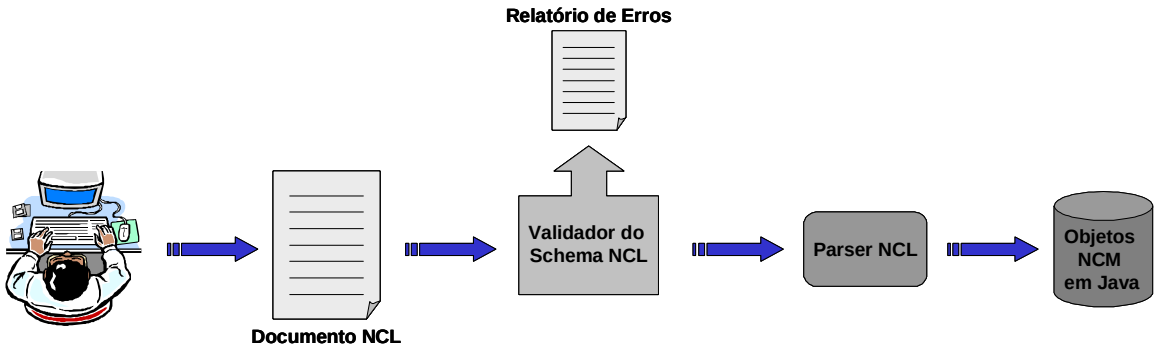

Figura 39. Processo declarativo de autoria no sistema HyperProp

A implementação do processamento de um documento NCL no sistema HyperProp foi desenvolvida usando o pacote JAXP (*Java API for XML Processing*) (JAXP, 2003), que fornece uma biblioteca com várias funções para processamento de documentos XML. Tanto o procedimento de validação do documento NCL, com base nas definições da especificação da linguagem NCL usando XML Schema (veja o Apêndice D), quanto o parser NCL, que converte o documento XML em objetos NCM, foram implementados com funções da API

JAXP. Os parsers utilizam o modelo de documentos DOM – *Document Object Model* (W3C, 1998d), padronizado pelo W3C, para manipular a árvore XML contendo os elementos do documento.

A estrutura do parser NCL é ilustrada na Figura 40. Um objeto da classe *HXML_DocParser* recebe o documento XML como entrada e instancia um objeto da classe *DocumentBuilder*, disponível no pacote JAXP. Esse objeto possui um método chamado *parse*, que recebe o mesmo documento XML como entrada e implementa o parser XML genérico, montando a árvore DOM correspondente ao documento. Essa árvore é, então, retornada através de um objeto da classe *Document*, também disponível no pacote JAXP. De posse do objeto *Document*, classe *HXML_DocParser* pode utilizar os métodos disponíveis na API JAXP para navegar na árvore do documento DOM e obter todas as informações sobre seus elementos e atributos. O primeiro passo é verificar o nome do elemento raiz da árvore. Dependendo do elemento raiz, a classe *HXML_DocParser* instancia um parser específico por tipo de documento e passa o elemento DOM como parâmetro.

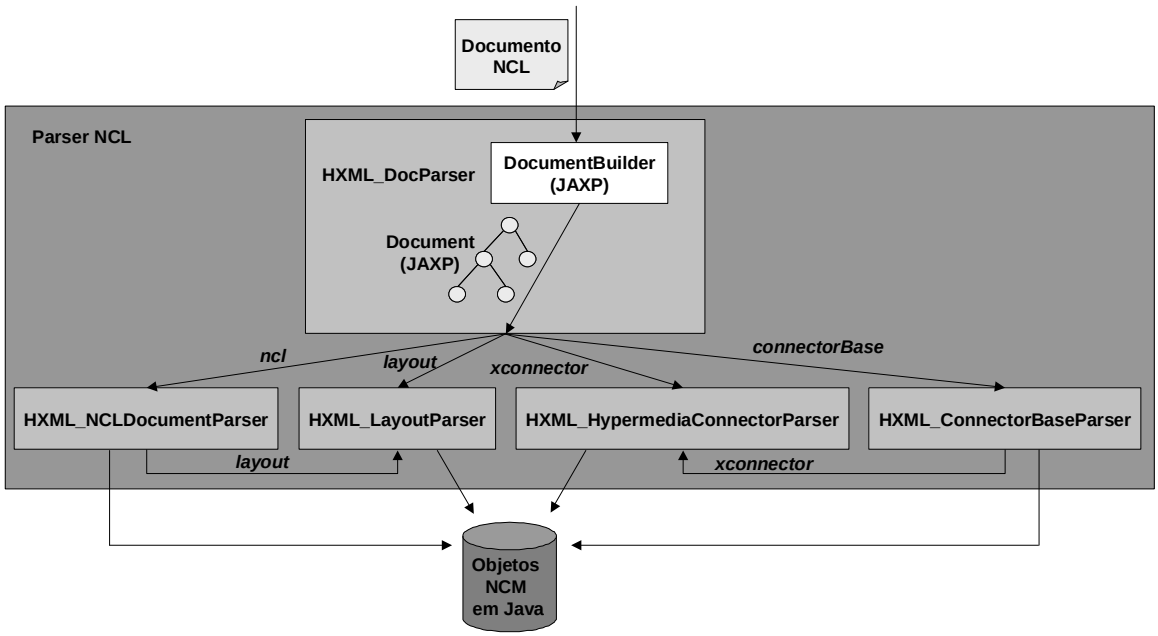

Figura 40. Estrutura do Parser NCL

O parser NCL está preparado para receber quatro tipos de documentos: um documento hipermídia NCL (elemento raiz *ncl*), estruturado de acordo com o que foi descrito no Capítulo 3; ou um documento contendo somente a descrição de um layout (elemento raiz *layout*), de acordo com o definido na Seção 3.2.8; ou um documento contendo a definição de um único conector hipermídia (elemento raiz

xconnector), ou ainda um documento contendo a definição de uma base de conectores (elemento raiz *connectorBase*), ambos descritos no Capítulo 4. As classes que implementam os parsers específicos são *HXML_NCLDocumentParser*, *HXML_LayoutParser*, *HXML_HypermediaConnectorParser* e *HXML_ConnectorBaseParser*. Os parsers específicos, por sua vez, percorrem a árvore DOM do elemento de entrada e implementam métodos para tratar cada tipo de elemento-filho e criar os objetos NCM e seus atributos correspondentes. Note que a classe *HXML_NCLDocumentParser* utiliza a classe *HXML_LayoutParser* para processar elementos *layout* do documento NCL e que a classe *HXML_ConnectorBaseParser* utiliza a classe *HXML_HypermediaConnectorParser* para processar elementos *xconnector* contidos na base de conectores.

O diagrama de atividades do método *parse* das classes *HXML_ConnectorBaseParser* e *HXML_HypermediaConnectorParser* é apresentado na Figura 41. Inicialmente, o método *parse* da classe *HXML_ConnectorBaseParser* instancia um objeto da classe *HNCM_ConnectorBase* e, depois, processa a lista de elementos *xconnector* contidos no elemento DOM *connectorBase*. Para cada conector, é instanciado um objeto da classe *HXML_HypermediaConnectorParser* e chamado seu método *parse*, passando o elemento DOM *xconnector* como parâmetro. O método *parse* da classe *HXML_HypermediaConnectorParser*, inicialmente, verifica o tipo do conector. Se for um conector causal, são processados os papéis do tipo propriedade, os papéis do tipo condição e os papéis do tipo ação. Em seguida, são processadas as expressões de disparo e de ações do *glue*. Se for um conector de restrição, são processados os papéis do tipo propriedade e, em seguida, é processada a expressão de propriedades do *glue*. O método *parse* da classe *HXML_HypermediaConnectorParser* retorna um objeto NCM da classe *HNCM_HypermediaConnector*. O controle volta, então, para o método *parse* da classe *HXML_ConnectorBaseParser*, que inclui o objeto conector NCM na base de conectores NCM e repete o processo para os outros elementos contidos no elemento DOM *connectorBase*. Finalmente, o método *parse* da classe

HXML_ConnectorBaseParser retorna um objeto NCM da classe *HNCM_ConnectorBase*.

A implementação dos outros parsers é análoga, possuindo métodos específicos para cada tipo de elemento XML. Para maiores detalhes, deve-se consultar (Muchaluat-Saade, 2003).

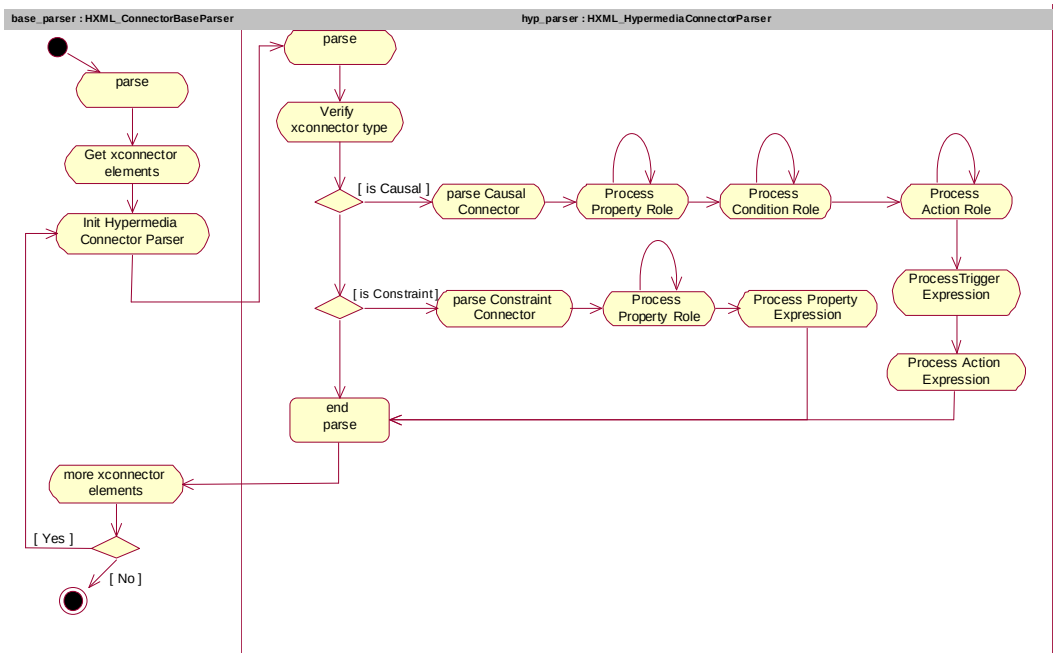

Figura 41. Diagrama de atividades do HXML_ConnectorBaseParser

6.3
Processador de Templates de Composição

Como salientado anteriormente na Seção 2.5, o conceito de template de composição foi introduzido na linguagem NCL 2.0 para dar semântica a um nó de composição. Sem comprometer a generalidade dos templates, como esta tese concentrou seus esforços na definição de relações de sincronização hipermídia, templates com semântica temporal podem ser criados através da linguagem XTemplate, comentada no Capítulo 5, fazendo uso de conectores hipermídia criados através da linguagem XConnector, comentada no Capítulo 4.

Com o uso de templates, a semântica de uma composição fica a critério da linguagem de autoria, e não precisa mais ser embutida em entidades do modelo conceitual NCM, modificando a abordagem utilizada em versões anteriores, que chegaram a introduzir os nós de composição paralelo e seqüencial no modelo

(Rodrigues, 2000; Rodrigues, 2002a). Dessa forma, as classes que implementam objetos NCM no sistema HyperProp não precisaram ser modificadas, pois o conceito de template foi introduzido apenas na linguagem NCL (versão 2.0) e não na nova versão do modelo NCM.

Para utilizar um template de composição, o autor define um documento NCL que faz referência ao template através de sua URI, como ilustrado no exemplo dado na Seção 5.4. O processamento de documentos usando templates é realizado pelo processador de templates, que gera um novo documento NCL a partir da definição do documento original e do template. O documento NCL final contém as definições de componentes e relacionamentos especificadas no template, além das definições do documento original. Depois que documento final for gerado, os parsers NCL podem ser usados para importar o documento para o sistema HyperProp e utilizar o formatador para executá-lo, como ilustrado também pelo exemplo da Seção 5.4. O uso de templates de composição, na implementação atual, é oferecido apenas no procedimento de autoria declarativo. O suporte a templates no sistema de autoria gráfico foi deixado para trabalho futuro.

A Figura 42 ilustra o processamento de documentos NCL com templates de composição. Para facilitar o entendimento do funcionamento do processador de templates, que será detalhado a seguir, sugere-se recordar a definição de um elemento *xtemplate*, discutida no Capítulo 5. Além da API JAXP, na implementação do processador de templates foi utilizado o processador XPath disponível no pacote Xalan-Java (Xalan-Java, 2003), para processar expressões XPath sobre elementos de um documento DOM.

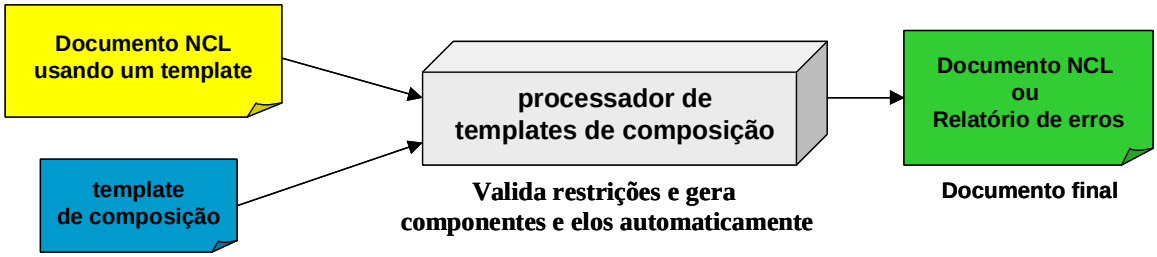

Figura 42. Processamento de documentos NCL com templates de composição

O processador de templates de composição foi implementado pela classe *HXML_TemplateProcessor*, ilustrada na Figura 43. Essa classe implementa um método público chamado *process*, que recebe como entrada a URI de um

documento NCL usando um template, e gera um novo documento NCL após o processamento. O diagrama de atividades desse método está detalhado na Figura 44. Os passos executados pelo método *process* são os seguintes: realiza o parser do documento NCL, transformando-o em um documento DOM; gera uma cópia do documento DOM, chamada de documento final (*final_doc*) no diagrama; realiza o parser do template de composição, conhecido através do atributo *xtemplate* da composição que o utiliza; chama o método *processComposite* que processa o template de composição (detalhado mais adiante), que retorna uma nova composição no formato final; substitui a composição original pela nova composição no documento final e, finalmente, serializa o documento final NCL no formato XML.

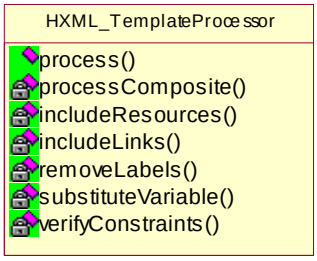

Figura 43. Classe HXML_TemplateProcessor

O método que processa o template de composição, chamado de *processComposite* está detalhado na Figura 45. Esse método recebe como parâmetros de entrada estruturas DOM contendo o documento final, a composição original e o template, e gera como saída uma nova composição no formato final. Os passos realizados por esse método são os seguintes: cria uma nova composição como cópia da original, chamada de *new_composite* no diagrama; chama o método *includeResources*, que processa elementos *resource* do template, incluindo os componentes na nova composição; chama o método *includeLinks*, que realiza o procedimento de inclusão de elos (detalhado mais adiante); chama o método *verifyConstraints*, que realiza a verificação de restrições (detalhada mais adiante); retira o atributo *label* dos elementos NCL contidos da nova composição e, finalmente, retorna como saída a nova composição, ainda como estrutura DOM.

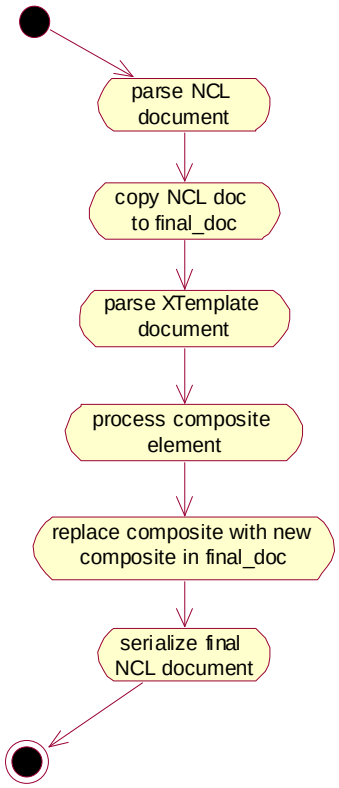

Figura 44. Diagrama de atividades do método *process*

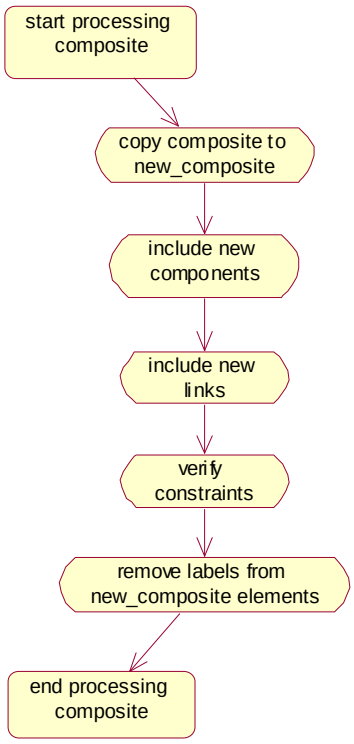

Figura 45. Diagrama de atividades do método *processComposite*

O procedimento de inclusão de elos, especificados pelo template, na nova composição sendo gerada, está detalhado na Figura 46. Esse procedimento, implementado pelo método *includeLinks*, é, sem dúvida, a parte mais complexa do processamento de um template. Inicialmente, verifica-se a existência de elementos *link* definidos diretamente no elemento *constraints* do template. Para cada elemento *link* encontrado, é criado um elemento *link* na nova composição NCL. Para criar os elementos *bind* do elo NCL, é necessário avaliar o atributo *select* de cada elemento *bind* do template, que define uma expressão XPath selecionando os componentes da nova composição, que serão as extremidades do elo sendo criado. Para cada componente selecionado pela avaliação da expressão XPath, é criado um elemento *bind* correspondente e inserido como filho do elemento *link* criado na nova composição NCL. Esse procedimento é repetido para cada elemento *bind* contido em cada elemento *link* do template.

Depois de processar elementos *link* do template, procura-se pela definição de variáveis feita no template (elementos *variable*). Caso existam, cada variável e seu valor são inseridos em um vetor, que será usado posteriormente para substituir os valores das variáveis nas expressões que as utilizarem. A implementação atual está considerando que todas as variáveis são do tipo inteiro, apesar disso não ser restrição da linguagem XTemplate.

Após o processamento das variáveis, inicia-se o processamento de instruções *for-each* definidas no template. Cada instrução *for-each* define um atributo *select*, que especifica uma expressão XPath. Essa expressão seleciona um conjunto de componentes da composição, usado para realizar a iteração para criação de novos elos. Para cada componente selecionado, é realizado o processamento dos elementos *link* contidos no elemento *for-each* do template. O processamento de cada elemento *link* é análogo ao descrito anteriormente para elementos *link* irmãos de elementos *for-each*. A diferença é que as expressões XPath, definidas nos elementos *bind* de cada *link*, podem conter extensões feitas pela linguagem XSLT à XPath, considerando, por exemplo, o uso da função *current()*. Além disso, as variáveis, declaradas no template, podem agora ser usadas dentro de cada expressão. Para a substituição de valores das variáveis nas expressões, usa-se o método *substituteVariable*, que faz a substituição de acordo com os valores armazenados no vetor de variáveis. Após o processamento dos

elos, verifica-se instruções para modificar o valor das variáveis, atualizando os novos valores adequadamente no vetor. A implementação atual está considerando que todas as intruções incrementam o valor das variáveis, apesar disso não ser restrição da linguagem XTemplate. Esses procedimentos são repetidos para cada componente selecionado por cada instrução *for-each*. Finalmente, após a execução desses procedimentos para todas as instruções *for-each*, a criação dos novos elos é concluída.

Na criação de novos componentes e elos, incluídos na nova composição NCL, seus rótulos (especificados pelo atributo *label*), correspondentes às definições do template, foram mantidos, pois eles são necessários para verificar se todas as restrições definidas pelo template são satisfeitas.

A verificação de restrições é feita pelo método *verifyConstraints*, detalhado na Figura 47. Inicialmente, os números mínimo e máximo de ocorrências para tipos de componentes especificados no template são checados analizando número de ocorrências de cada tipo na nova composição. Em seguida, um procedimento análogo é feito para conectores definidos no template e elos contidos na nova composição. Finalmente, cada elemento *constraint* do template, que define uma expressão XPath através de seu atributo *select*, é checado. Para cada restrição não satisfeita, é gerada uma mensagem de erro relatando o problema.

A implementação atual do processador de templates serviu como primeiro protótipo para testar as idéias propostas nesta tese e validar sua utilidade. Entretanto, pelo fato do processador não estar utilizando folhas de estilo XSLT (Gardner, 2002) para realizar a transformação do documento NCM original no documento final, a implementação restringe algumas facilidades contempladas na linguagem XTemplate. Essas restrições se referem à definição de variáveis, como comentado anteriormente, e ao processamento de expressões XPath que usam extensões feitas pela linguagem XSLT (W3C, 1999c).

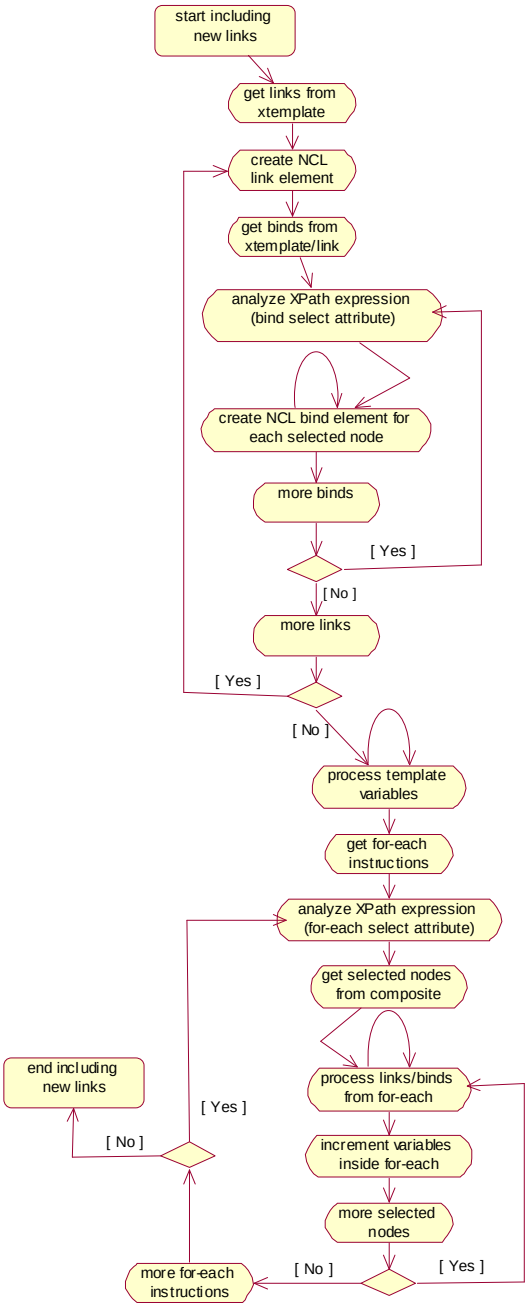

Figura 46. Diagrama de atividades do método *includeLinks*

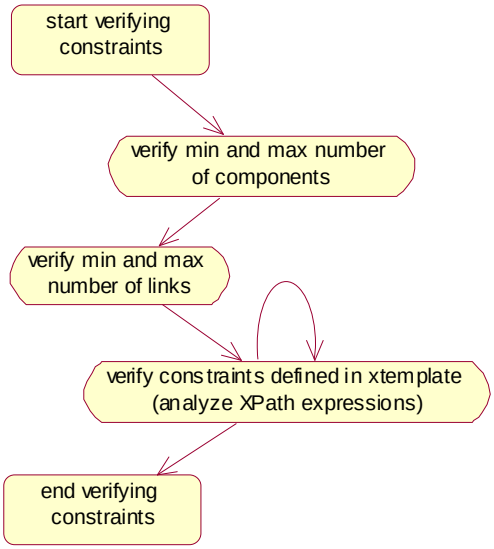

Figura 47. Diagrama de atividades do método *verifyConstraints*

O projeto de uma segunda implementação prevê o uso de um processador XSLT, para gerar o documento final, o que contemplaria o processamento de todas as facilidades da linguagem XTemplate para especificar um template de composição. Para realizar essa segunda implementação, pode-se utilizar o processador XSLT, disponível no mesmo pacote Xalan-Java, usado na implementação corrente, para avaliar expressões XPath. O processamento de templates usando folhas de estilo XSLT está ilustrado na Figura 48. O primeiro passo seria transformar o template de composição em duas folhas de estilo XSLT. Uma que especificaria todas as transformações necessárias para geração de novos componentes e elos e outra que especificaria todas as verificações de restrições no documento gerado. É necessário realizar a transformação em dois passos, já que a verificação de restrições deve ser feita considerando os novos nós e elos introduzidos pelo processamento do template.

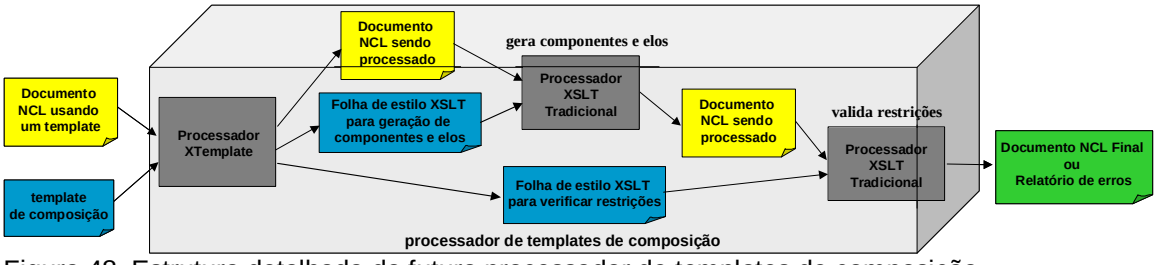

Figura 48. Estrutura detalhada do futuro processador de templates de composição

Se um mesmo documento NCL contiver mais de uma composição que use templates, o processador de templates de composição deverá realizar o

processamento de cada template de forma isolada, começando pelo processamento dos nós de composição mais internos do documento até os nós mais externos, considerando a hierarquia de composições definida no documento NCL. A implementação do processamento de documentos com mais de um template foi deixada para trabalho futuro.

Conforme comentado na Seção 5.3, templates podem ser definidos como especialização de outros templates, herdando todas as definições anteriores e ainda introduzindo novas. Entretanto, essa outra facilidade ainda não está contemplada na implementação corrente.