UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Apresentação de Conteúdos XML através de Exemplos

por

RAQUEL TRINDADE BORGES

Dissertação submetida à avaliação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, setembro de 2002

CIP – Catalogação na Publicação

Borges, Raquel Trindade
Apresentação de Documentos XML através de Exemplos / por Raquel Trindade Borges. Porto Alegre: PPGC da UFRGS, 2002.
77 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientador: Heuser, Carlos Alberto.

1. Apresentação de documentos XML em HTML, 2. XML, 3. XSLT, 4. Abordagem orientada a exemplos. I. Heuser, Carlos Alberto. II. Título.

Agradecimentos

Ao professor Dr. Carlos Alberto Heuser, meu orientador, pelo tema estimulante que propos para este trabalho e pela objetividade das suas orientações.

À Vanessa de Paula Braganholo, doutoranda e colega de grupo, por todos os conselhos e orientações na correção deste trabalho, bem como por toda atenção, incentivo e apoio.

Ao meu amado pai (em memória), que sempre me incentivou na busca de conhecimentos.

À minha mãe, ao meu marido, aos meus irmãos, aos meus filhos e sobrinhos pelo incentivo e apoio, principalmente durante a minha ausência.

A todos os colegas do MINTERCC. Em especial, às amigas: Ana Carla Macedo da Silva e Constância da Silva Santos.

À agência CAPES, pelo fomento.

À Universidade Federal do Pará e a Universidade Federal do Rio Grande do Sul, pela realização do Mestrado Interinstitucional.

Ao Instituto de Informática da UFRGS, pela utilização de suas dependências, e todo seu pessoal , sempre disposto a cooperar.

Sumário

Lista de Abreviaturas

API	Application Program Interface
DOM	Document Object Model
DTD	Document Type Definition
HTML	HyperText Markup Language
PDF	Portable Document Format
RTF	Rich Text File
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation
W3C	World Wide Web Consortium

Lista de Figuras

Lista de Tabelas

Resumo

A linguagem XSLT transforma documentos XML não apenas em novos documentos XML, mas também em documentos HTML, PDF e outros formatos, tornando-se bastante útil. Entretanto, como um ambiente de programação, XSLT apresenta algumas deficiências. Não apresenta um ambiente gráfico de programação e exige conhecimento prévio sobre manipulação de estrutura de dados em árvores, o que compromete a produtividade do programador e limita o uso da linguagem a especialistas. Assim, várias propostas têm sido apresentadas na tentativa de suprir estas deficiências, utilizando recursos variados como geração automática de *script* XSLT e reuso de transformações.

Este trabalho apresenta a ferramenta X2H que visa auxiliar a apresentação de documentos XML em HTML, aumentando a produtividade de programadores que utilizam a linguagem XSLT. Para facilitar a sua utilização, a X2H possui uma interface gráfica com abordagem baseada em exemplos, na qual o usuário compõe um documento exemplo HTML a partir de um documento fonte XML. Estes documentos são visualizados como árvores hierárquicas, nas quais é vinculado um conjunto de operações dependente do contexto, que permitem a composição do documento exemplo. Este documento serve de entrada para um gerador de regras, que gera um *script* na linguagem XSLT que, se executado, apresenta o documento HTML resultado desejado.

Palavras-chave: Apresentação de documentos XML em HTML, XML, XSLT, Abordagem orientada a exemplos

TITLE: "PRESENTATION OF XML DOCUMENTS CONTENTS BASED ON EXAMPLES"

Abstract

The language XSLT not only transforms documents XML into new documents XML, but also into HTML, PDF documents and other formats. However, as a programming environment, XSLT presents some shortcomings. It does not present a graphical environment for programming and it demands previous knowledge on manipulation of structure of data in trees. This compromises the programmer´s productivity and limits the use of the language to specialists. Thus, some proposals have been presented in the attempt to supply these shortcomings, using varied features as automatic generation of script XSLT and reuse of transformations.

This work presents the X2H tool. X2H helps the presentation of XML documents contents in HTML, increasing the productivity of programmers who use XSLT language. To facilitate its use, X2H has a graphic interface, based on examples in which the user composes an HTML example document from an XML source document. These documents are seen as a hierarchical tree, in which a set of operations is connected depending on the context. That permits the composition of the example document. These documents are the input to a rules generator that creates a XSLT script. When executed, it shows the resulting HTML document.

Keywords: Document presentation in HTML, XML, XSLT, Example-oriented approach.

1 Introdução

O comércio eletrônico e a utilização da Web têm aumentado a necessidade de troca de informações entre diferentes organizações e aplicações. Nesse contexto, XML vem se tornando um padrão no intercâmbio de documentos, pois apresenta benefícios que facilitam tais trocas. Entre eles podem-se destacar a separação dos dados de detalhes de apresentação e a transmissão de dados entre diferentes aplicações, sem a necessidade de grandes investimentos por parte das organizações em *softwares* integradores.

Mas, apesar dos benefícios trazidos pelo uso da XML, a necessidade de se converter documentos XML em outros formatos continua, pois diferentes organizações e aplicações utilizam dados em formatos e vocabulários variados. Além disso, ao se apresentar dados para serem lidos por seres humanos é necessário que estes sejam impressos ou exibidos em formatos como HTML ou PDF, etc. Por exemplo, dados em formato HTML podem ser exibidos em qualquer *browser*.

Nesse contexto, XSLT [W3C 99b] vem se destacando como uma linguagem para manipulação de dados XML, visto que possibilita a transformação de um documento XML, não só em outro documento XML, mas também em qualquer outro formato baseado em texto, como HTML, separado por vírgula, fontes C++, *script* SQL, etc.

Com a disseminação do uso da XSLT, várias deficiências foram identificadas em um ambiente de programação baseado somente nesta linguagem:

a) as transformações são expressas em formato textual;

b) exigência de conhecimento prévio de manipulação de estruturas baseadas em árvores, tornando o ambiente pouco amigável e aumentando o tempo de aprendizagem;

c) falta de associação entre documentos XML e transformações, o que impede o compartilhamento e/ou reaproveitamento de tais transformações entre documentos XML que apresentam estruturas semelhantes;

d) diversidade de finalidades, o que a torna muito extensa, para os casos em que se deseja soluções simples e/ou específicas.

Tais deficiências, além de comprometer a produtividade, têm acarretado vários problemas, tal como a limitação do uso da XSLT por parte de especialistas. Esse problema aumenta de proporção se forem levados em consideração, os ambientes atuais de programação, que utilizam interfaces gráficas com programação baseada em eventos, que os tornam mais amigáveis e intuitivos, além de proporcionarem a geração automática de código.

Isto tem motivado o desenvolvimento de diversos trabalhos, que visam suprir tais deficiências, utilizando recursos variados. Entre eles pode-se destacar: o sistema de transformação CoX (*Composition of XML –Transformations*), que proporciona o reuso de transformações já programadas para diferentes documentos XML baseados na

mesma DTD [EDE 2001]; o TIXP (*The Intregated XML Program*), um gerador de folha de estilo, que foi construído como parte de um *framework* para integração de sistemas heterogêneos [NAS 2001]; e *Visual XML Transformation* [LAU 99], que auxilia a obtenção de um novo documento XML, a partir da(s) DTD(s) de outros documentos já existentes.

1.1 Objetivo

O objetivo deste trabalho é propor uma ferramenta, que visa minimizar as deficiências identificadas em um ambiente XSLT. Para isto, foi projetada uma interface gráfica, na qual as transformações são especificadas por um usuário. Buscando tornar a ferramenta mais intuitiva, proporcionando a disseminação entre usuários não especialistas, a especificação das transformações segue a abordagem "por exemplos" proposta por [ZLO 75] com a QbE (*Query by Example*). O resultado dessa especificação é um documento exemplo, que servirá de entrada para o módulo gerador de *script* XSLT.

1.2 Organização do Texto

Este trabalho apresenta-se dividido da seguinte forma: o capítulo 2 descreve o modelo de execução da linguagem XSLT; o capítulo 3 apresenta algumas ferramentas de suporte a linguagem XSLT e realiza uma comparação entre suas principais características; capítulo 4 apresenta a X2H, ferramenta proposta neste trabalho, através de um exemplo e de um estudo de caso, no qual a interface gráfica e o módulo gerador de regras são descritos; o capítulo 5 apresenta a conclusão, onde identifica restrições apresentadas pela solução proposta e aponta trabalhos futuros.

2 A linguagem XSLT

XSLT foi criada inicialmente como parte de XSL (linguagem utilizada para expressar *stylesheets* (folhas de estilo) [W3C 2000]) para transformar um documento XML de estrutura arbitrária em um documento com sintaxe da *Formating Objects* DTD ou FO DTD. Esta sintaxe define uma coleção de elementos especializados, chamados de "objetos de formatação", utilizados para especificar detalhes relacionados à apresentação.

Durante o processo de transformação de XSLT, os elementos de um documento XML podem ser filtrados, reordenados, removidos, renomeados e novas estruturas podem ser adicionadas [W3C 99b, W3C 2000].

Devido ao poder na rearrumação de documentos XML, atualmente XSLT representa uma proposta geral para processamento de XML, sendo utilizada independente de XSL. Gera como resultado não somente outros documentos XML, mas qualquer outro formato baseado em texto, tais como: HTML, PDF, *script* SQL, etc.

Este capítulo apresenta o modelo de execução da linguagem XSLT, detalhando o modelo de dados utilizado, a entrada do processo de transformação, a estrutura de uma folha de estilo, como as transformações são aplicadas em um documento fonte para a obtenção de um documento resultado e os elementos que implementam a recursividade em XSLT.

2.1 Documentos XML como árvores

Apesar de ter documentos como entrada, as estruturas de dados manipuladas por XSLT são árvores, representações em árvores de documentos XML. Um documento XML é modelado nos seguintes tipos de nodos [KAY 2000]:

a) nodo elemento - é a parte do documento delimitado por uma *tag* de início e uma *tag* de final;

b) nodo raiz - é o elemento mais externo de um documento XML, chamado de "elemento documento";

c) nodo texto - é uma seqüência de caracteres consecutivos do tipo **PCDATA,** parte de um elemento**;**

d) nodo comentário - representa um comentário escrito em um documento fonte XML, delimitado por <!-- e -- >;

e) nodo instrução de processamento - representa uma instrução de processamento escrita em um documento XML, delimitado por <? e ?>;

f) nodo n*amespace* - representa uma declaração *namespace.*

Estes nodos também podem ser classificados, levando-se em consideração algumas características, tais como:

a) os nodos que têm filhos - raiz e elementos;

b) os nodos que têm pai - todos, exceto a raiz;

c) os nodos que têm nome - elementos, atributos, *namespaces* e instruções de processamento;

d) os nodos que têm conteúdo textual - atributos, textos, comentários, instruções de processamento e nodos *namespace*.

2.2 Entrada do Processo de Transformação

O processo de transformação da XSLT recebe como entrada dois documentos XML: um documento fonte, em que serão aplicadas as transformações e uma folha de estilo, na qual essas transformações são descritas.

2.2.1 Documento fonte

O documento fonte deve ser XML. XSLT só trabalha com documentos XML bem formados [W3C 98b]. A seguir são apresentados um exemplo de um documento fonte XML em formato textual e seu modelo em árvore (figura 2.1), com a discriminação de alguns tipos de nodos mencionados na seção 2.1.

Documento XML exemplo em formato textual

```
<listaalunos>
 <aluno id="195/00">
  <curso>medicina</curso>
  <turno>
  <!—manhã/tarde/noite -->
   manhã
  </turno>
  <nome>Roberto Andrade</nome>
 </aluno>
 <aluno id="172/99">
  <curso>direito</curso>
  <turno>
  <!—tarde -->
   tarde
  </turno>
  <nome>Kamily Marinho</nome>
  </aluno>
</listaalunos>
```

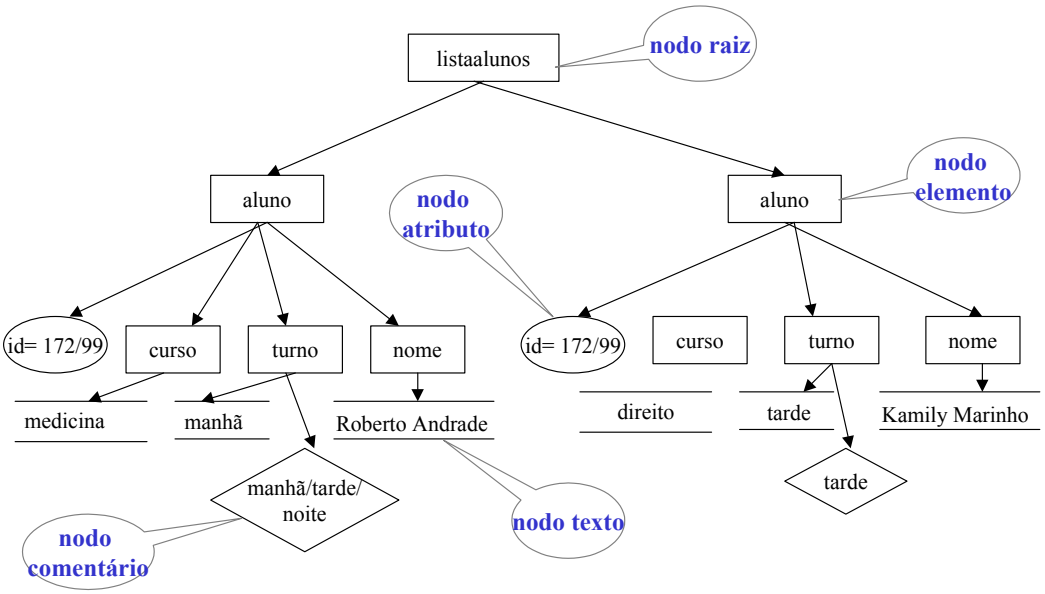


FIGURA 2.1 - Modelo em Árvore de um Documento XML

2.2.2 Folha de estilo

A folha de estilo tem como raiz um elemento `<xsl:stylesheet>` ou `<xsl:transform>` e é composta por um conjunto de regras chamadas de regras *template*.

Regras templates

Uma regra *template* em uma folha de estilo é um elemento `<xsl:template>` com um atributo *match,* que é comparado a um "padrão," que define a quais nodos do documento fonte a regra se aplica. O conteúdo do elemento `<xsl:template>` é chamado de corpo do *template* [KAY 2000]. O formato geral de uma regra *template* é mostrado a seguir:

```
<xsl:template match= padrão priority = valor numérico>
 corpo do template
</xsl:template>
```

Padrões

Um "padrão" seleciona um conjunto de nodos no documento fonte. A XSLT utiliza um subconjunto da linguagem XPath [W3C 99a], mais precisamente os l*ocation paths*, para expressar padrões. O resultado da avaliação de l*ocation paths*, utilizados como padrões, deve ser um conjunto de nodos. Um padrão é um conjunto de *location paths* separados por uma "|" . Um *location path* é composto de três partes:

- um eixo - não é parte obrigatória de um *location path*. Especifica o relacionamento hierárquico entre os nodos selecionados e o nodo corrente. Os tipos de eixo que podem ser utilizados em *location paths* para padrões são *child* e *attribute*, que indicam respectivamente, os filhos imediatos e os atributos do nodo corrente. Podem, também ser utilizados os operadores "/" e "//", que indicam eixos do tipo `descendant-or-self` ;

- um nodo teste - especifica o tipo e nome dos nodos selecionados;
- zero ou mais predicados - expressões, utilizadas para filtrar o resultado obtido pela avaliação do *location path*.

A sintaxe de um *location path* é o eixo seguido do nodo teste, separados por um duplo dois pontos, seguido por zero ou mais predicados dentro de colchetes. Por exemplo, em `child::autor[position()=1]`, `child` é o eixo, `autor` é o nodo teste e `[position()=1]` é o predicado. A tabela 2.1 contém alguns exemplos de padrões:

TABELA 2.1- Exemplos de padrões

Padrões	Significado
/	seleciona o nodo raiz
autor	seleciona qualquer elemento <autor>
livro/autor	seleciona qualquer elemento <autor> cujo o pai é um elemento <livro>
livro[ano-publicação=”2001”]	seleciona qualquer elemento livro que tem um elemento filho <ano-publicação>, cujo o valor é igual a 2001
livro/autor[1]	seleciona qualquer elemento <autor> que é o primeiro filho de um elemento <livro>

Resolução de conflitos

O atributo opcional `priority` de um elemento `<xsl:template>` é utilizado para resolver conflitos causados quando existe mais de uma regra que “casa” com o mesmo conjunto de nodos em uma folha de estilo . Esses critérios são descritos em [W3C 99b] e podem ser resumidos da seguinte maneira:

- se o elemento `<xsl:template>` possui o atributo `priority`, será considerada a regra com o valor de `priority` mais alto;
- se as regras consideradas não possuem o atributo `priority`, serão considerados os valores atribuídos a `priority` segundo os seguintes critérios:
  a) regras cujo padrão é somente o nome do elemento, recebem `priority ='0';`
  b) regras mais específicas, o valor de `priority` é `'0.5';`
  c) regras genéricas, cujos padrões são: “*”, “text()” e “comment()”, recebem `priority=-'0.5'`.

Assim, por exemplo, para decidir entre uma regra cujo padrão é "`autor`" e outra com padrão "`livro/autor`", a segunda é considerada mais apropriada

Corpo do *Template*

O conteúdo de um corpo do *template* pode ser classificado como dados ou como instruções. Os textos são classificados como dados e as instruções são identificadas através do prefixo “xsl”. Quando uma regra *template* é instanciada, as instruções são executadas e os dados são copiados para a árvore resultado[KAY 2000]. No exemplo a seguir, as *tags* `<livro>` e `</livro>` são copiadas para a árvore resultado e a instrução `<xsl:apply-templates/>` é executada.

```
<xsl:template match=”book”>
 <livro>
 <xsl:apply-templates/>
 </livro>
</xsl:templates>
```

Regras *Defaults*

Existem também regras *templates* implícitas em uma folha de estilo, chamadas de regras *default* (tabela 2.2), que são executadas caso não existam regras explícitas para um determinado tipo de nodo selecionado para processamento. A primeira delas é válida para os nodos raiz e elementos, e causa o processamento de todos os nodos filhos. A segunda regra *default* é válida para os nodos do tipo texto e atributo, e causa a cópia dos valores desses nodos para saída. A regra *default* para comentários e instruções de processamento não causa nenhum efeito.

As duas primeiras regras *default* em uma folha de estilo vazia, somente com o elemento `<xsl:stylesheet>`, geram na saída um documento com todo o conteúdo do documento fonte XML.

TABELA 2.2- Regras *default*

Tipos de nodos	Regra *default*	Efeito
Raiz e elementos	<xsl:template match="*|/"> <xsl:apply-templates/> </xsl:template>	causa o processamento recursivo de todos os nodos elementos
Textos e atributos	<xsl:template match="text()|@*"> <xsl:value-of select="."/> </xsl:template>	copia os valores dos nodos atributos e textos para a saída
Comentários e instruções de processamento	`<xsl:template match="processing-instruction()|comment()"/>`	sem efeito

2.3 Processo de transformação

Como mencionado, as estruturas de dados manipuladas por um processador XSLT são árvores. Portanto, antes do processamento das transformações é necessária a validação dos documentos XML de entrada: documento fonte, folha de estilo e a construção das árvores desses documentos. Essas funções são realizadas por um *parser* XML .

Após o processo de transformação a árvore resultado é serializada para a obtenção do documento resultado. Este processo de transformação é ilustrado através da figura 2.2.

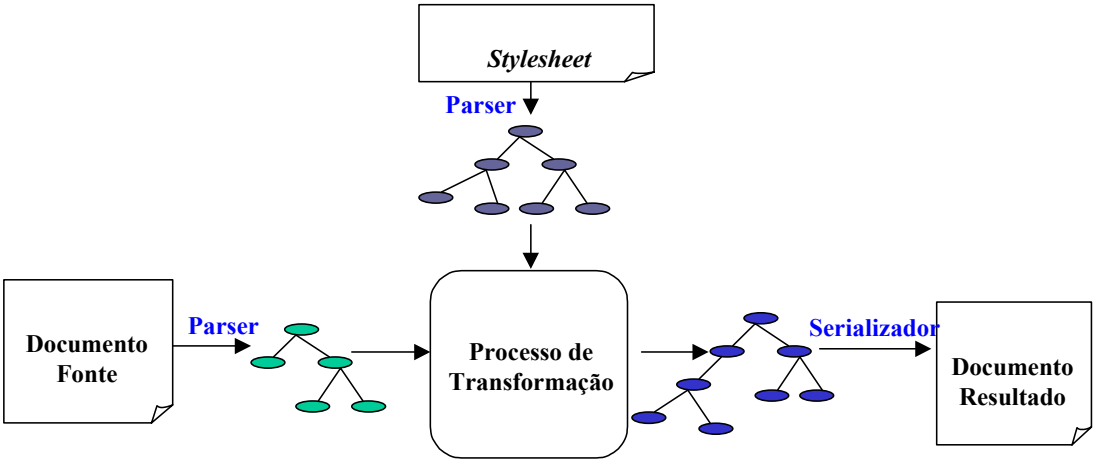


FIGURA 2.2 - Processo de transformação XSLT

2.3.1 Passos de transformação

De posse das árvores dos documentos fonte e folha de estilo, o processador deve executar os seguintes passos para obter o documento resultado [KAY 2000]:

a) constrói a árvore de transformação (figura 2.3), incluindo um nodo raiz acima das árvores dos documentos fonte e folha de estilo;

b) inicializa o nodo corrente como o nodo raiz;

c) pesquisa regras na folha de estilo para o nodo corrente;

d) se não existir regra, **executa regra *default*** para o tipo do nodo corrente, conforme definido em **regras *defaults*** da seção 2.2.2; se existir uma regra, **executa regra;** se existirem mais de uma regra **resolve conflito** conforme critérios definidos em **resolução de conflitos** da seção 2.2.2 e **executa regra**;

e) instância *template,* gerando dados para a construção de parte da árvore resultado;

f) verifica a seleção de nodos filhos do nodo corrente para processamento;

g) se não existem mais nodos para processamento, **finaliza processamento**. Se mais nodos forem selecionados, **nodo selecionado passa a ser o nodo corrente**;

h) se nodos selecionados através do elemento `<xsl:apply-templates>` **vai para o passo c**. Se a seleção ocorrer com o uso do elemento `<xsl:for-each>`, **vai para o passo e**.

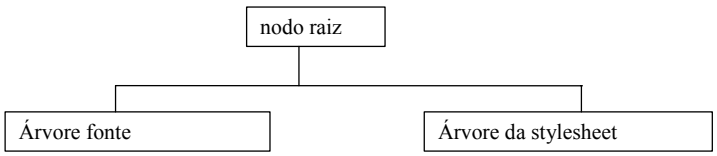


FIGURA 2.3 - Árvore de transformação XSLT

2.3.2 Um exemplo de transformação

Para ilustrar o processo de transformação descrito na seção anterior, serão considerados os documentos fonte, folha de estilo, descrição dos passos de transformação, documento resultado e a figura 2.4, apresentados a seguir.

documento fonte books.xml [HEU 2000]:

```
<?xml version='1.0'?>
<booklist>
 <book id="BOX00">
  <author>Box, D. and Skonnard, A. and Lam, </author>
  <editor>Series</editor>
  <title>Essential XML - Beyond Markup</title>
  <publisher>Addison-Wesley</publisher>
  <year>2000</year>
  <key></key>
  <volume></volume>
  <number></number>
  <series></series>
  <address></address>
  <edition></edition>
  <month>July</month>
 </book>
</booklist>
```

Folha de estilo exemplo

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
 <xsl:output indent="yes"/>

 <xsl:template match="/">
  <xsl:apply-templates/>
 </xsl:template>

 <xsl:template match="text()">

 </xsl:template>

 <xsl:template match="booklist">
  <ListaLivros>
   <xsl:apply-templates/>
  </ListaLivros>
 </xsl:template>

 <!-- Book -->
 <xsl:template match="book">
  <Livro>
   <xsl:apply-templates/>
  </Livro>
 </xsl:template>

 <!-- Author -->
 <xsl:template match="author">
  <Autor>
   <xsl:value-of select="."/>
  </Autor>
 </xsl:template>
```

1

2

3

4

5

```
<!-- Title -->
<xsl:template match="title">
 <Titulo>
  <xsl:value-of select="."/>
 </Titulo>
</xsl:template>
```

6

Descrição dos passos de transformação

a) são montadas as árvores dos documentos: books.xml e folha de estilo *(*figura 2.4);

b) é construída a árvore de transformação com a inclusão de um nodo raiz acima das árvores dos documentos;

c) o nodo raiz é comparado com todos os padrões de todas regras na folha de estilo, ele "casa" com regra "1". O elemento `<xsl:apply-templates>` causa o processamento de todos os filhos do nodo raiz da árvore, que nesse caso são: a folha de estilo e o elemento `<booklist>`;

d) é pesquisada na folha de estilo uma regra para o elemento `<xsl:stylesheet>`, nenhuma regra é encontrada e nenhuma saída é gerada;

e) o segundo filho do nodo raiz é o elemento `<booklist>`, ele casa com a regra "3". A *tag* `<ListaLivros>` é gerada. O elemento `<xsl:apply-templates>` causa o processamento dos filhos do elemento `<booklist>`, que é o elemento `<book>`;

f) o elemento `<book>` é processado (regra "4"). A *tag* `<livro>` é escrita no resultado e o elemento `<xsl:apply-templates>` causa o processamento de todos os filhos de `<book>`;

g) para cada filho de `<book>` são pesquisadas regras na folha de estilo. Somente são encontradas regras para os elementos: `<author>` e `<title>` ("5" e "6"), gerando as saídas das *tags* `<Autor>` e `<Titulo>` respectivamente, com a cópia de seus valores para a saída através da execução da instrução `<xsl:value-of select="./">`.

Documento resultado da transformação

Após a aplicação da folha de estilo sobre o documento fonte, o seguinte documento resultado é obtido:

```
<ListaLivros>
 <Livro>
  <Autor>Box, D. and Skonnard, A. and Lam, </Autor>
  <Titulo>Essential XML - Beyond Markup</Titulo>
 </Livro>
</ListaLivros>
```

Ilustração dos passos dos passos de transformação para o exemplo apresentado

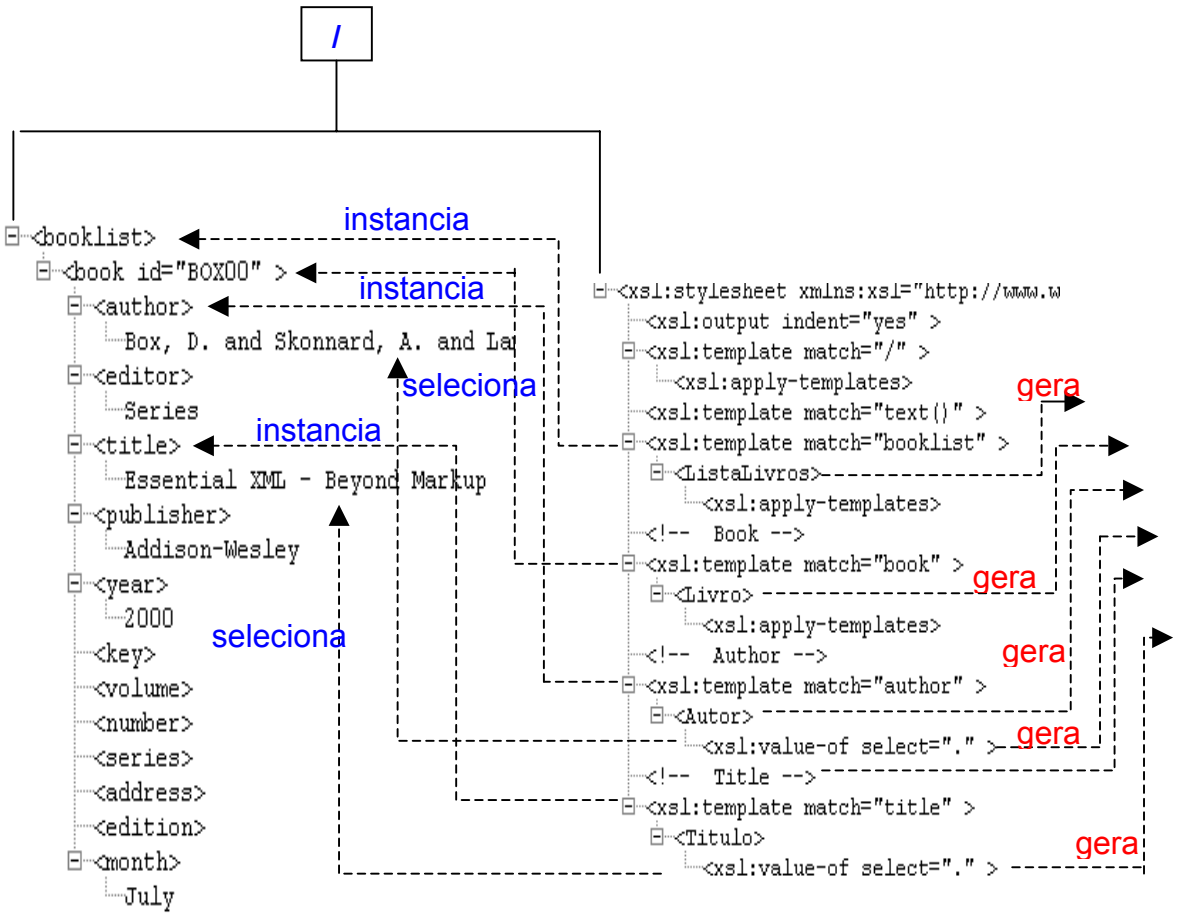


FIGURA 2.4 - Ilustração dos passos para transformação de um documento fonte

2.4 Recursividade em XSLT

Dois elementos XSLT implementam a recursividade do processo de transformação de um documento fonte XML, definindo assim, dois estilos de escrita de regras em XSLT. Um dos estilos utiliza o elemento `<xsl:apply-templates>` e o outro o elemento `<xsl:for-each>`.

Segundo [KAY 2000], escrever regras XSLT de uma maneira ou de outra, é uma questão de estilo pessoal. Os que utilizam `<xsl:apply-templates>` argumentam que o uso desse elemento deixa a folha de estilo menos "presa" a detalhes da estrutura do documento fonte, o que torna a escrita mais fácil e mais livre de contexto. Os que preferem o elemento `<xsl:for-each>`, dizem que este torna a lógica mais clara para o leitor e também pode aumentar a performance, uma vez que não há necessidade de identificar regras *templates* que "casem" com padrões, embora isto tenha um efeito muito pequeno.

Conclui-se, portanto, que as duas maneiras são equivalentes. A seguir, são apresentadas as folhas de estilo 1 e 2, que ilustram essas duas formas de escrita de XSLT, para a obtenção do mesmo documento resultado a partir do documento fonte PUBLICAÇAO.XML.

Documento fonte: PUBLICAÇAO.XML

```
<?xml version="1.0"?>
<PUBLICACOES>
 <LIVRO>
  <TITULO>A Certain Justice</TITULO>
  <AUTOR>P.D. James</AUTOR>
  <ANO-PUBLICACAO>1998</ANO-PUBLICACAO>
  <ISBN>0375401091</ISBN>
 </LIVRO>
 <ARTIGO>
  <TITULO>What kind of language is XSLT?</TITULO>
  <AUTOR>Michael Kay</AUTOR>
  <ANO-PUBLICACAO>2001</ANO-PUBLICACAO>
 </ARTIGO>
 <LIVRO>
  <TITULO>Ashworth Hall</TITULO>
  <AUTOR>Anne Perry</AUTOR>
  <ANO-PUBLICACAO>1997</ANO-PUBLICACAO>
  <ISBN>0449908445</ISBN>
 </LIVRO>
 <ARTIGO>
  <TITULO>Query by Example</TITULO>
  <AUTOR> ZLOOF M </AUTOR>
  <ANO-PUBLICACAO>1975</ANO-PUBLICACAO>
 </ARTIGO>
 <LIVRO>
  <TITULO>XSLT Programer's Reference</TITULO>
  <AUTOR>Michael Kay</AUTOR>
  <ANO-PUBLICACAO>2000</ANO-PUBLICACAO>
  <ISBN>0446674249</ISBN>
 </LIVRO>
 <LIVRO>
  <TITULO>Shadow Woman</TITULO>
  <AUTOR>Thomas Perry</AUTOR>
  <ANO-PUBLICACAO>1997</ANO-PUBLICACAO>
  <ISBN>0679453024</ISBN>
 </LIVRO>
</PUBLICACOES>
```

folha de estilo 1: exemplo de regras com o elemento `<xsl:apply-templates>`

```
<?xml version='1.0'?>
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/XSL/Transform/1.0
'>

<xsl:template match="PUBLICACOES">
  <xsl:apply-templates select="LIVRO"/>
  <xsl:apply-templates select="ARTIGO"/>
</xsl:template>

<xsl:template match="LIVRO">
  <xsl:value-of select="AUTOR"/>
  <xsl:value-of select="TITULO"/>
</xsl:template>
```

os nodos filhos são selecionados para processamento através do elemento `<xsl:apply-templates>`

é escrito um elemento `<xsl:template>` para cada conjunto de nodos selecionado através do elemento `<xsl:apply-templates>`

```
<xsl:template match="ARTIGO">
  <xsl:value-of select="AUTOR"/>
</xsl:template>

</xsl:stylesheet>
```

folha de estilo 2: exemplo de regras com o elemento <*xsl:for-each*>

```
<?xml version='1.0'?>
<xsl:stylesheet
xmlns:xsl='http://www.w3.org/XSL/Transform/1.0
'>

<xsl:template match="PUBLICACOES">
  <xsl:for-each select="LIVRO">
    <xsl:for-each select="AUTOR">
     <xsl:value-of select="."/>
    </xsl:for-each>
    <xsl:for-each select="TITULO">
     <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:for-each>
  <xsl:for-each select="ARTIGO">
    <xsl:for-each select="AUTOR">
     <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

os nodos filhos são selecionados para processamento através de elementos `<xsl:fo-each>` aninhados de acordo com a estrutura do documento

os *templates* de saída são escritos na corpo do elemento `<xsl:for-each>` correspondente, sem elementos `<xsl:templates>` adicionais

2.5 Documento Resultado

Em XSLT, o formato de saída do documento resultado é definido através do elemento `<xsl:output>`, com atributo `method` comparado ao formato de saída desejado. Existem três formatos de saída padrão em XSLT: XML, HTML ou TEXT. O método XML gera um documento XML na saída. Se o atributo `method` for comparado a `html`, o processador XSLT poderá reconhecer as várias convenções e estruturas HTML. O formato TEXT pode ser utilizado quando se deseja uma saída com qualquer outro formato baseado em texto como, por exemplo: RTF, PDF , *Script* SQL, etc.

2.6 Conclusão

Apesar de todo o poder de expressão da XSLT na transformação e manipulação de documentos XML, a programação nesta linguagem não é uma tarefa trivial, uma vez que exige um conhecimento prévio sobre manipulação e navegação de estruturas em árvore. Além disso, não oferece um ambiente que facilite e/ou agilize a tarefa de programação. Tais dificuldades têm motivado o desenvolvimento de ferramentas, que procuram suprir essas deficiências, proporcionando o aumento da produtividade de

programação nesta linguagem. No capítulo seguinte serão descritos alguns exemplos destas ferramentas.

3 Ferramentas de suporte a XSLT

XSLT tem se tornado um padrão para transformação de documentos XML. Porém, um ambiente de programação baseado somente nessa linguagem não oferece facilidades de programação. As transformações são expressas em formato textual, e exigem um certo conhecimento de navegação e manipulação das estruturas de dados em árvores. Essas dificuldades elevam a curva de aprendizagem e comprometem a produtividade de programação. Tais deficiências têm motivado o desenvolvimento de ferramentas de suporte a XSLT. Geralmente são ferramentas baseadas em interfaces gráficas, que procuram aumentar a produtividade utilizando alguns recursos. Entre estes, pode-se destacar: geração de *script* XSLT, a partir de especificações realizadas por usuários; reuso de transformações já programadas e/ou restrição da expressividade da XSLT, para satisfazer as necessidades de um ambiente específico.

Três propostas relacionadas com este tema serão apresentadas neste capítulo. A primeira é o sistema de transformação CoX(*Composition of XML –Transformations*), que tem como idéia principal, o reuso de transformações para diferentes documentos XML baseados na mesma DTD [EDE 2001]. A segunda é o TIXP (*The Intregated XML Program*), um gerador de folha de estilo, que foi construído como parte de um *framework* para integração de sistemas heterogêneos [NAS 2001]. A terceira proposta, é a ferramenta *Visual XML Transformation* [LAU 99], que faz parte de um pacote de ferramentas visuais da IBM para XML. Essa ferramenta auxilia a obtenção de um documento resultado, a partir da(s) DTD(s) de outros documentos já existentes.

3.1 CoX (Composition of XML –Transformations)

O sistema CoX [EDE 2001] aponta uma deficiência em um ambiente padrão de transformação XSLT: a ausência de associação entre um documento XML e um programa de transformação. Para solucionar este problema, o CoX propõe um sistema de transformação no qual as transformações XSLT são associadas às DTDs dos documentos fonte e alvo. Estas informações são utilizadas para encontrar e aplicar transformações apropriadas para instâncias específicas de documentos. O CoX também utiliza os conceitos de especialização e composição de documentos.

Com o objetivo de aumentar a eficiência de programação em XSLT, transformações já desenvolvidas são reusadas. Isto é baseado no fato de que uma DTD pode validar vários documentos XML, permitindo assim, o reuso das transformações para as estruturas compartilhadas por diferentes documentos XML, que possuem a mesma DTD.

O CoX utiliza um meta-esquema no qual são armazenadas as informações sobre as associações, composições e especializações de documentos e tipos de documentos.

3.1.1 Composição de documentos XML

No meta-esquema CoX (figura 3.1), um documento pode ser visto como uma agregação de componentes, o que é representado através do relacionamento de

agregação *consist_of*. Cada componente é associado a uma única DTD (relacionamento *has_DTD)* e possui exatamente um elemento raiz. Essas duas informações determinam o "tipo base" de um componente. As especializações são representadas pelo auto-relacionamento *is_a* entre componentes. Os relacionamentos *from* e *to* determinam as transformações programadas para um determinado componente.

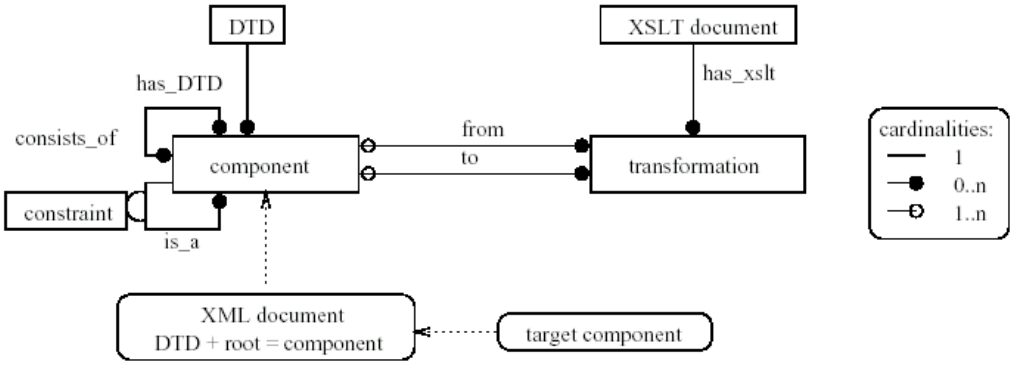


FIGURA 3.1 - Meta-esquema CoX [EDE 2001]

3.1.2 Especialização de componentes

Para a especialização de um componente são consideradas as seguintes restrições:

- por ambiente - usada para diferenciar elementos de mesmo tipo, mas com semântica diferente;
- por estrutura - onde uma DTD mais específica é utilizada para substituir uma DTD muito genérica, reduzindo o conjunto de documentos válidos;
- por instância - restringe possíveis instâncias através da restrição de valores para determinados elementos.

No meta-esquema CoX, os três tipos de especialização são representadas de maneira uniforme: um componente "A" é uma especialização de um componente "B", se todo documento válido para "A" é também válido para "B" e a *constraint* de especialização é satisfeita.

Como mencionado, as especializações de componentes são representadas pelo relacionamento *is_a.* Todas as associações herdadas, exceto *has_DTD,* podem ser sobrescritas pelo componente especializado. Dessa maneira, o "tipo base" de uma hierarquia de especialização é a mesma para todos os componentes de um mesmo tipo de documento.

Em uma hierarquia de especialização, a raiz é o componente mais genérico com a transformação mais genérica. A cada especialização, uma restrição é adicionada às herdadas de componentes antecessores.

As restrições de especialização são expressas através da linguagem XPath. O teste para verificar as restrições de especialização é realizado através da comparação entre o resultado da expressão XPath e o elemento raiz do componente especializado.

3.1.3 Transformação de um documento

Considerando que as informações sobre os documentos fonte e alvo, necessárias para a transformação estejam armazenadas no meta-esquema CoX, os seguintes passos são realizados na transformação de um documento fonte para a obtenção de um documento alvo:

- seleção do documento fonte e tipo de documento alvo na interface gráfica da ferramenta;
- a partir da DTD e do elemento raiz, o sistema determina o componente principal do documento fonte. Com a identificação dos tipos de fonte e alvo, CoX aplicará somente as transformações que levem a DTD do alvo;
- decomposição do documento fonte através da pesquisa do relacionamento *is_a* e identificação dos seus sub-componentes especializados, pela checagem das *constraints* de especialização;
- definição do escopo do componente pela extração de sua subárvore XML. Dessa maneira, cada componente pode ser transformado como um documento separado;
- através da pesquisa da associação *consist_of*, são identificados os menores componentes para os quais uma transformação para a DTD do alvo foi registrada. Para cada um desses componentes, a hierarquia de especialização é pesquisada para a identificação de subtipos dinâmicos;
- a transformação do componente ocorre quando a sua relação *consist_of* é vazia ou se todos os seus sub-componentes já foram transformados;
- através dos relacionamentos *from* e *to* é aplicada a transformação apropriada ao componente;
- após a transformação de todos os componentes, estes serão utilizados na composição do documento resultado.

3.2 TIXP (*The Integrated XML Program*)

A ferramenta TIXP [NAS 2001] é um gerador de folha de estilo, que foi construído como parte de um *framework* para integração de sistemas heterogêneos. Foi desenvolvido em um ambiente que utiliza o formato XML como padrão para representação de dados e um processador XSLT, para executar as transformações necessárias durante o processo de troca de informações entre diferentes aplicações.

O desenvolvimento deste gerador de folha de estilo foi motivado pelo fato de que, com a utilização da XSLT, a funcionalidade de adaptação dos dados integrados que foi retirada do interior dos programas de aplicação, passando a ser desenvolvida pelos programadores de folhas de estilo.

No TIXP, o programador XSLT especifica as transformações de um documento XML através de uma interface gráfica. Nessa interface, o programador pode especificar dois tipos de transformações no documento fonte: tradução de estruturas e mapeamento de valores.

3.2.1 Interface gráfica

Na interface gráfica do TIXP (figura 3.2), a especificação de uma transformação começa com a seleção de um documento XML em que se deseja realizar transformações; este documento é apresentado no formato de uma árvore. Após a seleção do primeiro documento, o programador seleciona um segundo documento XML que servirá de meta e que é apresentado no mesmo formato do documento fonte. Utilizando as árvores desses documentos, o programador poderá especificar as transformações necessárias.

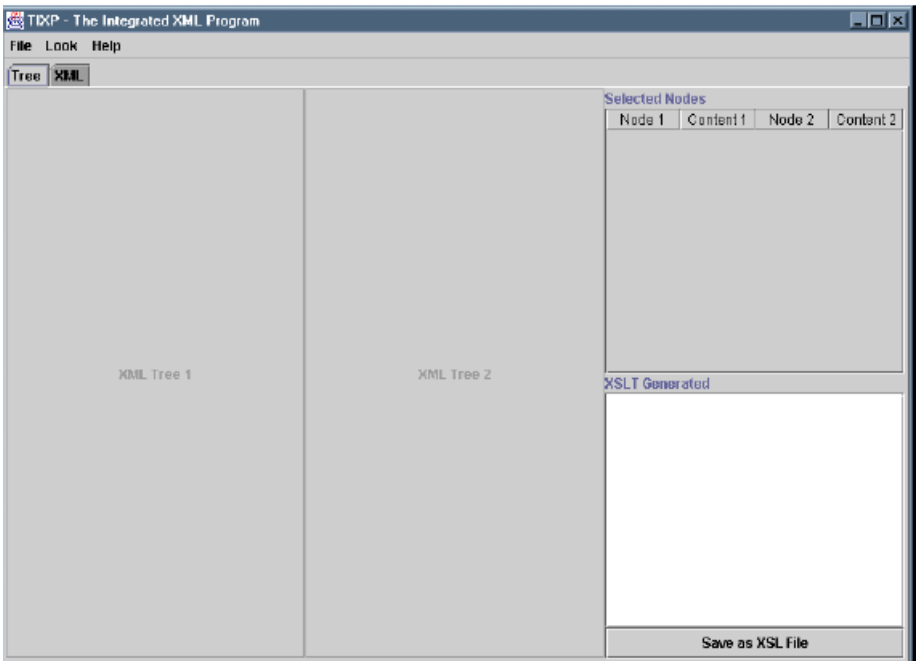


FIGURA 3.2 - Interface gráfica do TIXP

3.2.2 Especificação de Transformações

As transformações no TIXP são realizadas através de mapeamentos entre os nodos da árvore do primeiro documento aos nodos da árvore do segundo documento. Os primeiros nodos que devem ser mapeados são os nodos raiz dos dois documentos. As transformações do TIXP são classificadas em dois tipos: mudanças estruturais (mudança de vocabulário) e mudanças de conteúdo.

As transformações estruturais são realizadas através de mapeamentos dos nodos do documento fonte para os nodos do segundo documento (figura 3.3). Cada mapeamento realizado é inserido na tabela "*Selected Nodes*", que contém as colunas "*Node 1*", "*Content 1*", "*Node 2*" e "*Content 2*". Os nodos do documento fonte são inseridos nesta tabela através da operação *Insert* gerando o preenchimento das colunas "*Node 1*" e "*Content 1*". Já os nodos do documento meta são inseridos nas coluna "*Node 2*" e "*Content 2*" através da operação *Set/Change*.

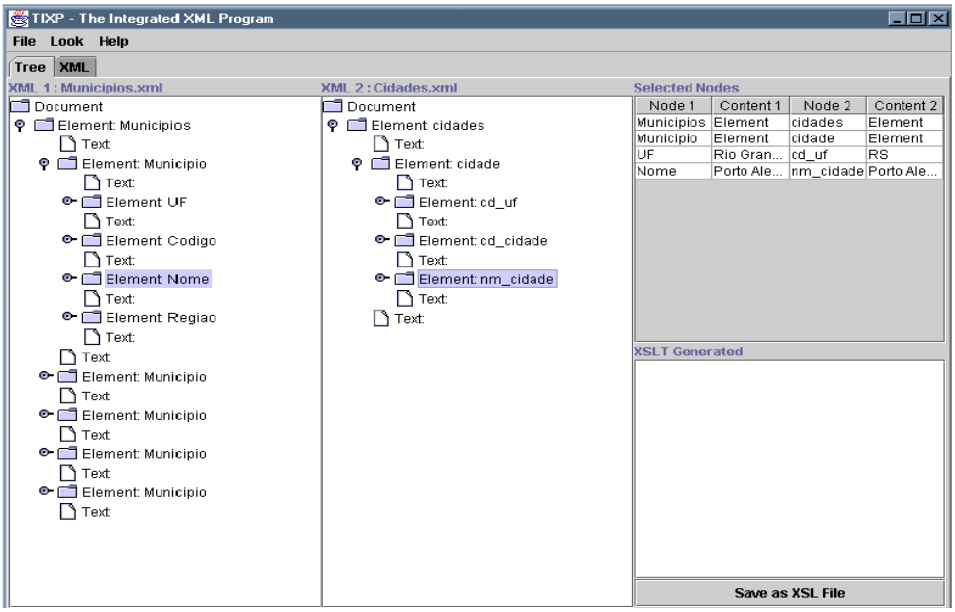


FIGURA 3.3 - Mapeamentos de nodos na interface gráfica do TIXP

As transformações de conteúdo são realizadas através do preenchimento da tabela "*Conversion Table*" (figuras 3.4 e 3.5). Para cada conversão de domínio o programador deve aplicar uma operação *Create/Edit Conversion Table* na linha da tabela de mapeamentos referente ao nodo cujo conteúdo deve ser convertido. A tabela de conversão possui duas colunas: *From* e *To,* que são preenchidas com os valores das colunas "*Content 1*" e "*Content 2*" da "*Selected Nodes*", e representam o valor a ser substituído e o valor de substituição, respectivamente. A inserção de novas linhas nessa tabela pode ser realizada através da operação "*Add Row*".

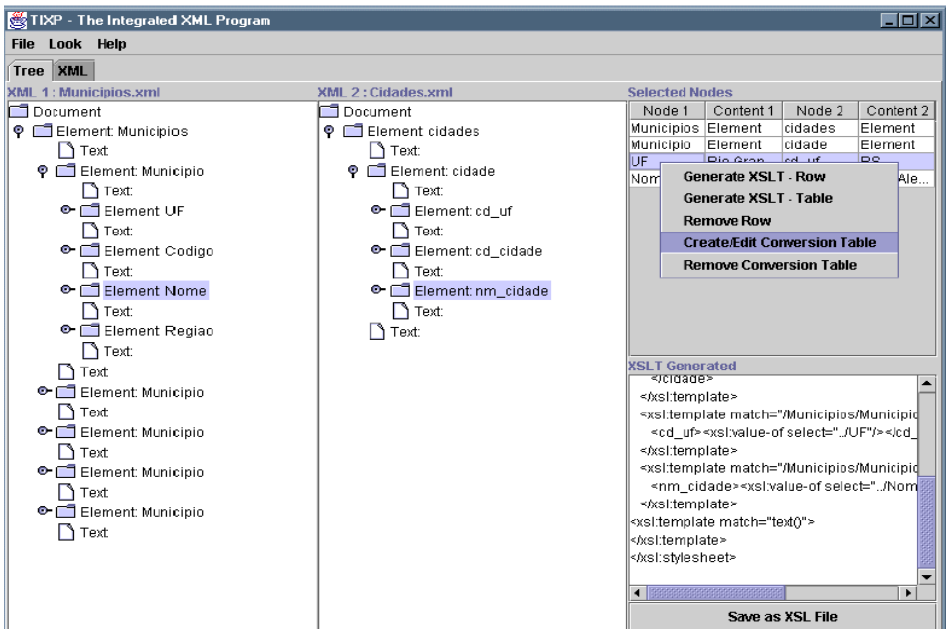


FIGURA 3.4 - Criação de uma *Conversion Table*

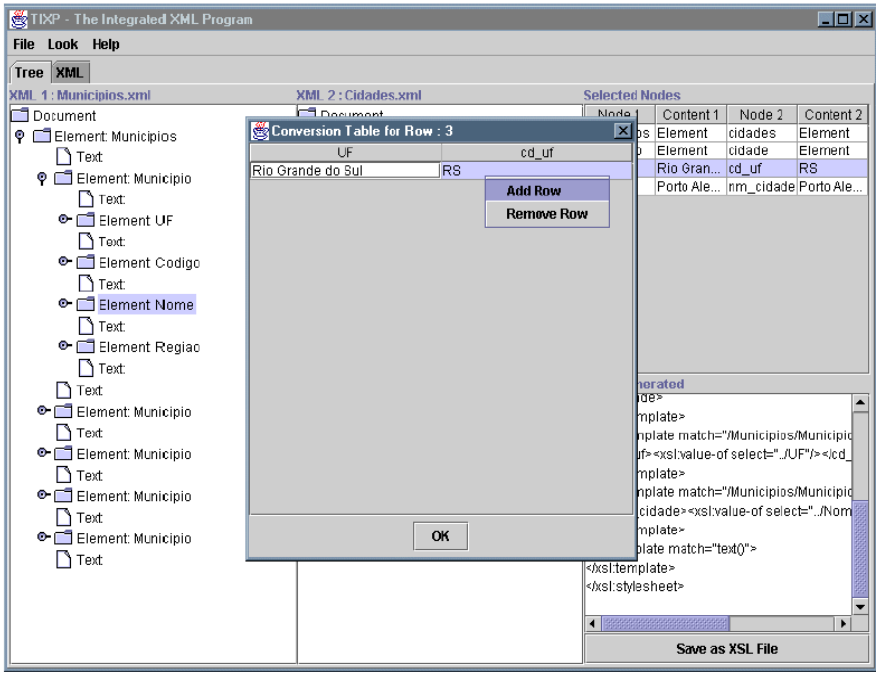


FIGURA 3.5 - Preenchimento da *Conversion Table*

3.2.3 Geração de Regras XSLT

No TIXP, as entradas do processo de geração de um *script* XSLT são as tabelas "*Selected Nodes*" e "*Conversion Table*" preenchidas pelo programador na interface gráfica. A "*Selected Nodes*" é utilizada para gerar as regras referentes a alterações estruturais (mudança de vocabulário). A "*Conversion Table*" é utilizada para gerar as regras para a realização para conversão de domínios.

Para cada linha da tabela "*Selected Nodes*" é gerada uma regra. O formato dessa regra depende do tipo de preenchimento da tabela, podem ser apresentadas nos seguintes formatos:

1) O conteúdo da coluna "*Content 1*" é o valor "*Element*".

```
<xsl:template match = "expressão XPath de Node1">
 <Node 2>
 <xsl:apply-templates/>
 </Node 2>
</xsl:template>
```

2) O valor da coluna "*Content 1*" é diferente de "*Element* " e não existe uma "*Conversion Table*" associada a esta linha.

```
<xsl:template match = "expressão XPath de Node1">
 <Node 2>
 <xsl:value-of-select="expressão XPath de Node1" />
 </Node 2>
</xsl:template>
```

3) O valor da coluna "*Content 1*" for diferente de "*Element* " e existe uma "*Conversion Table*" associada à linha da "*Selected Nodes*", é gerada a instrução *<xsl:choose>* , sendo que cada linha desta tabela gera uma instrução *<xsl:when>* test="*From*"> *<Node2>To</Node2>* dentro da *<xsl:choose>*.

```
<xsl:template match = "expressão XPath de Node1">
 <xsl:choose>
  <xsl:when test text() = "Valor da coluna "From" da linha 1">
```

```
    <Node 2>Valor da coluna To da linha 1" </Node 2>
   </xsl:when>
   <xsl:when test text() = "Valor da coluna "From" da linha N">
    <Node 2>Valor da coluna To da linha N" </Node 2>
   </xsl:when>
   <xsl:otherwise>
    <Node 2>
    <xsl:value-of-select="expressão XPath de Node1" />
    </Node 2>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
```

3.3 Visual XML Transformation

A ferramenta *Visual XML Transformation* faz parte de um pacote de ferramentas visuais da IBM para XML[LAU 99], que visa facilitar a edição, a conversão, a geração e a composição de dados XML. O papel da *Visual XML Transformation* dentro deste pacote é auxiliar a criação de outro documento XML a partir de outros documentos XML já existentes. Para atingir este objetivo, um *script* XSLT é gerado a partir de uma especificação realizada pelo usuário em uma interface gráfica.

3.3.1 Interface Gráfica

A especificação de um novo documento XML a partir de outros documentos XML existentes na ferramenta Visual XML Transformation é realizada em uma interface gráfica, através da abertura de uma *transformation session*, onde uma ou mais DTDs podem ser selecionadas para a composição do novo documento. A interface se apresenta divida em páginas com as seguintes finalidades.

- DTD - onde as estruturas das DTDs selecionadas são apresentadas para a especificação das transformações;
- *Transformation* - onde é acionada a geração do *script* XSLT e os resultados são apresentados;
- *Summary* - exibe os mapeamentos realizados na página DTD para a obtenção do documento resultado.

3.3.2 Especificação de Transformações

A especificação para a criação de um documento resultado XML na *Visual XML Transformation*, é realizada através da estrutura obtida a partir das DTDs dos documentos fonte XML, exibida na janela DTD do lado *Source* DTDs (figura 3.6). Primeiro são selecionados os elementos da *Source* DTDs que farão parte da *Target* DTD; em seguida, um conjunto de operações que poderão ser aplicadas para inclusão, remoção e reordenamento de elementos, pode ser realizada em cima da Target DTD até que se obtenha a estrutura desejada na saída.

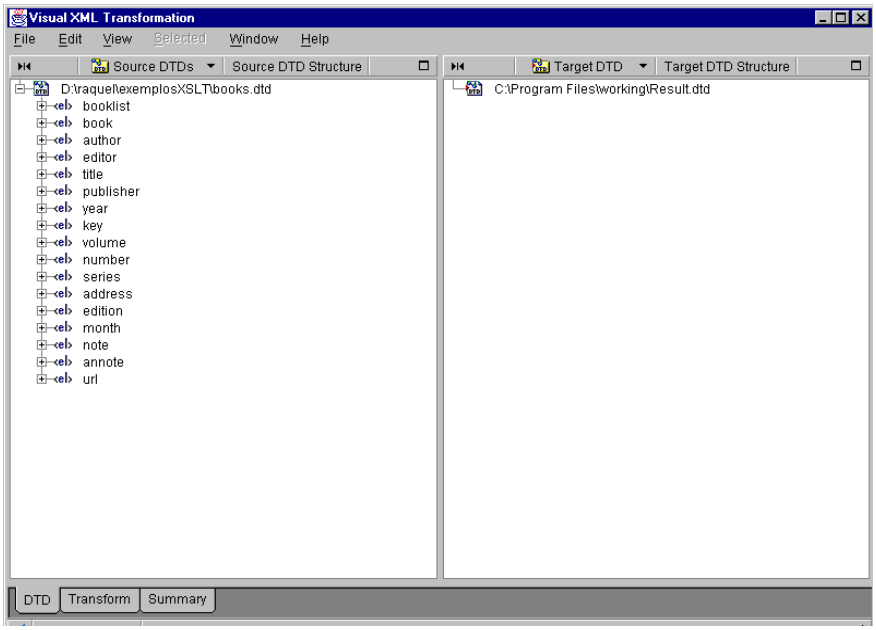


FIGURA 3.6 - Interface gráfica da *Visual XML Transformation*

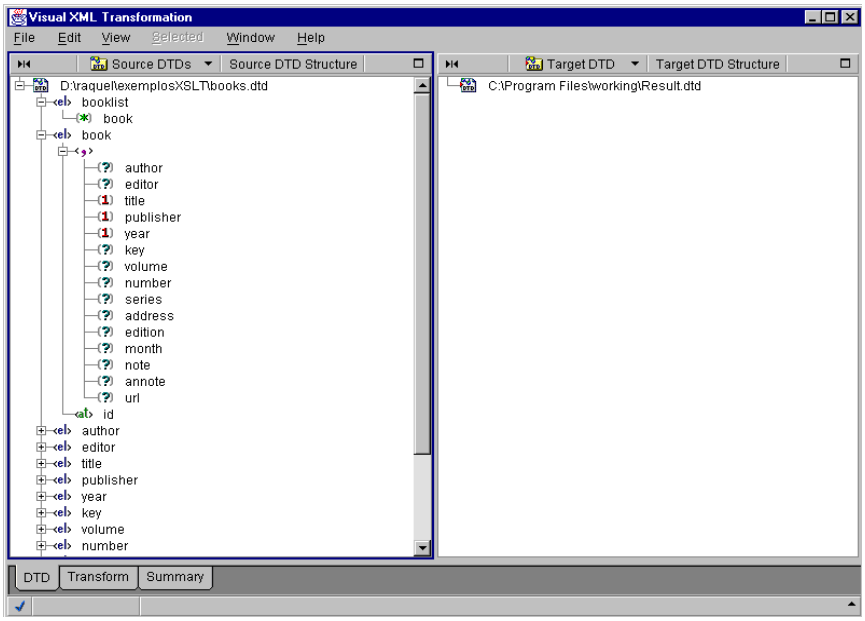


FIGURA 3.7 - Representação expandida da DTD na *Visual XML Transformation*

Seleção de elementos da *Source* DTD

A seleção dos elementos que farão parte da *Target* DTD é realizada através das operações *copy* e *paste* aplicadas na *Source* DTDs e *Target* DTD respectivamente. (Figura 3.8).

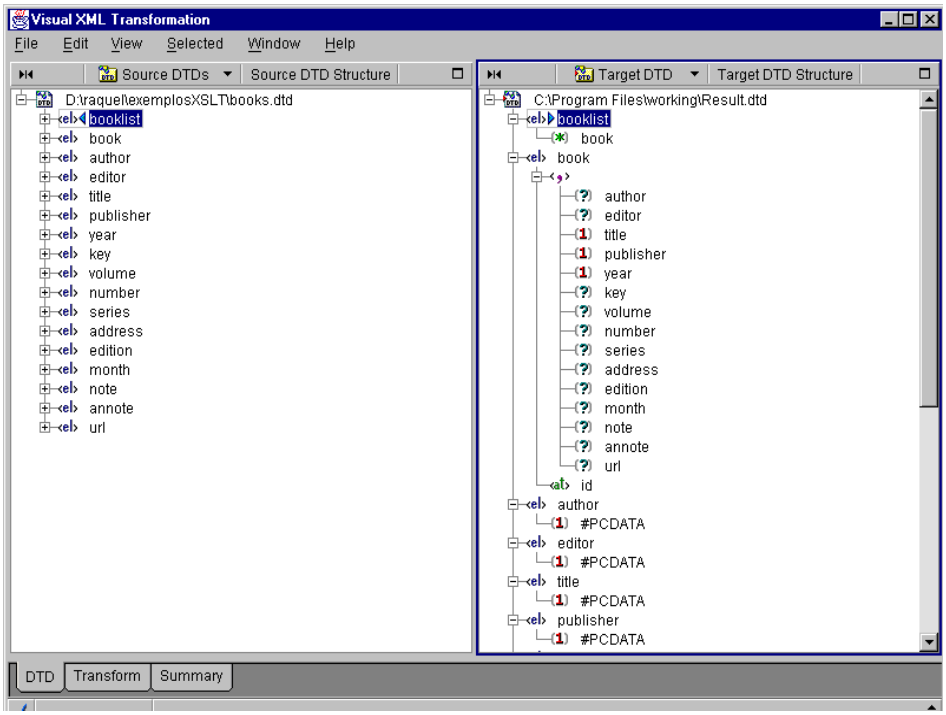


FIGURA 3.8 - Copia elementos para a a *Target* DTD

Aplicação de Operações

Uma vez selecionados os elementos da *Source* DTDs que participarão da especificação da *Target* DTD, um conjunto de operações pode ser aplicado sobre estes. Essas operações são disponibilizadas na interface através de um *menu* (figura 3.9).

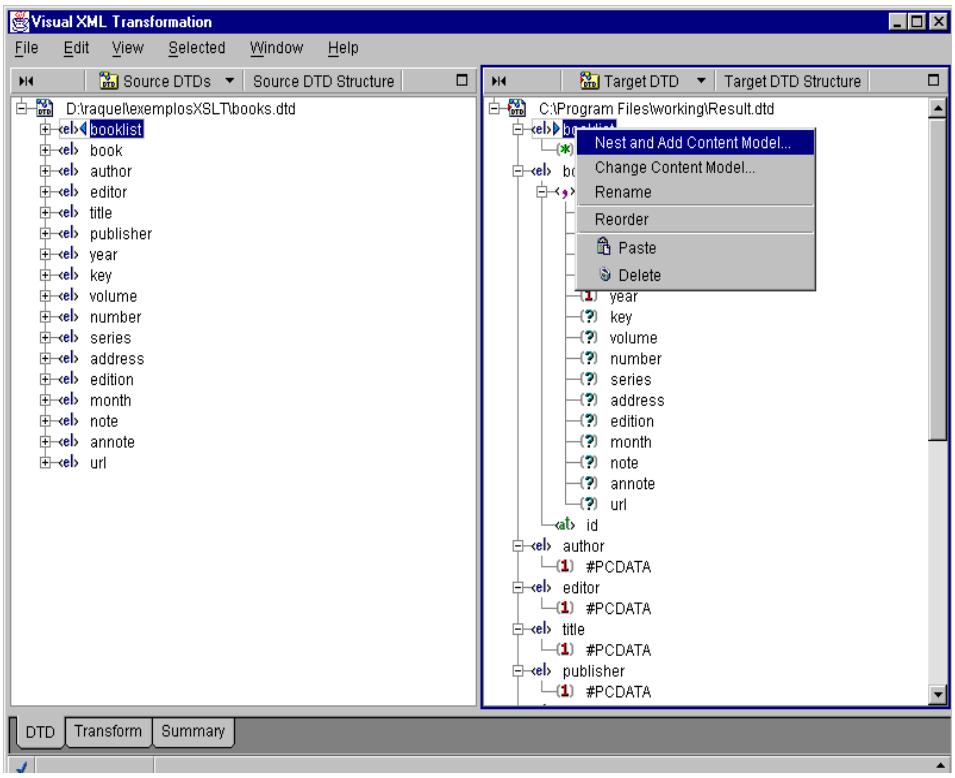


FIGURA 3.9 - Menu de operações sobre os elementos da *Target* DTD

As operações são disponibilizadas no menu dependendo do tipo de elemento selecionado e são basicamente operações direcionadas para manipulação de DTDs. A seguir são apresentadas as operações disponíveis na ferramenta:

- *Delete* - remove um elemento ou atributo da estrutura, é válida para todos os elementos, exceto para o elemento superior da estrutura que faz referência ao arquivo onde será gravada a DTD gerada;
- *Rename* - renomeia um elemento. Válida para os nodos do tipo elemento;
- *Convert to Attribute* - transforma um elemento em atributo. Válida para os elementos sem descendentes;
- *Reorder* - altera a ordem em que um elemento ou atributo aparece na estrutura; válida para todos os tipos de elementos, exceto o que faz referência ao arquivo onde será gravada a DTD gerada;
- *Nest and Add Content Model* - adiciona um elemento ou um atributo como filho do elemento selecionado, informando também o tipo de relacionamento;
- *Add Content Model Before* - adiciona um elemento ou atributo antes do elemento selecionado;
- *Add Content Model After* - adiciona um elemento ou atributo após o elemento selecionado;
- *Change Content Model* - altera o tipo de relacionamento entre um elemento ou atributo e o nodo selecionado;
- *Add Element* - insere um elemento na DTD alvo.

3.3.3 Geração de Regras XSLT

A entrada para geração do *script* XSLT, é a estrutura da *Target* DTD na qual são armazenadas as especificações realizadas pelo usuário na interface gráfica. A seguir é apresentado um exemplo de *script* XSLT gerado por esta ferramenta.

```
<?xml version="1.0" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:output method="xml" indent="no" xml-declaration="yes" />
<xsl:strip-space elements="*" />
<xsl:variable name="v0"
select="document('D:/raquel/exemplosXSLT/books.xml')" />

<xsl:template match="/">
 <xsl:apply-templates select="$v0//booklist[1]" />
</xsl:template>

<xsl:template match="booklist">
 <listalivros>
 <xsl:apply-templates select="*|@*|comment()|processing-
  instruction()|text()" />
 </listalivros>
</xsl:template>

<xsl:template match="book">
 <livro>
 <xsl:apply-templates select="@*" />
 <xsl:apply-templates select="author[1]" />
 <xsl:apply-templates select="title[1]" />
```

```
 </livro>
</xsl:template>

<xsl:template match="author">
 <autor>
 <xsl:apply-templates select="*|@*|comment()|processing-
  instruction()|text()" />
 </autor>
</xsl:template>

<xsl:template match="title">
 <titulo>
 <xsl:apply-templates select="*|@*|comment()|processing-
  instruction()|text()" />
 </titulo>
</xsl:template>

<xsl:template match="*|@*|comment()|processing-instruction()|text()">
<xsl:copy>
<xsl:apply-templates select="*|@*|comment()|processing-
instruction()|text()" />
</xsl:copy>
</xsl:template>

</xsl:transform>
```

3.3.4 Resultados Gerados

Os resultados gerados pela *Visual XML Transformation* poderão ser visualizados na página *Transform* da interface gráfica (figura 3.10).

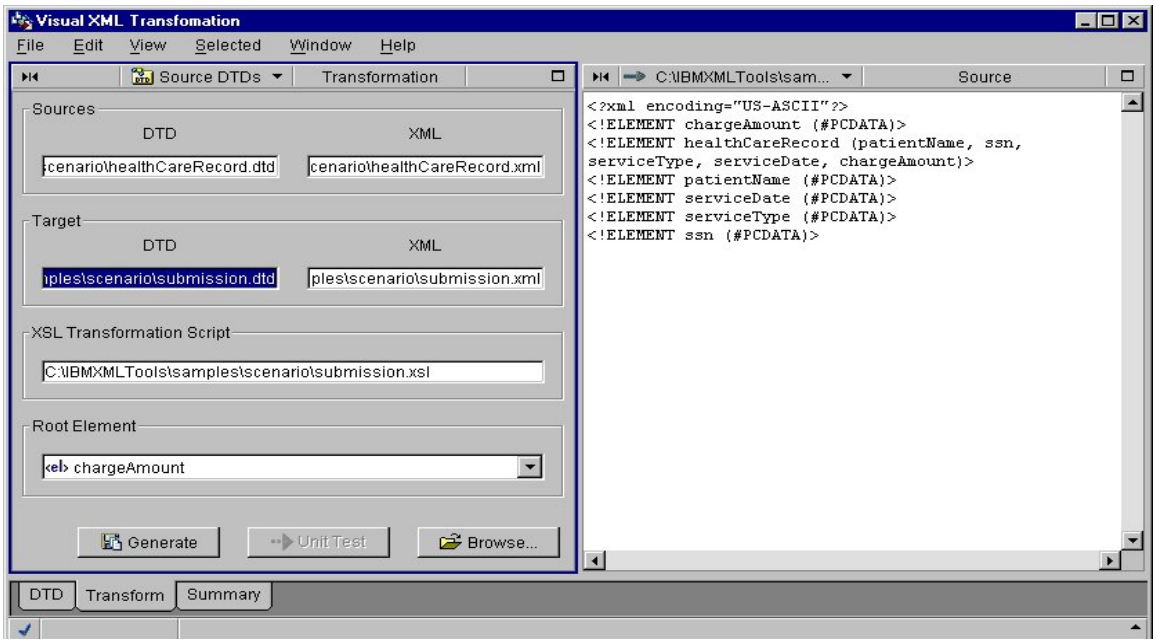


FIGURA 3.10 - Página *Transform* com os resultados gerados

Após a especificação das transformações na página DTD, os seguintes resultados são obtidos:

- *Target* DTD: gerada automática, a medida em que os eventos vão ocorrendo na interface gráfica ;

- Um *script* XSLT para transformar o(s) documento(s) fonte(s) no documento resultado desejado: este é gerado através do botão <*Generate*>;
- Uma *Unit test* que chama um processador XSLT para a execução do *script* gerado a partir do(s) documento(s) fonte(s), gerando o documento resultado.

3.4 Conclusão e Comparação

As ferramentas apresentadas neste capítulo têm um objetivo comum: aumentar a produtividade de programação na linguagem XSLT. Para atingir este objetivo utilizam diferentes recursos, o que as dividem em dois grupos. O primeiro grupo permite o reuso de transformações já programadas, que é o caso do sistema CoX. O segundo grupo é composto das ferramentas que geram *script* XSLT a partir da especificação de transformações realizadas em uma interface gráfica; nesse grupo são classificadas as ferramentas TIXP e *Visual XML Transformation* que apresentam, portanto mais características em comum.

As ferramentas TIXP e CoX foram construídas dentro de um contexto de compartilhamento de documentos XML entre diferentes aplicações; isto determinou a necessidade de um tipo de documento alvo para o qual o documento fonte deve ser transformado. No CoX, além de um tipo alvo de documento, é necessário que os documentos fonte e alvo possuam uma DTD, uma vez que a idéia principal dessa abordagem é: compartilhar transformações entre diferentes documentos que são baseados na mesma DTD. A *Visual XML Transformation*, neste sentido, é mais genérica, pois gera um documento resultado a partir de qualquer documento fonte, desde que este possua uma DTD.

Em relação à entrada do processo de transformação, o CoX e o TIPX selecionam uma instância de um documento fonte XML. A entrada para *Visual XML Transformation* é o esquema de um documento XML, representado pela DTD.

Apesar das três ferramentas utilizarem uma interface gráfica, dependendo do grupo ao qual pertencem, diferentes funções podem ser executadas em cada uma delas. No sistema CoX, a interface é utilizada para a seleção dos tipos de documentos fonte e alvo e para visualização de resultados. Nas outras duas ferramentas, além das funcionalidades apresentadas pela primeira, a interface é utilizada para a especificação das transformações que servirão de entrada para a geração do *script* XSLT.

As ferramentas do segundo grupo, TIXP e *Visual XML Transformation*, definem um escopo de transformações XSLT que podem ser especificadas através de suas interfaces, restringindo assim, o conjunto de operações XSLT que serão incluídas no *script* gerado. Como o sistema CoX utiliza a própria XSLT para a especificação de transformações, toda a expressividade desta linguagem pode ser utilizada na especificação das transformações.

Quanto ao reaproveitamento de transformações realizadas no sistema CoX, todas as transformações realizadas para um tipo de documento podem ser reusadas, uma vez que isto é a base desta ferramenta. No TIXP, a especificação para transformação de um documento do tipo **A** para um documento do tipo **B**, não poderá ser reusada para transformar o mesmo documento **A** em um documento do tipo **C**. *Na Visual XML*

Transformation, uma seção de transformação pode ser reusada para a especificação de um novo documento.

Para que um resultado gerado possa ser refinado ou aprimorado, é necessário que uma especificação possa ser armazenada e reaproveitada para melhorar o resultado obtido. Como já mencionado, na *Visual XML Transformation* os resultados são armazenados e a especificação é vinculada a uma seção de transformação. No TIXP, uma especificação é chamada de XML *Transformated*, mas no *menu* da ferramenta não existe uma opção de abertura para a retomada de uma transformação. O CoX não faz referência a essa característica.

Na XSLT, um documento resultado pode ser gerado a partir de mais de um documento fonte. Somente a *Visual XML Transformation* implementa esta característica, visto que, em uma seção de transformação, mais de uma DTD pode ser selecionada para compor um novo documento.

Em relação ao formato de saída, as ferramentas do segundo grupo geram basicamente outros documentos XML. A ferramenta CoX, classificada no primeiro grupo, utiliza toda a expressividade da linguagem XSLT; conseqüentemente, pode gerar saída em todos os diferentes formatos possíveis para esta linguagem.

As comparações entre as ferramentas descritas acima são resumidas na tabela 3.1.

TABELA 3.1- Resumo das principais características das ferramentas de suporte a XSLT

Características	CoX	TIXP	Visual XML Transformation
Recurso utilizado para aumentar a produtividade	reuso de transformações	Geração automática de *script*	geração de automática de *script*
Pré-requisitos	Dados do meta-esquema	Documento XML meta	documento fonte tem que possuir uma DTD
Entrada	Documento fonte e um tipo alvo	dois documentos XML	DTDs
Utilização da Interface Gráfica	seleção o documento fonte e um tipo de documento alvo	Especificar transformações	especificar transformações
Transformações XSLT possíveis	todas	Mudança de vocabulário e conversão de conteúdos	mudança de vocabulário, inclusão, remoção e reordenamento de elementos e atributos
Reuso de Transformações	sim	não	indiretamente, pelo reuso de transformações armazenadas em uma seção de transformação
Refinamento de Resultados	não	não	sim
Suporte para mais de um documento fonte	não	não	sim
Formato de Saída	XML	XML	XML

Atualmente, a maior demanda e, portanto, o uso mais comum da XSLT é converter documentos XML para HTML[KAY 2000]. Alguns inconvenientes podem ser encontrados na tentativa de se realizar este tipo de conversão nas ferramentas descritas neste capítulo:

a) CoX - Como já mencionado, essa ferramenta não gera *script* XSLT, e sim a utiliza como linguagem para programação de transformações. Portanto, a cada novo tipo alvo de documento novas transformações têm que ser programadas, e para isso, esta ferramenta não apresenta nenhuma facilidade que auxilie o programador;

b) TIXP - mapeamento de nodos nesta ferramenta é um relacionamento um para um entre os nodos, do primeiro e do segundo documento XML. Assim é possível, associar o elemento raiz do primeiro documento com o elemento raiz de um documento HTML, que nesse caso seria o próprio elemento HTML, mas os outros elementos da estrutura de um documento HTML, como, por exemplo, o elemento *body*, ficaria sem par;

c) *Visual XML Transformation* - como a entrada para a geração de regras é a *Target* DTD, é possível a construção de uma para a estrutura HTML; porém, a definição da hierarquia entre os elementos é difícil, exigindo um certo conhecimento sobre construção de DTDs. Esta dificuldade aumenta quando existe necessidade de se inserir elementos HTML, que podem aparecer mais de uma vez na estrutura do documento resultado e/ou que podem apresentar conteúdos diferentes, como por exemplo, o elemento *table*. Como a especificação é realizada a partir da DTD, não se pode acessar os conteúdos, impossibilitando a formatação destes ou a inclusão de novos valores; isso pode ser confirmado através das folhas de estilos geradas por esta ferramenta na qual há somente uma regra para todos os nodos do tipo texto, como pode ser observado no exemplo da seção 3.3.3.

No próximo capítulo será apresentada uma ferramenta que auxilia o programador XSLT a gerar documentos HTML a partir de documentos fontes XML. Esta ferramenta pertence ao segundo grupo, pois a partir de uma especificação realizada por um usuário em uma interface gráfica, gera um *script* XSLT para a apresentação de um documento fonte XML no formato HTML desejado.

4 X2H

A X2H é uma ferramenta que visa aumentar a produtividade de programação na linguagem XSLT, auxiliando a apresentação de conteúdos XML em HTML. Em relação aos recursos utilizados para alcançar o objetivo proposto, a X2H pode ser classificada como integrante do segundo grupo, identificado na seção 3.4 do capítulo 3, pois, a partir de uma especificação realizada em uma interface gráfica, gera um *script* XSLT.

A especificação de transformações da X2H segue a abordagem orientada a exemplos proposta em [ZLO 75], com a linguagem QbE utilizada para consultar bancos de dados relacionais. Na QbE, a formulação de consultas é realizada pela representação de modelos ou *templates*, que servem como exemplos para filtrar os dados e selecionar aqueles que devem constar no resultado [KAD 2001]. Esta proposta serviu de base para outros trabalhos, tais como: DEByE, uma ferramenta para extração de dados semi-estruturados [LAE 99, SIL 99], baseada em padrões gerados a partir de exemplos de objetos especificados pelo usuário; e a QSByE, linguagem de consulta para dados gerados pela ferramenta DEByE [EVA 2001a, EVA 2001b], que estende o paradigma QBE para dados semi-estruturados.

Neste capítulo, serão descritos por primeiro, os módulos componentes da X2H através de um exemplo. Em seguida, será realizado um estudo de caso no qual poderá ser observado o comportamento da ferramenta perante um documento fonte XML, que possui uma estrutura diferente da apresentada pelo documento utilizado como exemplo na descrição da ferramenta.

4.1 Descrição da X2H

Na X2H, a especificação do exemplo é realizada em uma **interface gráfica**, na qual o usuário seleciona partes do documento fonte XML e copia para um documento exemplo que servirá de entrada para o módulo **gerador de regras** (figura 4.1).

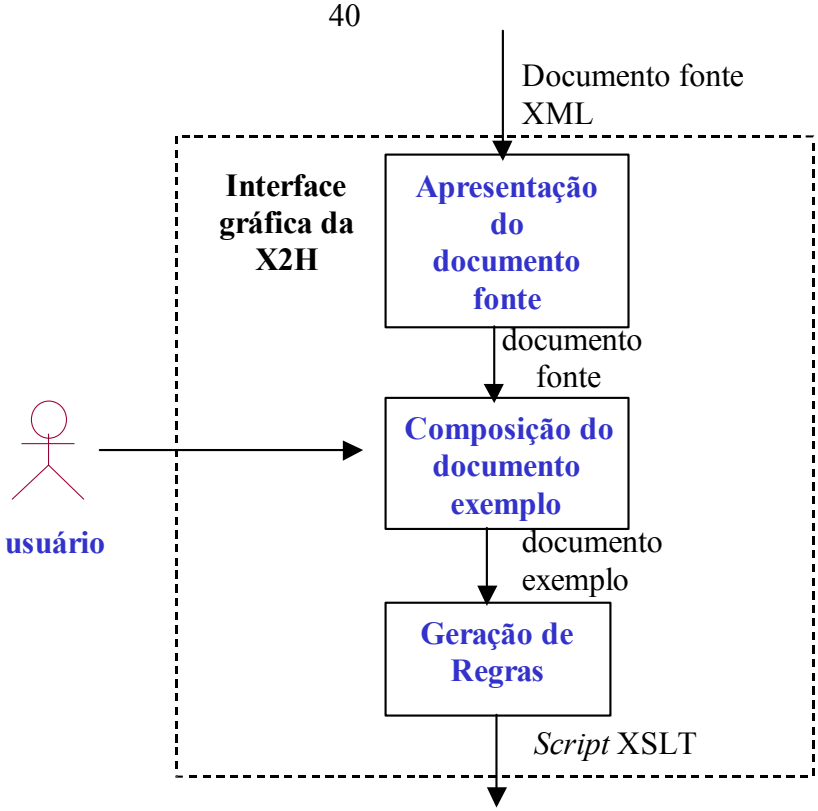


FIGURA 4.1 - Módulos componentes da X2H

Para descrever a X2H, será utilizado o documento fonte XML, **books.xml**. O resultado desejado é o documento HTML da figura 4.2. Descreve-se primeiro a apresentação do documento fonte na interface gráfica da ferramenta. A partir desta apresentação, será realizada a composição de um documento exemplo que servirá de entrada para o processo de geração de regras, tendo como saída um *script* XSLT.

O documento XML fonte (books.xml) usado no exemplo é o seguinte:

```
<?xml version="1.0" ?>
<booklist>
 <book id="BOX00">
  <author>Box, D. and Skonnard, A. and Lam, J.</author>
  <editor>Series</editor>
  <title>Essential XML - Beyond Markup</title>
  <publisher>Addison-Wesley</publisher>
  <year>2000</year>
  <key />
  <volume />
  <number />
  <series />
  <address />
  <edition />
  <month>July</month>
  <note />
  <annote />
  <url>http://www.develop.com/books/essentialxml</url>
 </book>
 <book id="MAR99">
  <author>Maruyama, H. and Tamura, K. and Uramoto, N.</author>
  <title>XML and Java: Developing of Web Applications</title>
  <publisher>Addison-Wesley</publisher>
  <year>1999</year>
  <address>MA</address>
  <month>August</month>
```

```
</book>
<book id="BRA00">
 <author>Bradley, N.</author>
 <title>The XML Companion</title>
 <publisher>Addison-Wesley</publisher>
 <year>2000</year>
 <address>Great Britain</address>
 <edition>2</edition>
 <month>August</month>
</book>
</booklist>
```

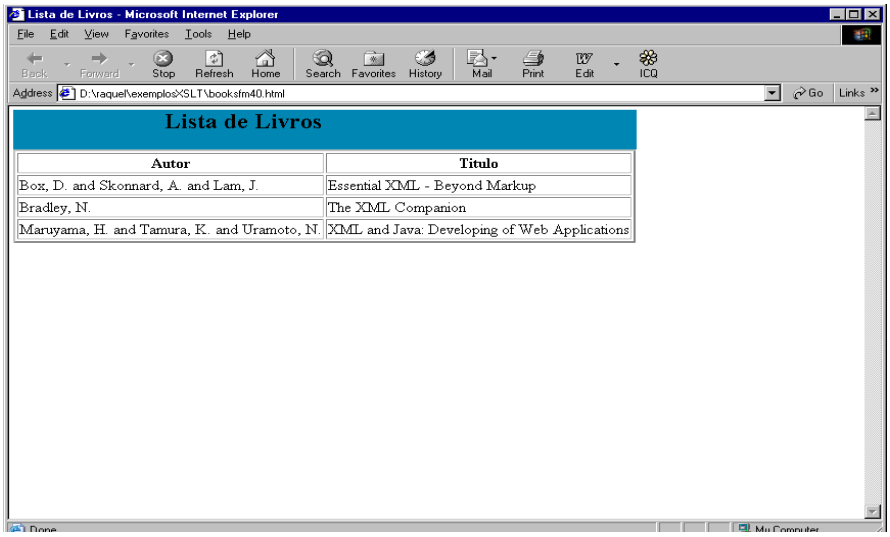


FIGURA 4.2 - Documento resultado exibido em web browser

4.1.1 Apresentação do documento fonte

A interface da X2H é composta por quatro páginas: XML, Exemplo, *Script* XSLT e HTML.

O documento **books.xml** é apresentado nas páginas XML e Exemplo. Na primeira, é exibido em formato textual (figura 4.3). Na Segunda, é apresentado em forma de árvore hierárquica (figura 4.4) . Além desta árvore, esta página, na qual será realizada a especificação das transformações, contém a árvore do documento exemplo inicializado com a estrutura básica de um documento HTML (elemento *html* e seus filhos *head* e *body*), e uma área de atributos associada aos nodos da árvore exemplo.

As páginas *Script* XSLT e HTML são utilizadas para, respectivamente, apresentar o *script*.XSLT e o documento resultado HTML.

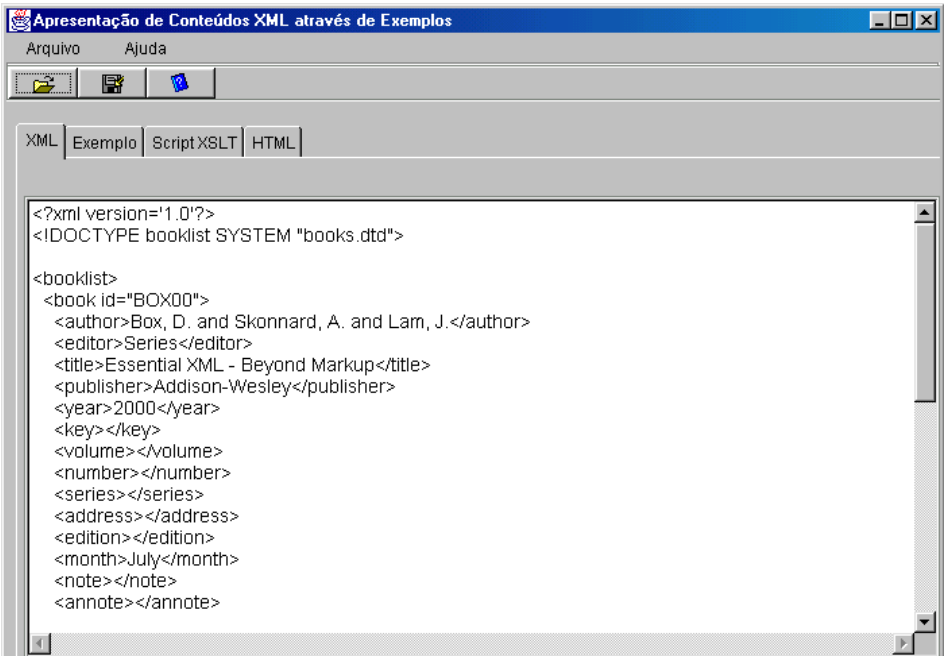


FIGURA 4.3 – Documento fonte em formato textual

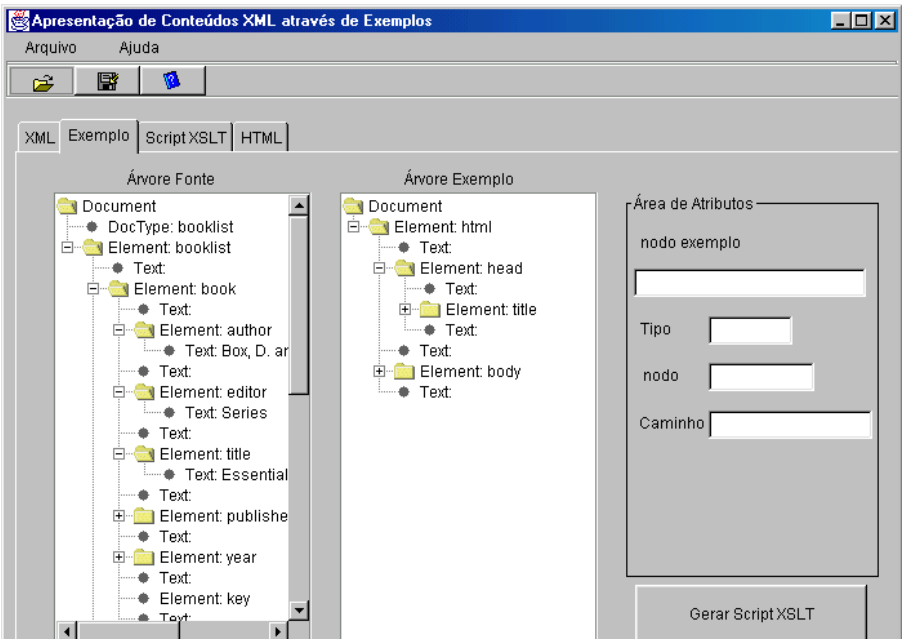


FIGURA 4.4 - Documento fonte em formato de árvore hierárquica

Árvores fonte e exemplo

As árvores fonte e exemplo (figura 4.4), que representam respectivamente o documento fonte books.xml e o documento exemplo são geradas por um *parsing* XML através de uma API DOM [W3C 98a, BRA 2000]; que implementa uma interface para

acesso e manipulação de nodos. Essa interface permite a identificação dos diversos tipos de nodos que podem compor um documento XML: elementos, textos, comentários, instruções de processamento, seções CDATA, documento, fragmentos de documento e atributos de um elemento. Além disso, esta interface possui um conjunto de métodos que permite:

a) obter as características dos nodos como nome, tipo e valor;

b) localizar e navegar em árvores de documentos, possibilitando o acesso relativo através dos relacionamentos hierárquicos entre os nodos de uma árvore;

c) manipular os nodos de uma árvore, permitindo operações de remoção, adição, ordenamento e modificação de conteúdos.

Área de Atributos

A Área de Atributos da página Exemplo da interface gráfica da ferramenta, é associada à árvore exemplo. Esta área é composta por dois tipos de atributos que dependem do tipo de nodo da árvore exemplo. Para os nodos que representam elementos HTML, são **os atributos HTML**; para os nodos copiados a partir da árvore fonte, são os **atributos de tradução**. Os primeiros permitem atribuir aos elementos HTML detalhes de apresentação como tipo fonte, cor de fundo e tamanho das margens. Já os segundos permitem aplicar operações aos nodos copiados da árvore fonte como ordenação de elementos, inserção de elementos em tabela e inserção de outros elementos HTML.

4.1.2 Composição do Documento Exemplo

Após a apresentação do documento fonte na interface gráfica, é necessário compor um documento exemplo que reflita a estrutura desejada no documento de saída, que no caso do exemplo considerado é composto de duas tabelas: a primeira é um elemento HTML *table*, que contém um cabeçalho no formato HTML h2 e a *string*: "Lista de Livros"; a segunda tabela é outro elemento *table* na qual cada linha apresenta os conteúdos dos elementos `<author>` e `<title>`, filhos de um elemento `<book>` do documento fonte.

O documento exemplo é representado na interface através da árvore exemplo, portanto, esta estrutura será utilizada para a composição deste documento. Para isto, os seguintes elementos deverão ser inseridos na árvore exemplo:

a) os nodos exemplos, selecionados na árvore fonte;

b) elementos HTML.

Seleção de nodos exemplos

A seleção de um nodo na árvore fonte para compor o exemplo, é realizada através da operação **Copiar** (figura 4.5), acionada através do botão direito do *mouse* em cima do nodo desejado. Esta operação é válida somente para os nodos fonte do tipo elemento. Neste exemplo, esta operação será aplicada a um elemento `<book>` selecionado.

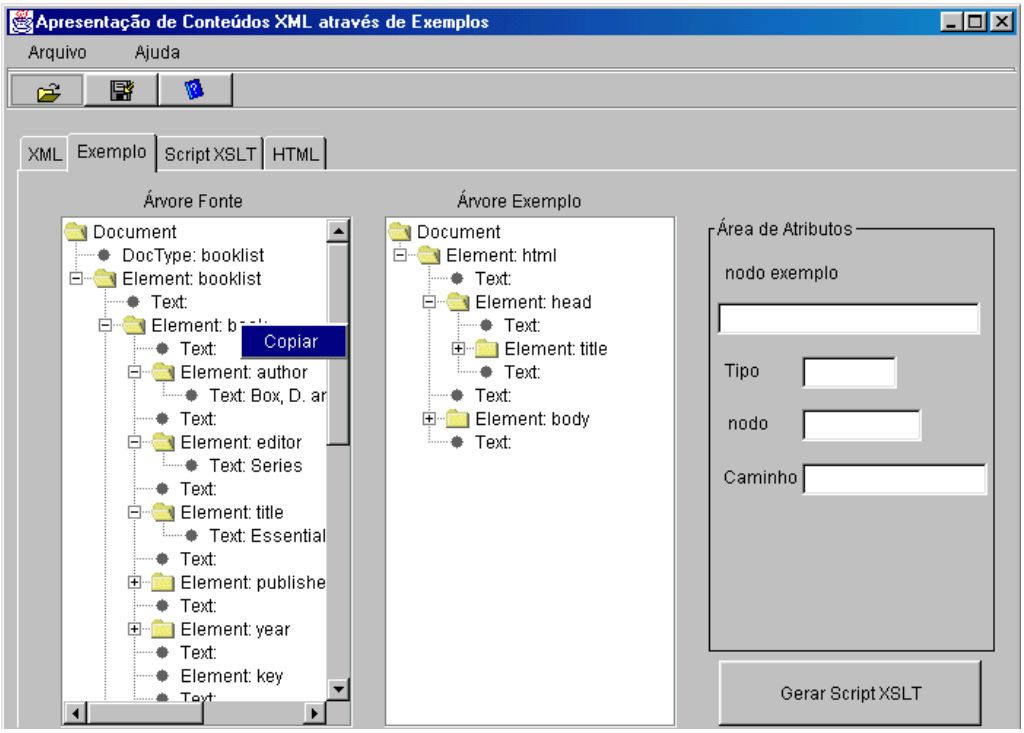


FIGURA 4.5 - Seleção de nodos exemplos na árvore fonte

A execução da operação **Copiar** exibe a janela **"Selecionar nodos exemplos"** (figura 4.6) que contém dois objetos árvores e uma área para especificação de predicados para o nodo selecionado.

A primeira árvore é a subárvore do documento fonte que tem como raiz o elemento selecionado, no caso, a ocorrência do nodo `<book>` com todos os seus nodos filhos. A segunda árvore é construída a partir dos nodos selecionados na primeira subárvore, copiados através dos botões apresentados na janela **"Selecionar nodos exemplos"** (figura 4.6). No caso, os nodos filhos de `<book>`: `<author>` e `<title>`.

A construção da subárvore exemplo pode ser realizada das seguintes formas:

a) seleção de todos os nodos da **subárvore fonte** através de uma única operação, esta opção facilita a seleção de nodos exemplos, para os casos em que a estrutura da **subárvore exemplo** apresenta poucas alterações em relação a estrutura da subárvore fonte selecionada;

b) seleção e cópia de um nodo da **subárvore fonte** para a **subárvore exemplo** de cada vez, esta opção pode ser utilizada nos casos em que a estrutura da **subárvore exemplo** apresenta um subconjunto bem pequeno da **subárvore fonte**.

Para o exemplo que está sendo considerado, é mais adequado optar pela segunda forma, uma vez que somente dois nodos filhos de `<book>` são necessários para a composição do exemplo.

Deve-se notar que a raiz das duas subárvores é o mesmo elemento `<book>` .

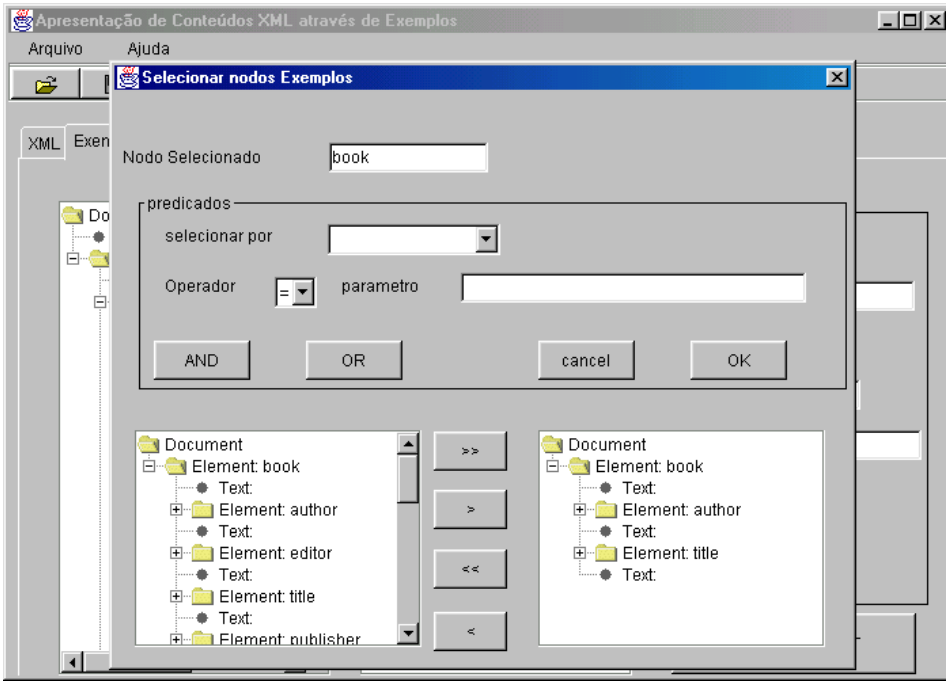


FIGURA 4.6 - Construção da subárvore exemplo

Tanto na seleção da subárvore fonte como na seleção da subárvore exemplo, é necessário somente um **"elemento exemplo"**, representante de um conjunto de elementos repetidos. Assim, apesar da árvore fonte apresentar três elementos `<book>`, somente um nodo deste tipo de elemento deve ser selecionado para servir de "modelo" para os demais.

Na área de predicados da janela "Selecionar nodos Exemplo", pode-se definir filtros para os elementos do nodo selecionado. Por exemplo, se no resultado fossem apresentados somente os elementos `<book>` com year='1999', as seguintes informações deveriam ser preenchidas no predicado para este elemento: *selecionado por* ***year*****,** *operador* ***"="*** e *parâmetro* **1999**. Um predicado pode ser composto por mais de uma condição, neste caso, os operadores ***AND*** e ***OR*** devem ser utilizados.

Após a seleção da subárvore exemplo, os nodos `<book>` e seus filhos `<author>` e `<title>`, são inseridos na árvore exemplo através da operação **Colar** (figura 4.7), como filhos do elemento HTML *body*.

Para os nodos fontes inseridos na árvore exemplo, a **Área de atributos** exibe os seguintes valores:

- **tipo=fonte:** para indicar que o nodo foi copiado a partir da árvore do documento fonte;
- **o atributo caminho= "expressão de caminho em relação ao nodo raiz do documento fonte"**, somente para o elemento raiz da subárvore exemplo inserida. Neste caso, somente para o elemento `<book>`**;**
- **os atributos de tradução:** válidos somente para os nodos fontes do tipo elemento. No exemplo considerado, está sendo utilizado o atributo ***sort*** para definir a ordem na qual os elementos do tipo `<book>` devem ser exibidos na saída; o *default* é a ordem em que aparecem no documento fonte. Na X2H,

esta ordem pode ser alterada para que os nodos do elemento selecionado sejam exibidos de acordo com um de seus nodos filhos. Além do atributo ***sort***, a X2H possui o atributo de tradução ***number***, que pode ser utilizado para gerar automaticamente uma seqüência numérica e/ou formatar como esta será apresentada na saída.

No exemplo, para o elemento `<book>`, a Área de Atributos apresenta os seguintes valores: **caminho= "book", fonte= "true" e sort= "author"** (figura 4.8).

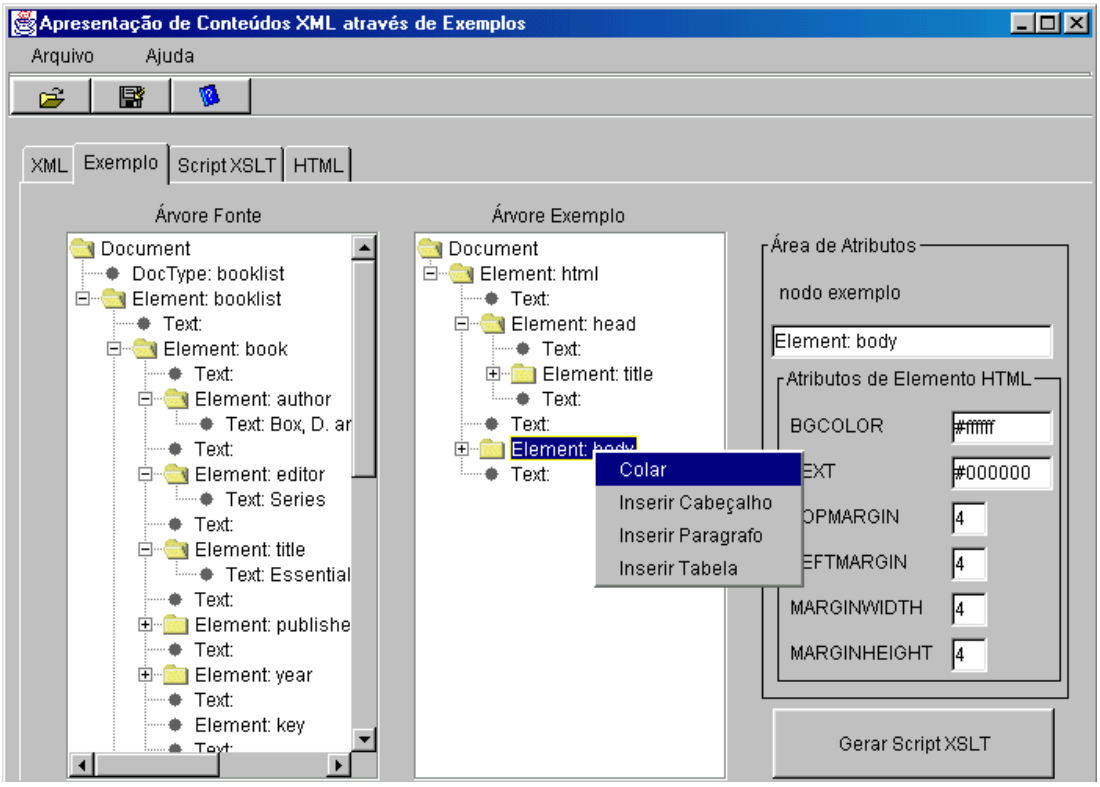


FIGURA 4.7 - Inclusão da subárvore exemplo na árvore exemplo

Inclusão de elementos HTML

Após a inclusão dos nodos fontes na árvore exemplo, o usuário deverá inserir os elementos HTML que constam no resultado. Considerando como resultado o documento exemplo, devem ser inseridos dois elementos do tipo *table*. O primeiro *table* não contém elementos fontes; o segundo é composto pelos conteúdos dos nodos `<author>` e `<title>` filhos de `<book>`.

Para facilitar a inserção deste tipo de elemento foram projetados dois assistentes devido a:

a) os elementos *table* serem compostos, o que tornaria pouco produtiva a inserção de um elemento componente de cada vez;

b) a maioria dos documentos XML ser composta de dados provenientes de tabelas e grande parte dos exemplos de programas XSLT com saída HTML, apresentarem elementos *table*.

O assistente **Inserir em Tabela** faz parte do conjunto de operações disponível para os nodos do tipo **fonte**, enquanto que o **Inserir Tabela** é válido para os nodos **HTML**.

Assistente de Inserir em Tabela

Começando pelo elemento *table*, que apresenta os conteúdos de `<author>` e `<title>` para cada elemento `book`, deve ser aplicada a operação **Inserir em Tabela** ao elemento `<book>`. Essa operação exibe a janela na qual o elemento `<book>` representa uma linha de *table* (elemento HTML *tr*); os conteúdos das células da *table* (elemento HTML *td*) deverão ser preenchidas com os nodos filhos do elemento `<book>`: `<author>` e `<title>`. É possível atribuir cabeçalhos para as células, no exemplo, esses cabeçalhos recebem os valores **Autor** e **Título**, em formato HTML *h2*

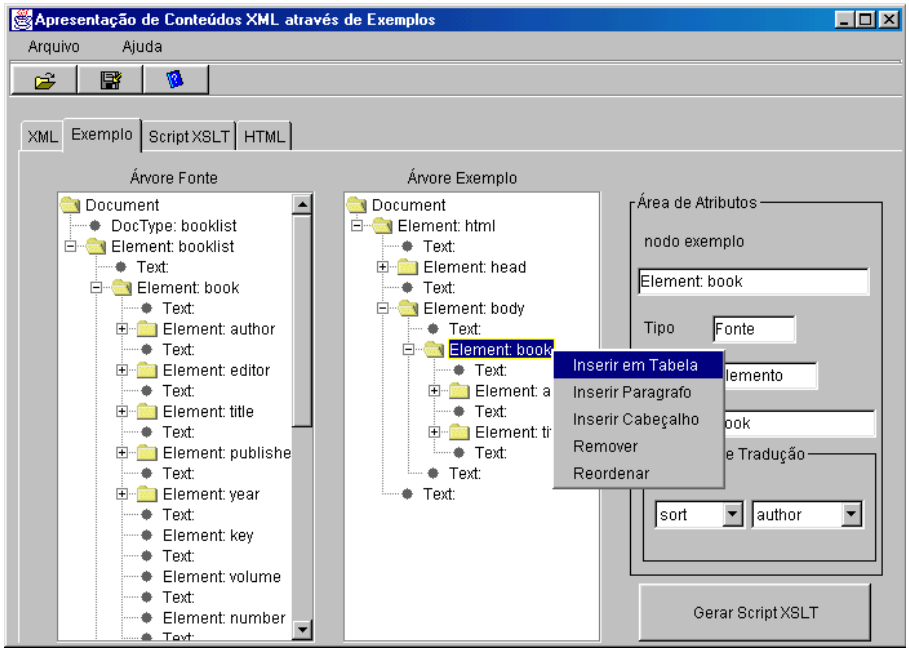


FIGURA 4.8 - Operação <Inserir em Tabela>

A operação **Inserir em Tabela**, faz parte do conjunto de operações disponível para os nodos **fonte do tipo elemento**. Além desta operação, para este tipo de nodo são disponibilizadas as operações:

a) Reordenar - que permite o reordenamento entre elementos de mesmo nível na árvore exemplo;

b) Remover - remove o nodo elemento selecionado e todos os seus descendentes;

c) Inserir Parágrafo - para inserir o elemento HTML com um conteúdo do tipo texto que pode ser alinhado através do atributo align, com os valores left, center, rigth e justify;

d) Inserir Cabeçalho - que insere aos elementos HTML cabeçalho dos tipos: <h1></h1>, <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5> e <h6></h6>, de conteúdo do tipo texto, que podem ser alinhados através do atributo align, com os valores left, center, rigth e justify;

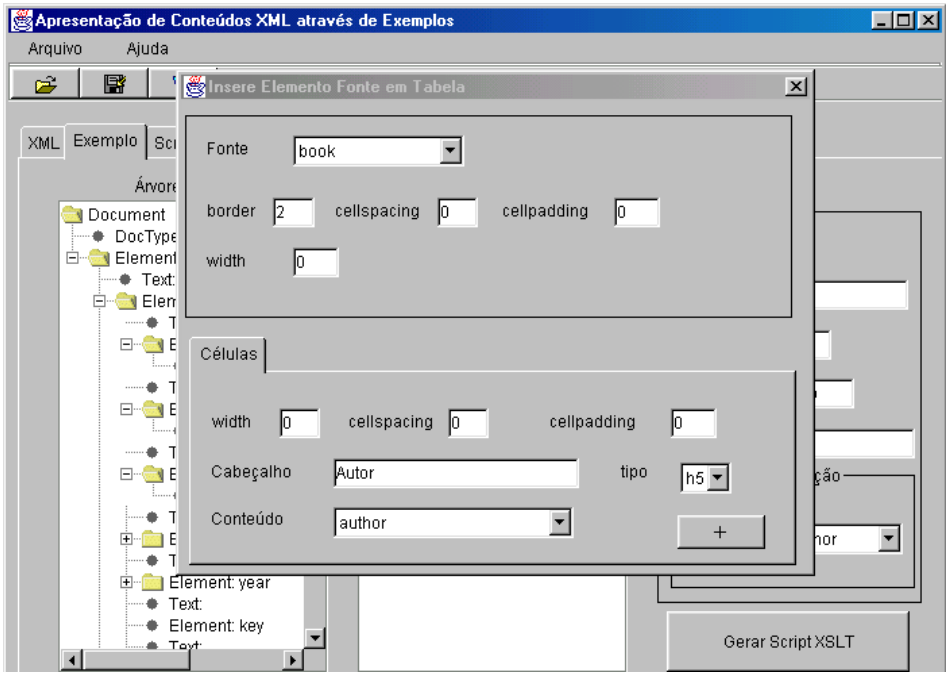


FIGURA 4.9 - Assistente de Inserir em Tabela

Para os nodos **fonte do tipo texto** é disponibilizada a operação **Formatar Texto**, que permite a formatação de seu conteúdo em:

a) cabeçalho dos tipos - h1, h2, h3, h4, h5 e h6 alinhados através do atributo *align*, com os valores *left*, *center*, *rigth* e *justify*;

b) parágrafo com a inserção do elemento alinhado através do atributo *align*, com os valores *left*, *center*, *rigth* e *justify*;

c) alteração do tipo de fonte com a inserção de elementos como, por exemplo: <font face>; , ; etc.

Quando o assistente Inserir em Tabela é finalizado, o seu resultado é refletido na árvore exemplo (figura 4.10).

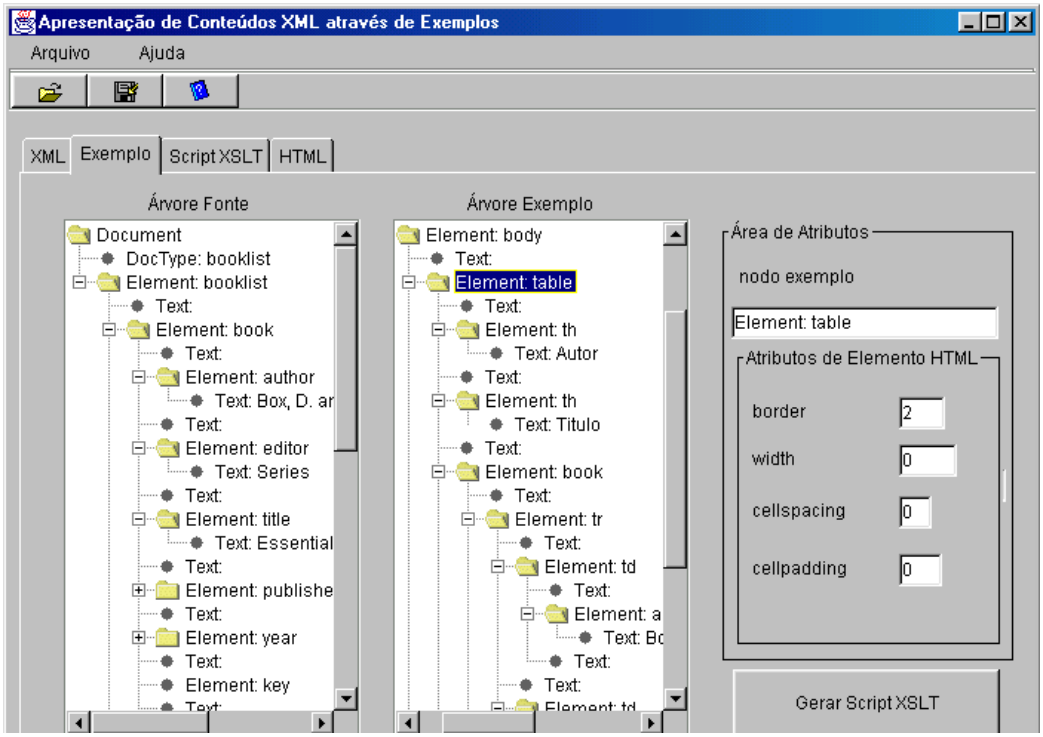


FIGURA 4.10 - Resultado de inserir em tabela na árvore exemplo

Assistente Inserir Tabela

Como outro elemento *table* apresentado no documento resultado não possui conteúdos do documento fonte, o assistente Inserir em Tabela deve ser utilizado para inserir este elemento na árvore exemplo.

Este assistente é acionado através da operação **Inserir Tabela**, disponível no conjunto de operações do elemento HTML *body*. Além desta operação, estão disponíveis as operações: **Colar**, já descrita e **Inserir Cabeçalho**, utilizada para a inserção dos elementos HTML: h1, h2, h3, h4, h5 e h6, que tem como conteúdo uma *string*.

O elemento *table* do exemplo contém uma linha com uma célula, cujo conteúdo é preenchido com a *string* “Lista de Livros”, apresentada como um elemento cabeçalho do tipo *h2*.

Na janela deste assistente um elemento *table* pode ser composto de uma ou mais linhas, cuja inserção deve ser realizada através da ativação do botão **“Insere linha”**. Cada linha pode ser composta de uma ou mais células que devem ser inseridas através do botão **“Insere célula”**

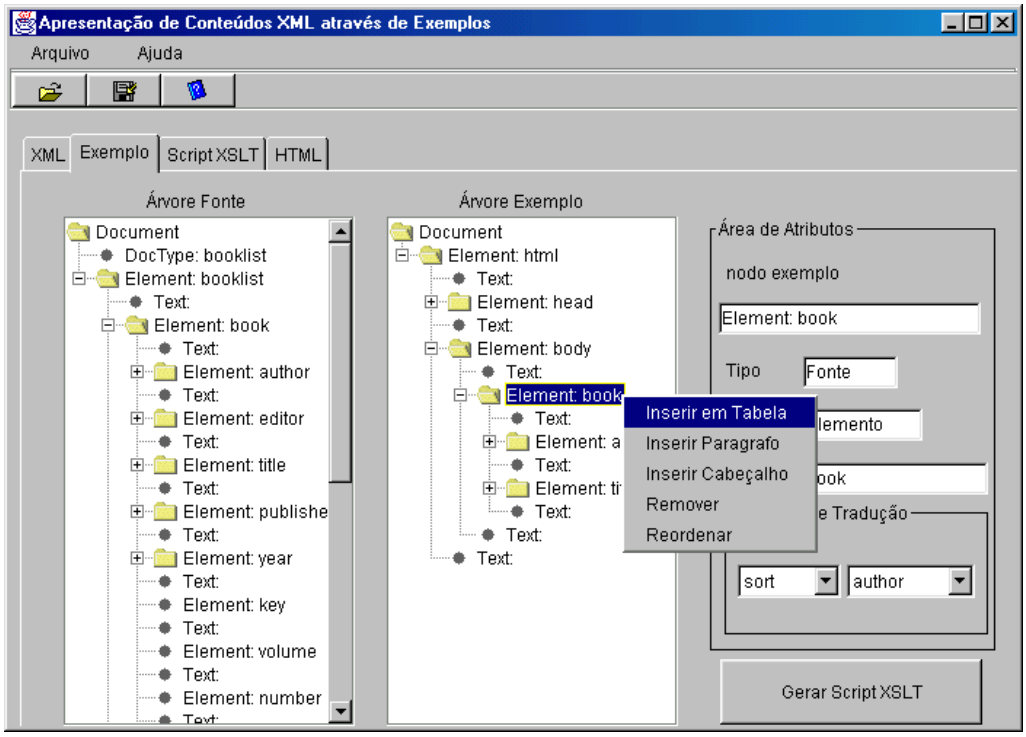


FIGURA 4.11 - Operação <Inserir Tabela>

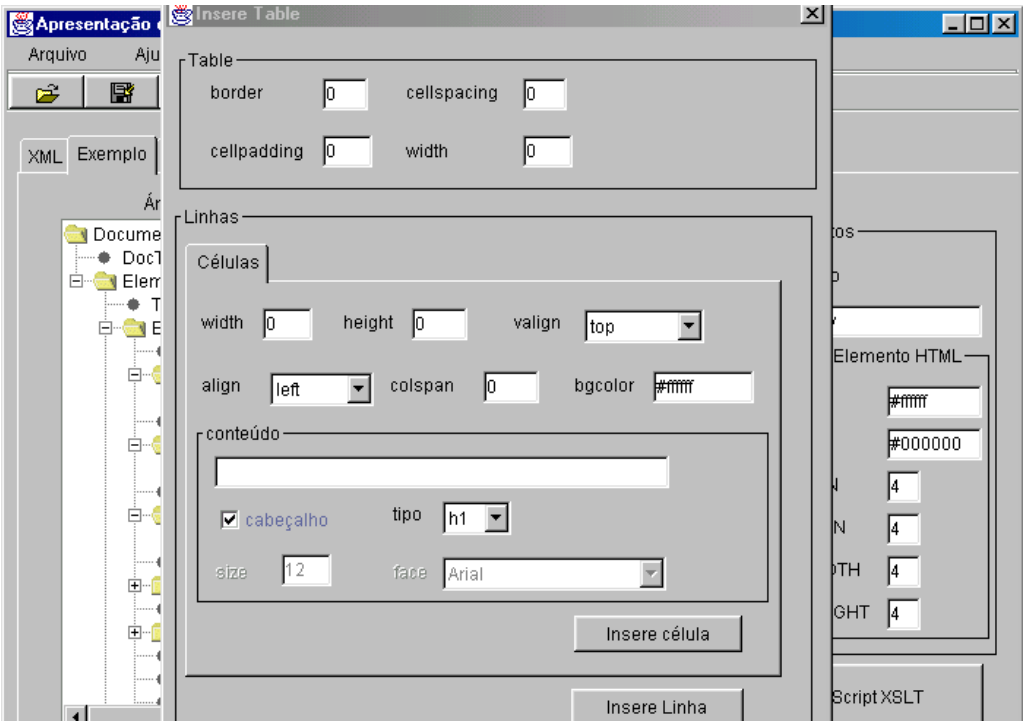


FIGURA 4.12 - Assistente de Inserir Tabela

Da mesma forma, do assistente Inserir em tabela, o resultado da especificação do Inserir Tabela é refletido na árvore exemplo.

Para os elementos HTML a Área de Atributos apresenta a informação **tipo=HTML** e um conjunto de atributos**,** que depende do tipo do elemento HTML

selecionado; por exemplo, para o elemento *body* são exibidos os atributos: *bgcolor*, *text*, *topmargin*, *marginwidth*, *leftmargin* e *marginheight*. Para os elementos *table* são apresentados os atributos: *border*, *cellspacing*, *cellpadding* e *width*.

O resultado final da especificação do exemplo é a árvore apresentada na figura 4.13, que corresponde ao documento exemplo que servirá de entrada para o módulo gerador de regras.

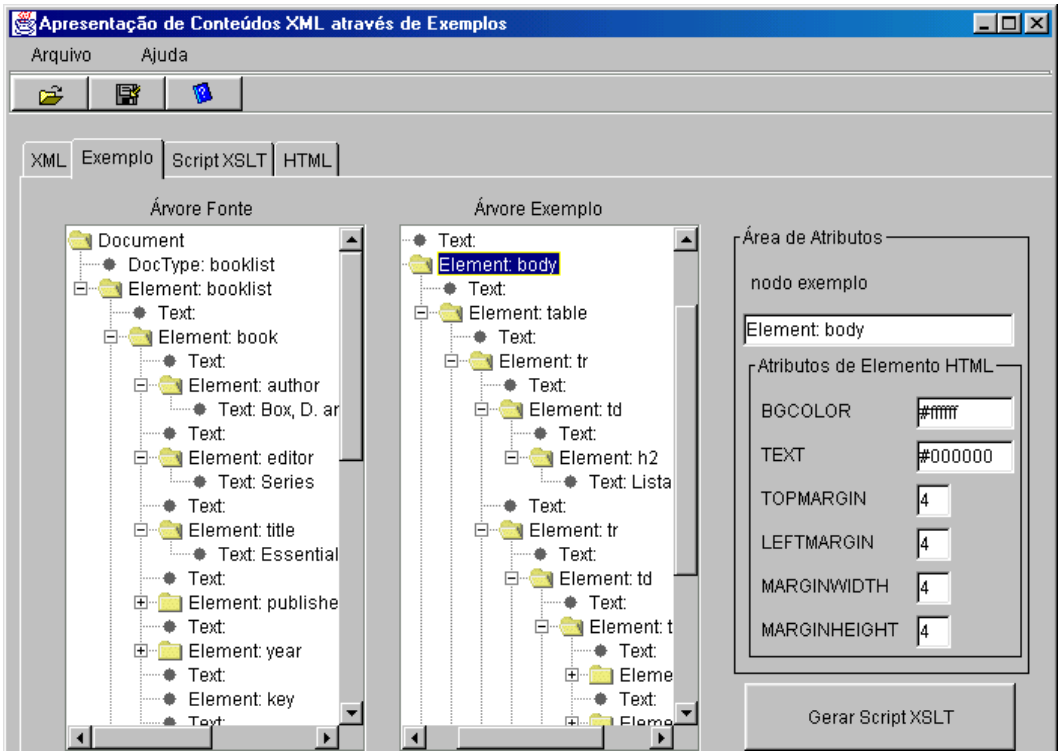


FIGURA 4.13 - Árvore final do documento exemplo

4.1.3 Geração de Regras

Após a especificação do documento exemplo na interface gráfica, este juntamente com uma folha de estilo inicializada no momento da seleção do documento fonte, serão utilizados como entrada para o algoritmo de geração de regras da X2H.

Algoritmo de Geração de Regras

No capítulo 2, foram descritas duas formas de escrita de regras nesta linguagem. Uma utilizando o `<xsl:apply-templates>` e, a outra, com o elemento `<xsl:for-each>`. Concluiu-se que estas duas maneiras são equivalentes. As regras geradas através da X2H, utilizam o elemento `<xsl:for-each>`, para implementar o processamento recursivo da XSLT. A opção por esta forma levou em consideração a simplificação da implementação do gerador de regras, pois, desta maneira, pode-se padronizar a forma de expressar os padrões utilizados para selecionar determinado conjunto de nodos.

A seguir, são descritos e apresentados em forma de algoritmo, os passos, seguidos na geração de regras na X2H. Nesta descrição, são considerados o documento exemplo e a folha de estilo apresentados a seguir.

O Documento exemplo, resultado da especificação do exemplo na X2H, que será utilizado como entrada para o gerador de regras dessa ferramenta é o seguinte:

```
<html>
 <head>
  <title>Lista de Livros</title>
 </head>
 <body text="#000000" topmargin="4" leftmargin="4" bgcolor="#ffffff">
  <table width="610" cellspacing="0" cellpadding="0" border="0">
   <tr>
    <td width="450" height="35" valign="top" align="center"
     colspan="4"  bgcolor="#0086b2">
     <h2>Lista de Livros</h2>
    </td>
   </tr>
  </table>
  <table border="2">
   <th>autor</th>
   <th>titulo</th>
   <book fonte=”true” caminho=”book” sort=”author”>
    <tr>
     <td>
      <author fonte=”true”>Box, D. and Skonnard, A. and Lam,
       J.</author>
     </td>
     <td>
      <title fonte=“true”>Essential XML - Beyond Markup</title>
     </td>
    </tr>
   </book>
  </table>
 </body>
</html>
```

A Folha de estilo inicial que será utilizada na geração das regras é a seguinte:

```
<?xml version='1.0'?>

<xsl:stylesheet xmlns:xsl='http://www.w3.org/XSL/Transform/1.0'>

<xsl:output indent="yes"/>
<xsl:output method="html">

<xsl:template match=”/”>
 <html>
  <head>
   <title>titulo</title>
  </head>
  <body>
   <xsl:apply-templates/>
  </body>
 </html>
</xsl:template>

</xsl:stylesheet>
```

Esta folha de estilo garante um documento de saída HTML com todo o conteúdo do documento fonte.

Passos para gerar regras:

1) o elemento `<apply-templates>` filho do elemento `<body>` é removido da folha de estilo;

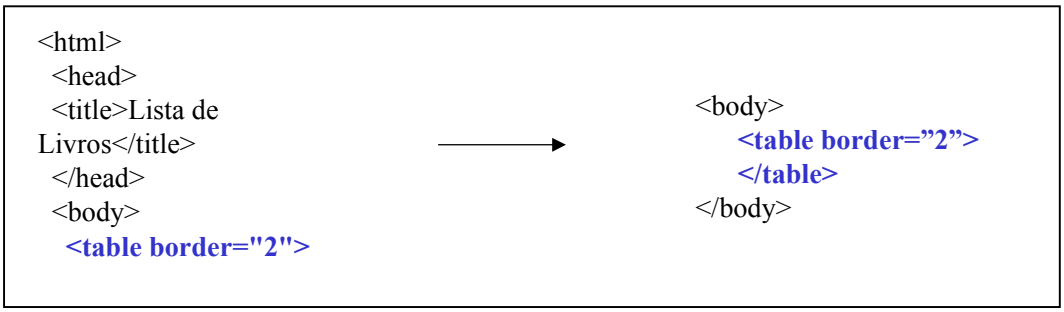
2) o valor do nodo texto filho do elemento `<title>` é substituído pelo valor do elemento de mesmo nome do documento exemplo*;*

3) para cada nodo descendente de `body` no documento exemplo, verificar o tipo: se for um nodo fonte ou se é um elemento HTML. Inserir elemento na folha de estilo de acordo com o resultado do teste, mantendo a mesma hierarquia apresentada no documento exemplo;

{**Verificar se nodo fonte ou HTML**}

```
Para cada nodo exemplo {
se possui atributo fonte
    traduzir fonte
senão
    copiar elemento HTML
} fim Para
gerar folha de estilo
```

4) **copiar elemento HTML**: copia elemento do documento exemplo para a folha de estilo resultado. No exemplo apresentado a seguir, o elemento <table border="2"> do documento exemplo é inserido no corpo do elemento *body* ;



5) **traduzir fonte**: os nodos copiados da árvore fonte são traduzidos em elementos XSLT de acordo com o seu tipo.

{**Verificar tipo de nodo fonte**}

```
se nodo elemento
   traduzir fonte elemento
se nodo texto
   traduzir fonte texto
se nodo atributo
   traduzir atributo
```

5.1) **traduzir fonte elemento**: cada nodo fonte do tipo elemento, gera o elemento `<xsl:for-each>` com o atributo `select` comparado a expressão de caminho do nodo na árvore fonte, se este possuir o atributo `caminho`. Se o elemento não possuir este atributo, o `select` será comparado ao *nome do elemento*.

- Para os elementos com atributo *caminho*, será verificada a presença de predicados. Os predicados são criados a partir do *conteúdo da função predicado.* Por exemplo, se documento exemplo contém um elemento *`<book caminho="book"`*

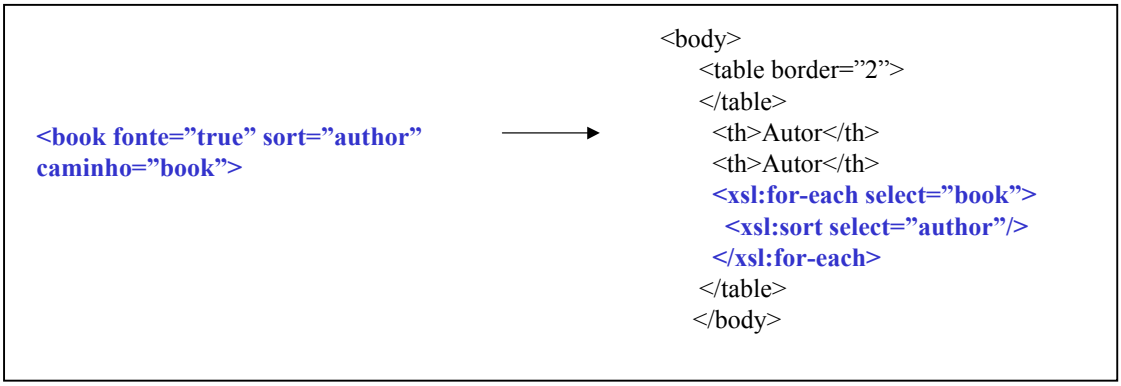
predicado(year="1999")>, isso produziria o seguinte elemento: `<xsl:for-each select="book[year='1999']">`;

- Se o elemento possuir atributos de tradução, estes serão adicionados na folha de estilo de acordo com o elemento XSLT correspondente. Por exemplo, o atributo *sort*, produz o elemento XSLT `<xsl:sort>`.

{**Traduzir fonte elemento**}

```
inserir elemento <xsl:for-each>
se caminho
   atributo match="caminho"
senão
   atributo match="nome-elemento";
se predicados
   incluir predicados
se atributos de tradução sort
   inserir elemento XSLT correspondente
```

O exemplo a seguir ilustra a tradução do elemento fonte: **`<book fonte="true" caminho="book" sort="author">`**.




```
<body>
   <table border="2">
   </table>
    <th>Autor</th>
    <th>Autor</th>
    <xsl:for-each select="book">
     <xsl:sort select="author"/>
    </xsl:for-each>
   </table>
  </body>
```

5.2) **traduzir fonte texto**: para os nodos fonte do tipo texto, será incluído um elemento `<xsl:if>` que testa se o nodo texto está preenchido, e um elemento `<xsl:value-of selec="."/>`, que copia o valor do nodo para a saída.

{**traduzir fonte texto**}

```
<xsl:if test=string-length(.)>0>
 <xsl:value-of select="."
</xsl:if>
```

O exemplo, a seguir, mostra a tradução do nodo texto Box, D. and Skonnard, A. and Lam, J, filho do nodo fonte elemento `<author>`.

```
<author fonte="true">
      Box, D. and Skonnard, A. and Lam,
J.
 </author>
```

→

```
<body>
    <table border="2">
    </table>
     <th>Autor</th>
     <th>Autor</th>
     <xsl:for-each select="book">
      <xsl:sort select="author"/>
      <tr>
       <td>
        <xsl:if test="string-length(.)>0">
         <xsl:value-of select="."/>
        </xsl:if>
       </td>
      </tr>
     </xsl:for-each>
    </table>
   </body>
```

5.2) **traduzir fonte atributo**: para os nodos fonte do tipo atributo: é adicionado o elemento <xsl:value-of> com o atributo *select* comparado ao atributo.

{**Inserir elemento XSLT `xsl:value-of`**}

```
<xsl:value-of select=@nome-do-atributo/>
como filho do elemento <for-each> do elemento pai
```

Saídas geradas

A folha de estilo obtida pelo módulo gerador de regras é apresentada na página *Script* XSLT da interface gráfica da X2H (figura 4.14) e pode ser executada através da chamada de um processador XSLT, que gera como saída o documento resultado HTML (figura 4.15), exibido na página HTML da interface gráfica.

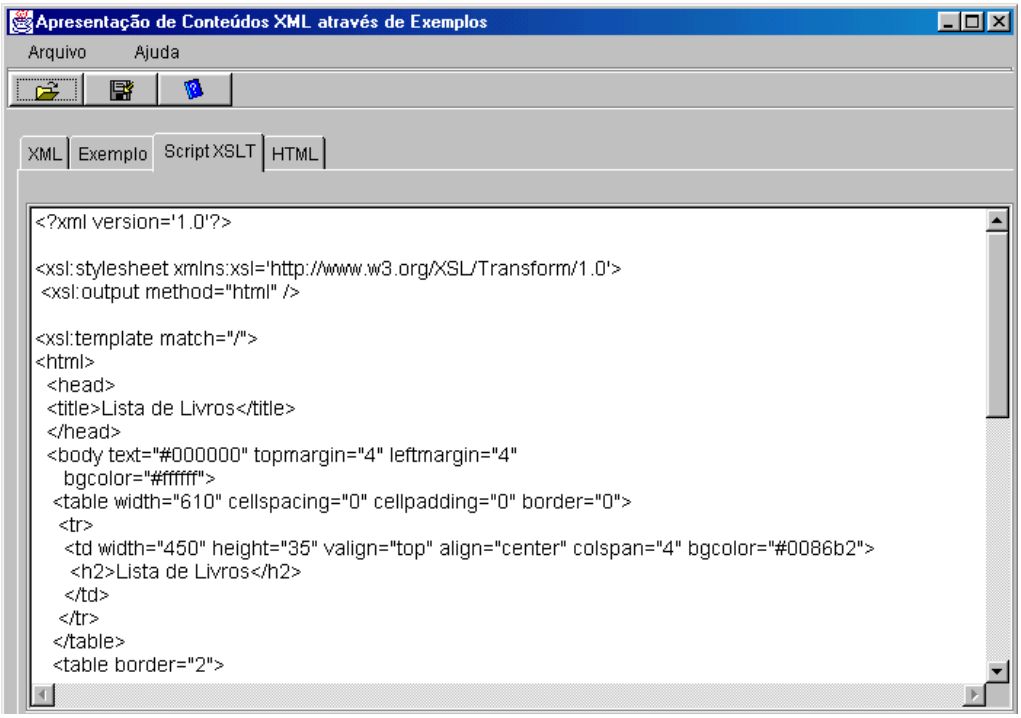


FIGURA 4.14 - folha de estilo resultado

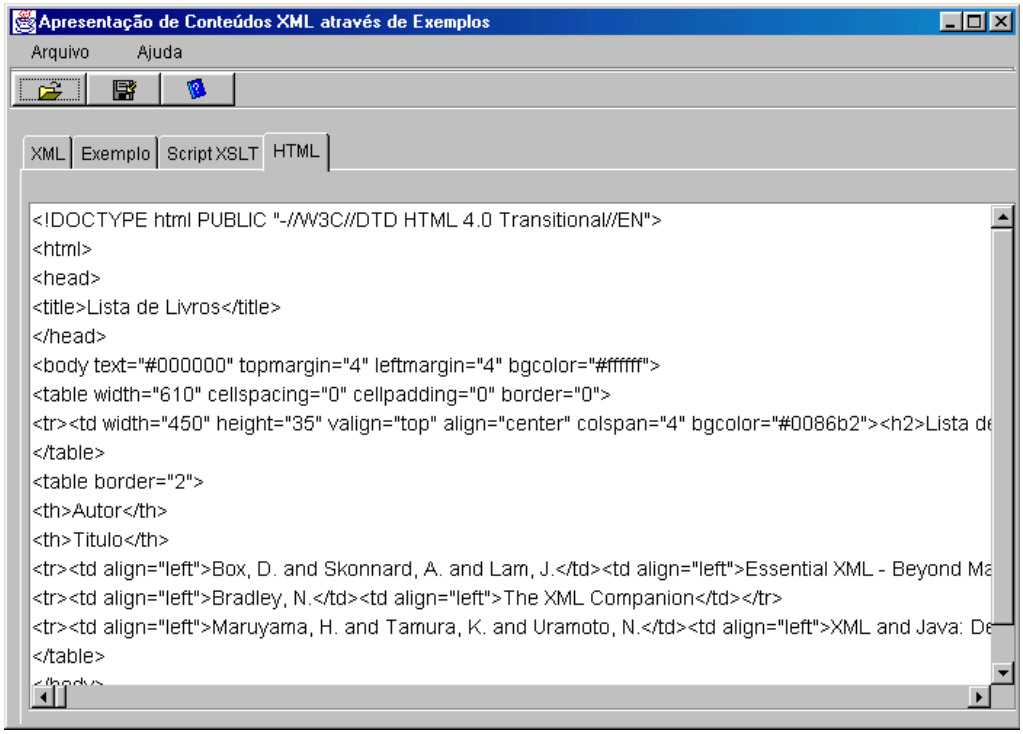


FIGURA 4.15 - documento resultado HTML

4.2 Estudo de caso

Esta seção ilustra a utilização da ferramenta X2H através de um exemplo, no qual o documento fonte XML é o artigo.xml [HEU 2000], apresentado a seguir, de forma reduzida, por questões didáticas. Este documento foi selecionado porque possui uma estrutura diferente da apresentada pelo documento books.xml, utilizado na seção 4.1, para descrever a ferramenta; assim pode ser verificado o comportamento da X2H na presença de elementos, que apresentam relacionamentos 1:n entre si. O documento de saída desejado é o documento HTML das figuras 4.16a e 4.16b.

Os seguintes passos serão realizados na descrição do exemplo: seleção e apresentação do documento fonte; a especificação do documento exemplo; geração do script XSLT e, apresentação do resultado.

Documento artigo.xml

```
<?xml version='1.0'?>
<artigo data="" ultima_revisao="" versao="1">
 <autor>
  <nome>Arnaud Sahuguet</nome>
  <instituicao>University of Pennsylvania</instituicao>
  <endereco></endereco>
 </autor>
 <titulo_artigo>Everything you ever wanted to know about DTD's, but
  were afraid to ask</titulo_artigo>
 <resumo></resumo>
 <secao titulo="What we have benn looking at" numero="s3.1">
  <paragrafo> </paragrafo>
  <paragrafo>...</paragrafo>
```

```
 </secao>
 <bibliografia>
  <referencia id="b1">
   <obra>Data on the Web: From Relations to Semistructured Data and
    XML</obra>
   <autor>
    <nome>Serge Abiteboul</nome>
    <endereco></endereco>
   </autor>
   <autor>
    <nome>Peter Buneman</nome>
    <endereco></endereco>
   </autor>
   <autor>
    <nome>Dan Suciu</nome>
    <endereco></endereco>
   </autor>
   <ano>1999</ano>
  </referencia>
  <referencia id="b2">
   <obra>Union Types for Semistructured Data</obra>
   <autor>
    <nome>Peter Buneman</nome>
    <endereco></endereco>
   </autor>
   <autor>
    <nome>Benjamin Pierce</nome>
    <endereco></endereco>
   </autor>
   <ano>1999</ano>
   <local>University of Pennsylvania</local>
  </referencia>
  <referencia id="b5">
   <obra>Optimizing   regular   path   expressions   using   graph
    Schemas</obra>
   <autor>
    <nome>Mary F. Fernandez</nome>
    <endereco></endereco>
   </autor>
   <autor>
    <nome>Dan suciu</nome>
    <endereco></endereco>
   </autor>
   <ano>1998</ano>
   <local>IEEE Computer Society</local>
  </referencia>
 </bibliografia>
</artigo>
```

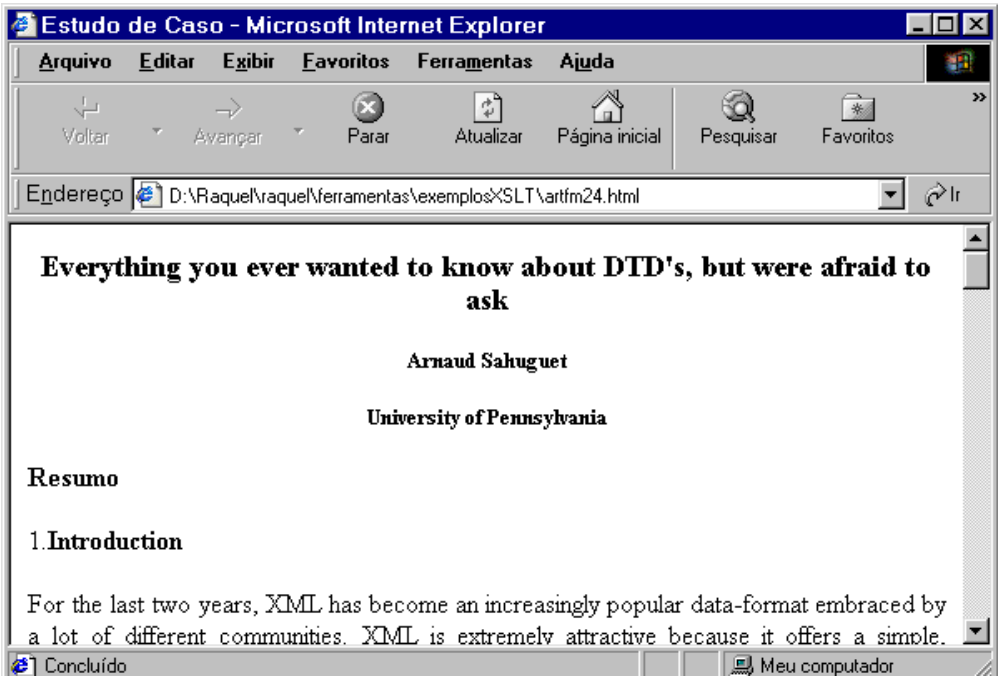


FIGURA 4.16(a) - Documento resultado HTML

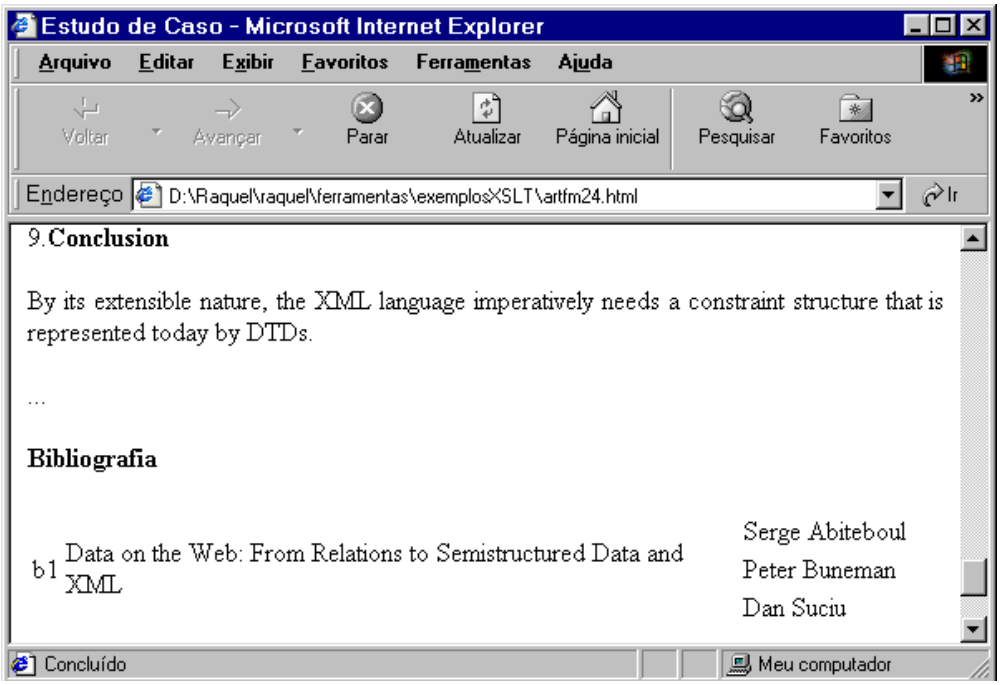


FIGURA 4.16(b) - Documento resultado HTML

4.2.1 Seleção e apresentação do documento fonte

Após a seleção do documento fonte artigo.xml, através da operação de *menu* da ferramenta "Abrir documento XML", serão apresentadas as árvores do documento artigo.xml e do documento exemplo (figura 4.17) e a folha de estilo correspondente será inicializada.

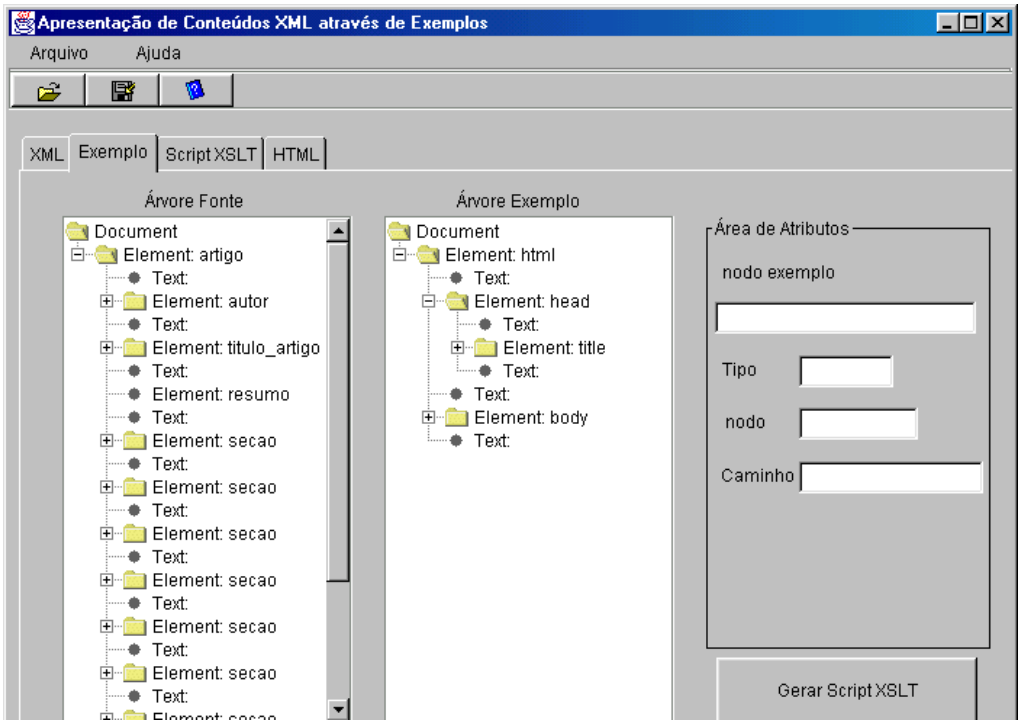


FIGURA 4.17- Árvore do documento artigo.xml

4.2.2 Composição do documento exemplo

A partir das árvores dos documentos fonte e exemplo, o usuário deverá compor um documento exemplo que possa ser utilizado para a obtenção do documento resultado desejado.

O documento resultado, apresentado nas figuras 4.16a e 4.16b, apresenta o documento artigo.xml com as seguintes informações: **título, autor, resumo, as seções (numeradas) com seus parágrafos justificados e as referências bibliográficas do artigo com os dados: id, obra e autores.**

Primeiro, o usuário deverá selecionar os nodos da árvore do documento fonte que servirão de "modelo" ou "exemplo" para a geração da saída. Como se deseja as informações referentes ao artigo, o nodo `<artigo>` deve ser selecionado; a ele deve ser aplicada a operação **Copiar** (figura 4.18).

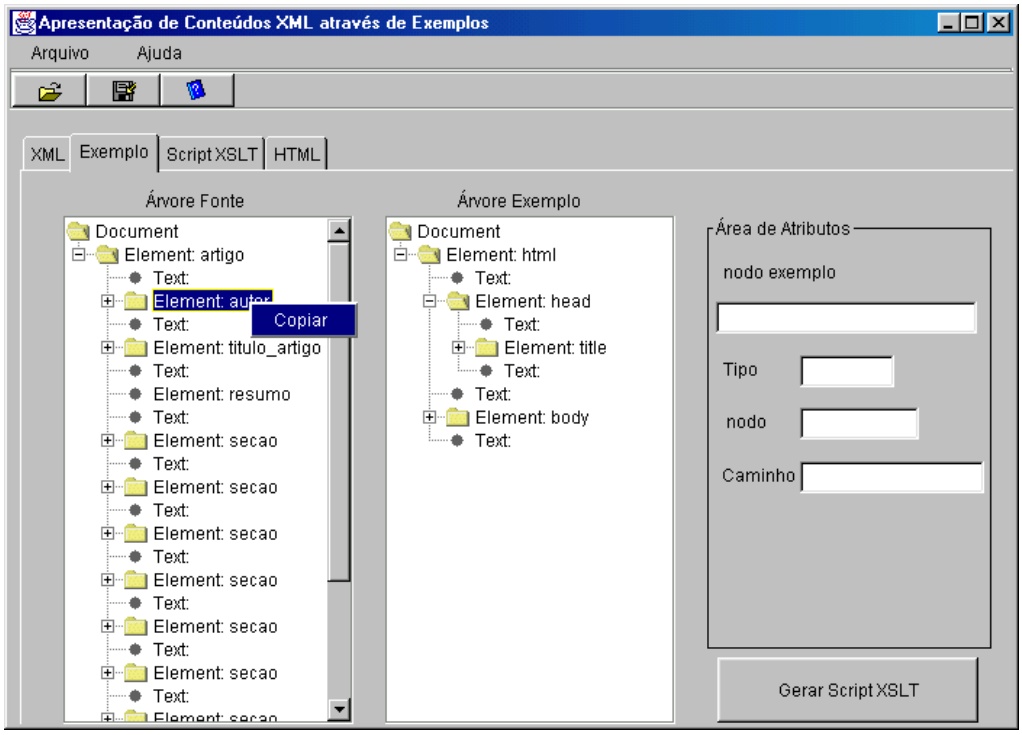


FIGURA 4.18 – Seleção do nodo fonte artigo

A operação **Copiar** exibe a janela **"Selecionar nodos Exemplos"** (figura 4.19), que apresenta a subárvore fonte selecionada, que tem o elemento <artigo> como raiz.

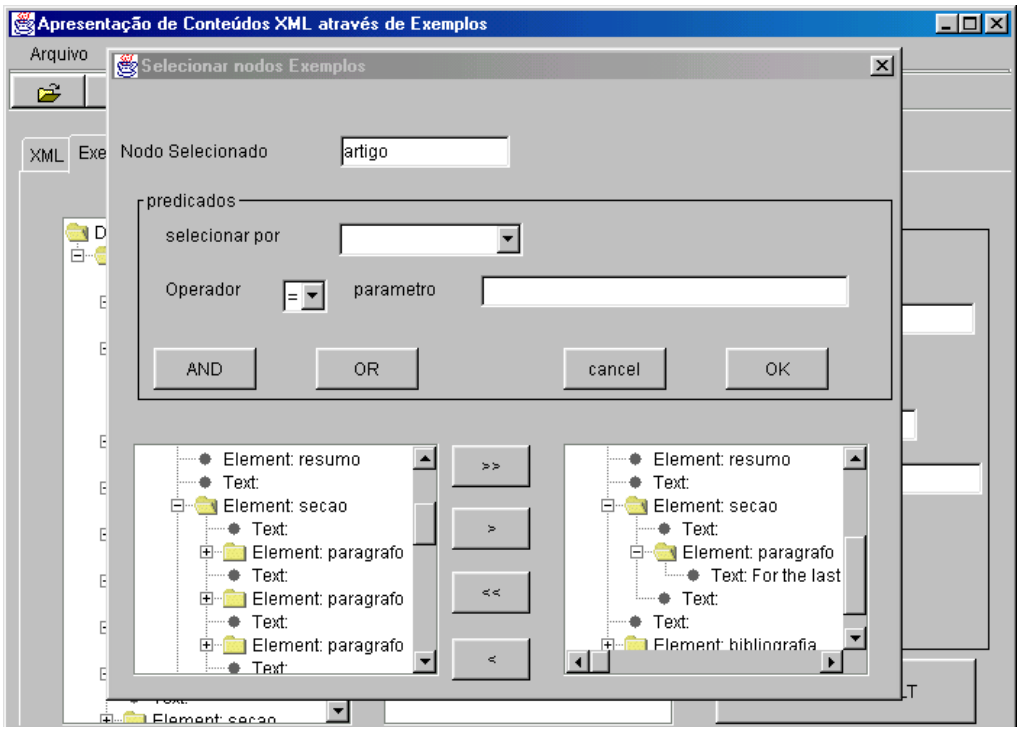


FIGURA 4.19 - Apresentação da subárvore fonte selecionada

A partir desta subárvore, o usuário deverá construir a subárvore exemplo selecionando os nodos filhos de `artigo` que constam no resultado, no caso: `<titulo_artigo>`, `<autor>` (`<nome>` e `<instituicao>`), `<resumo>`, `<secao>`, `<paragrafo>`, `<bibliografia>`, uma `<referencia>` com id, `<obra>` e `<autor>(<nome>)`.

Deve ser selecionado um nodo como "modelo" para cada tipo de elemento; por exemplo, somente um elemento `<secao>` deverá ser copiado, entre os vários presentes para a subárvore exemplo. Neste caso, já que existem muitos elementos repetidos, é mais adequado selecionar um elemento por vez na subárvore fonte e copiá-lo para subárvore exemplo.

Como nenhuma condição foi especificada para a apresentação do documento, não é necessário o preenchimento de nenhum predicado.

A subárvore construída deverá ser inserida na árvore exemplo, o que deve ser realizado através da operação **Colar** aplicada ao elemento *body* da árvore exemplo como mostrado na figura 4.20.

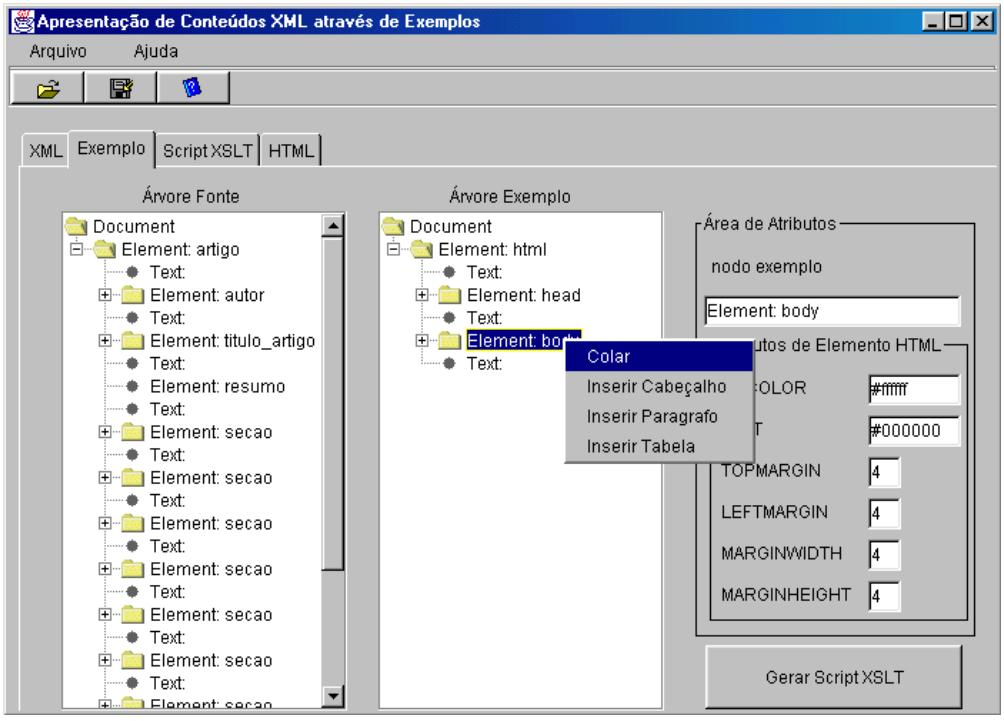


FIGURA 4.20: Inserção dos nodos fontes na árvore exemplo

Após a inclusão dos elementos fontes na árvore exemplo, o usuário deverá inserir os elementos HTML adequados para a obtenção dos efeitos desejados no documento resultado. A seguir serão descritos os passos que devem ser realizado pelo usuário para a inserção destes elementos no documento exemplo, destacando-se o efeito desejado na saída, as operações a serem executadas e os elementos HTML que devem ser inseridos:

a) **título do artigo centralizado no cabeçalho -** o usuário deverá aplicar a operação **Formatar Texto** (figura 4.21) ao nodo texto filho do elemento **`<titulo>`**. Essa operação causa a abertura da janela **"Formatar Texto"** (figura 4.22), na qual deve ser selecionada a opção cabeçalho do tipo **h3** e alinhamento ***center***;

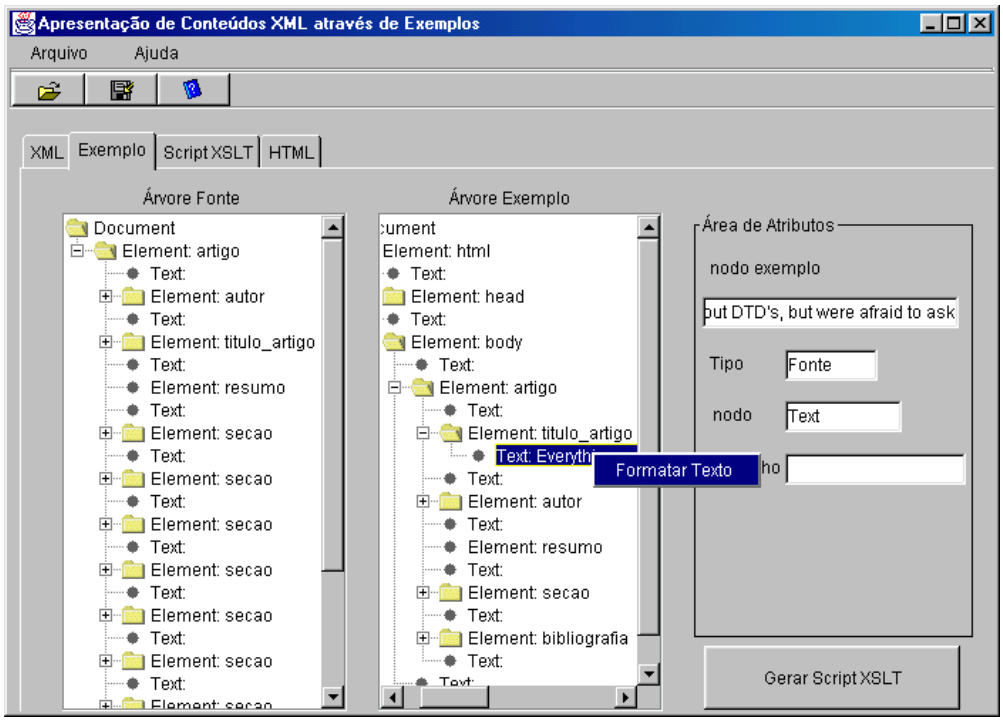


FIGURA 4.21 - Operação <Formatar Texto> para o nodo texto filho de `titulo_artigo`

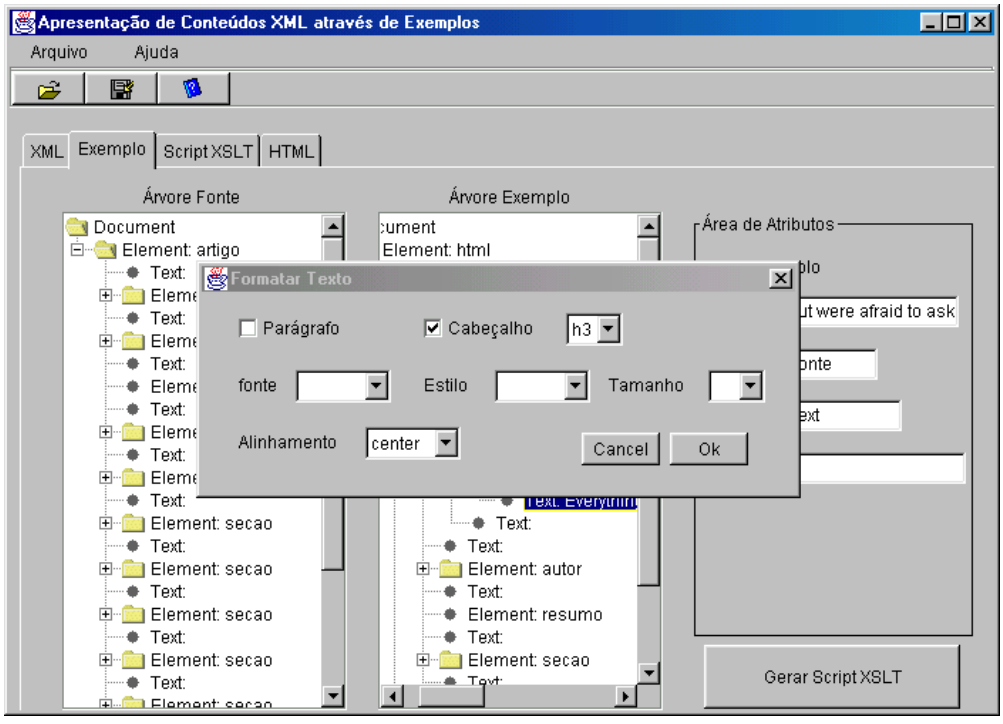


FIGURA 4.22 - Operação <Formatar Texto> para o nodo texto filho de `titulo_artigo`

b) **nome do autor do artigo -** para obter tal efeito, deve ser selecionado o **nodo texto** filho do elemento **<nome>** filho de **<autor>**, e a este deve ser aplicada a operação **Formatar Texto** (figura 4.23). Neste caso, na janela **"Formatar**

Texto" (figura 4.24), deve-se selecionar novamente a opção cabeçalho; desta vez o tipo deve ser h5 com o alinhamento ***center***;

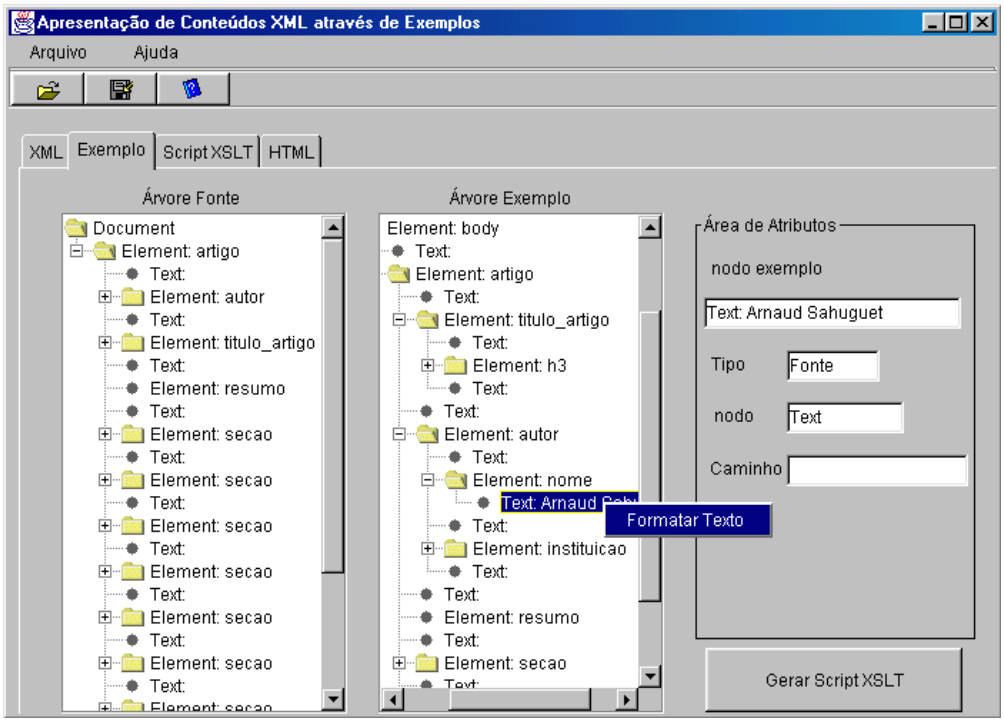


FIGURA 4.23 - Operação <Formatar Texto> para o nodo texto filho de nome

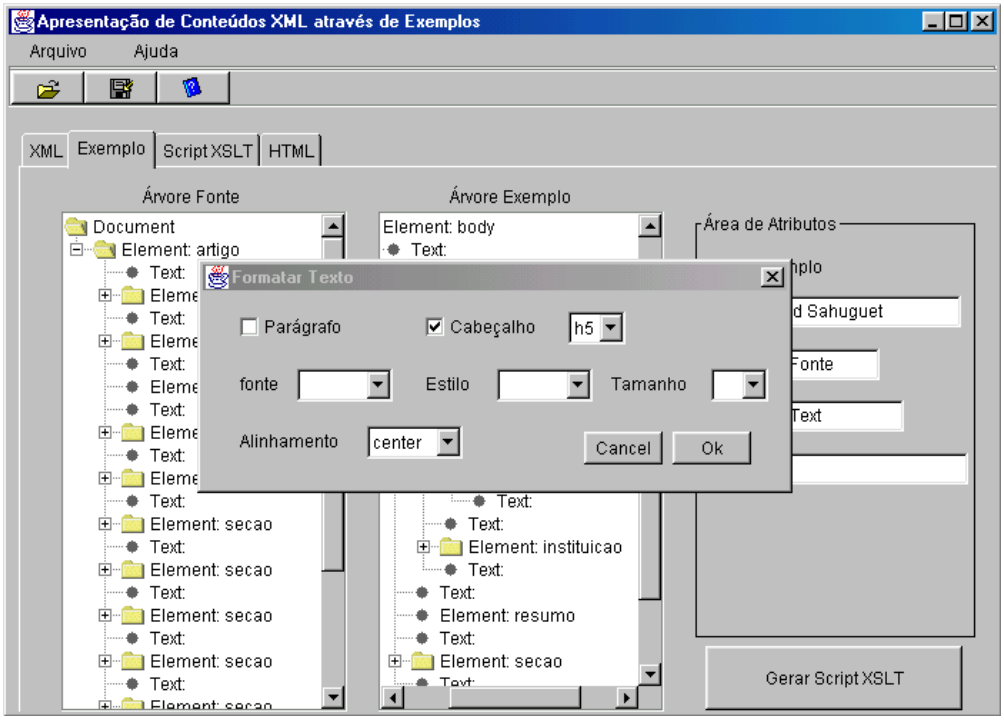


FIGURA 4.24 - Janela “Formatar Texto para o nodo texto filho de nome”.

c) **instituição do autor -** novamente dever ser aplicada a operação **Formatar Texto**, desta vez, no nodo texto filho do elemento **<instituição>** filho do elemento **<autor>**. Na janela **"Formatar Texto",** deve ser selecionado o elemento **cabeçalho do tipo h5** com alinhamento ***center***(figura 4.25);

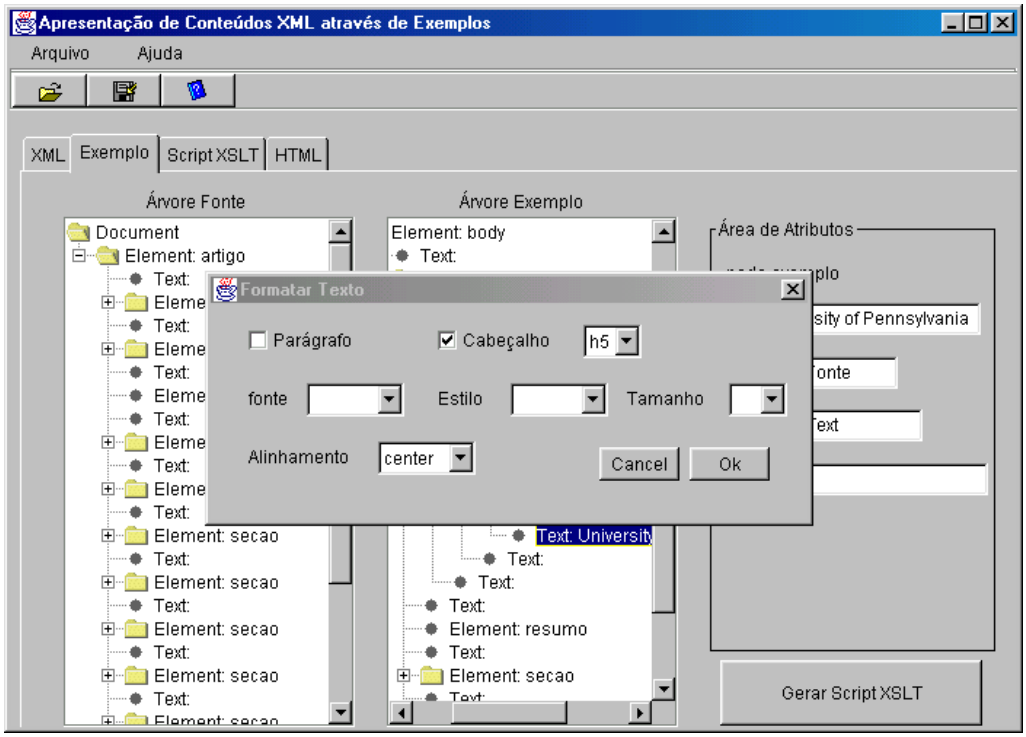


FIGURA 4.25 - Janela "Formatar Texto" para o nodo `instituicao`

d) **apresentação do cabeçalho "Resumo" -** deve ser inserido um elemento **cabeçalho do tipo h4** com alinhamento ***left*** como filho do elemento **<resumo>**, através da operação **Inserir cabeçalho** (figura 4.26) aplicada a este elemento. Esta operação exibe a janela **"Inserir Texto"** (figura 4.27), onde as informações citadas são preenchidas;

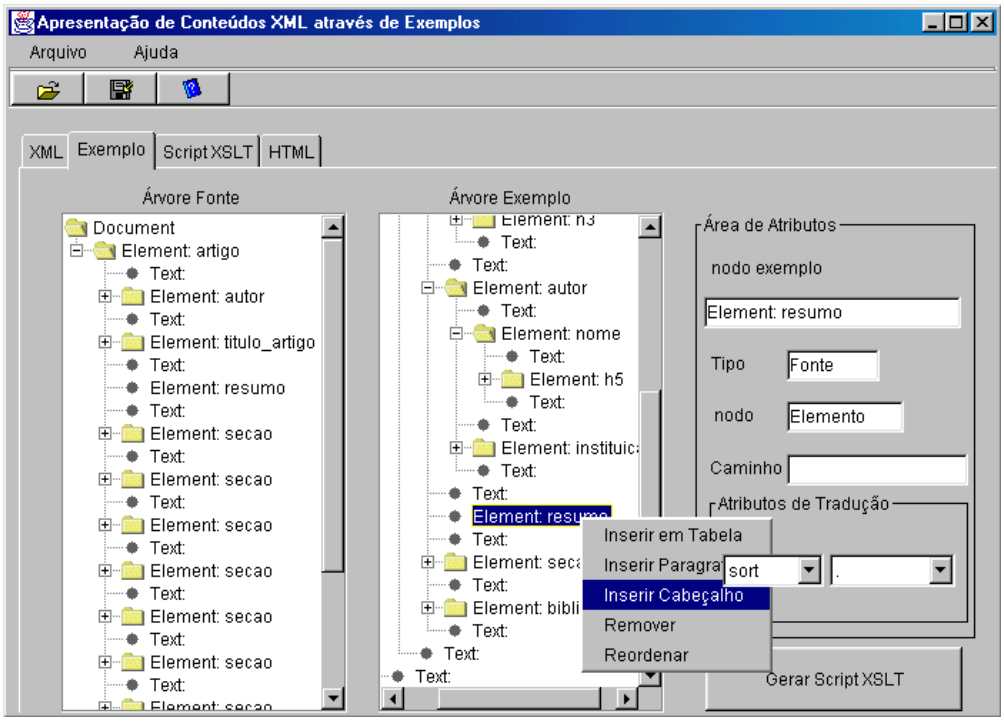


FIGURA 4.26 - Operação <Inserir Cabeçalho> para o elemento `resumo`

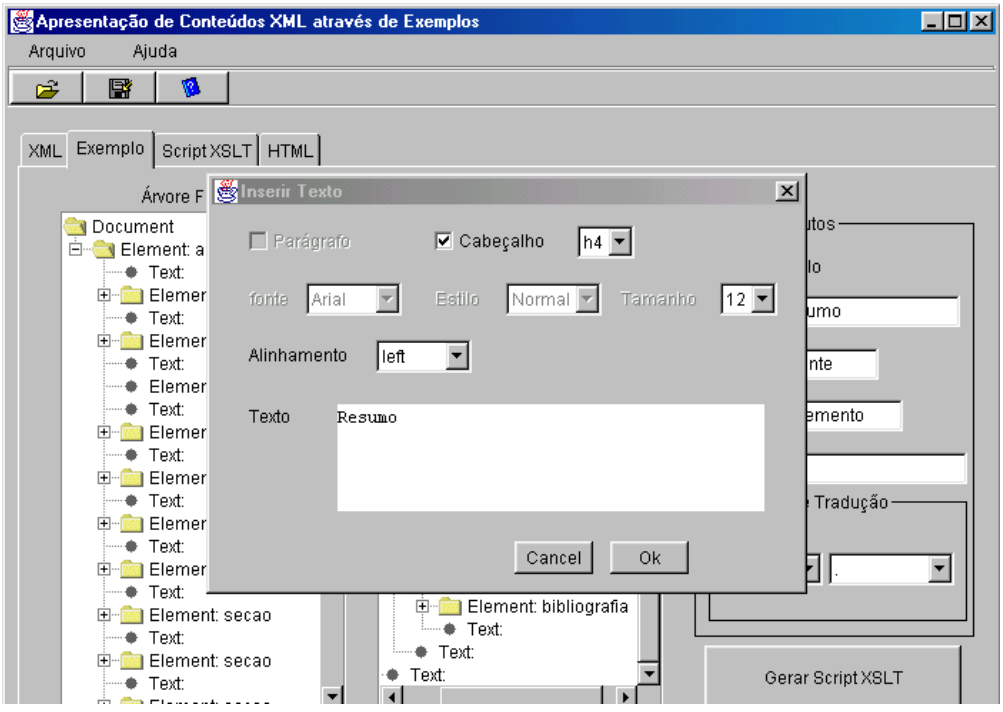


FIGURA 4.27 - Preenchimento da janela "Inserir Texto " para o elemento `resumo`

e) **cada seção do artigo deve ser numerada e seu título deve ser exibido em negrito -** para obter este resultado, deve ser inserido o **atributo de tradução *number*** com **formato "1."**ao elemento **`<secao>`** (figura 4.28) . O conteúdo do atributo **`titulo`** de **`<secao>`**, deve ser formatado em negrito através da operação

Formatar Texto aplicada a este nodo com o preenchimento da informação **estilo negrito** na janela **"Formatar Texto"** (figura 4.29);

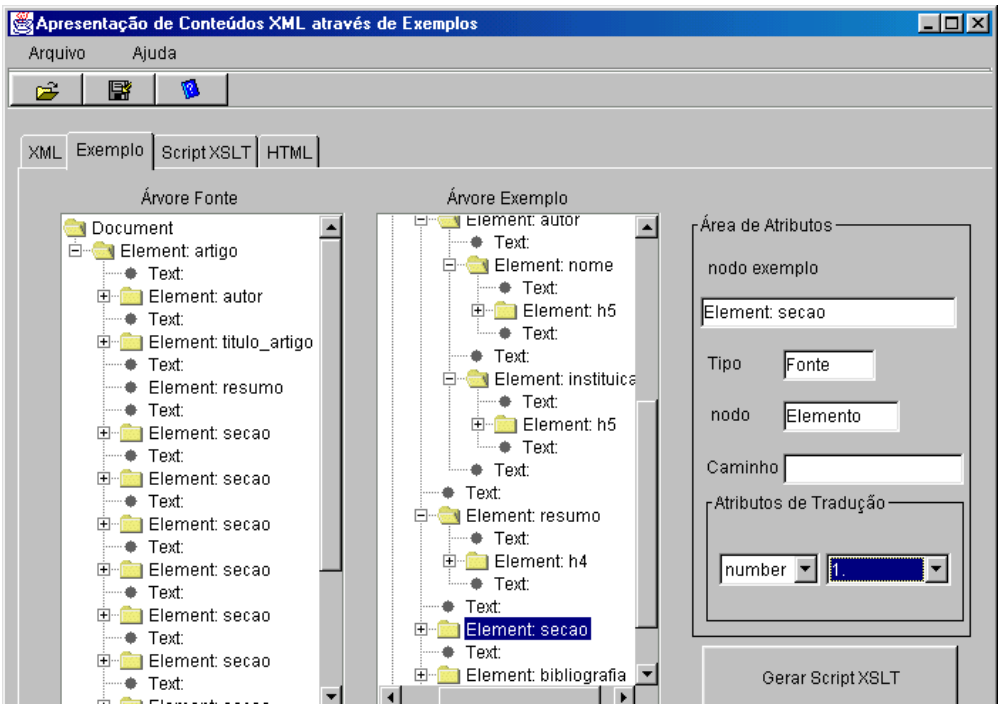


FIGURA 4.28 - Inserção do atributo de tradução *number* o elemento `secao`

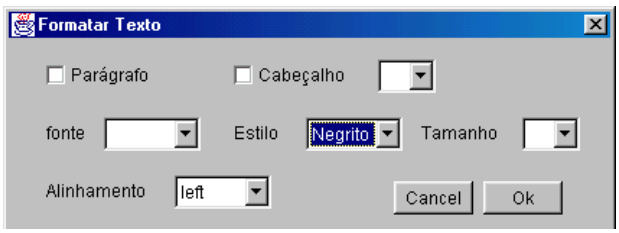


FIGURA 4.29 - Preenchimento da janela "Formatar Texto" para o atributo `titulo` do elemento `secao`

f) **apresentar os parágrafos das seções justificados -** isto pode ser obtido através da operação **Formatar Texto** (figura 4.30) aplicada ao **nodo texto filho elemento `<paragrafo>`**, com o preenchimento da janela **"Formatar Texto"** com a opção **parágrafo** e **alinhamento *justify***;

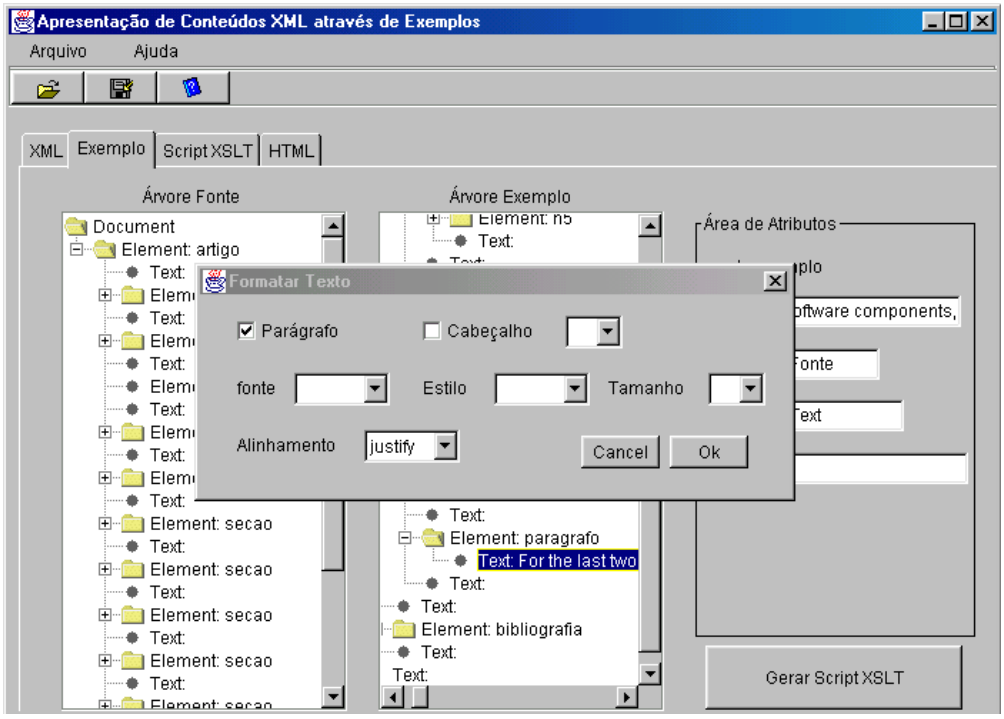


FIGURA 4.30 - Preenchimento da janela "Formatar Texto" para o conteúdo do elemento `paragrafo`

g) **inserir o texto Bibliografia como cabeçalho -** o usuário deverá aplicar a operação **Inserir Cabeçalho** ao elemento **<bibliografia>** e preencher a janela **"Inserir Texto"** com as seguintes informações: cabeçalho **h4**, alinhamento ***left*** e Texto **"Bibliografia"** (figura 4.31);

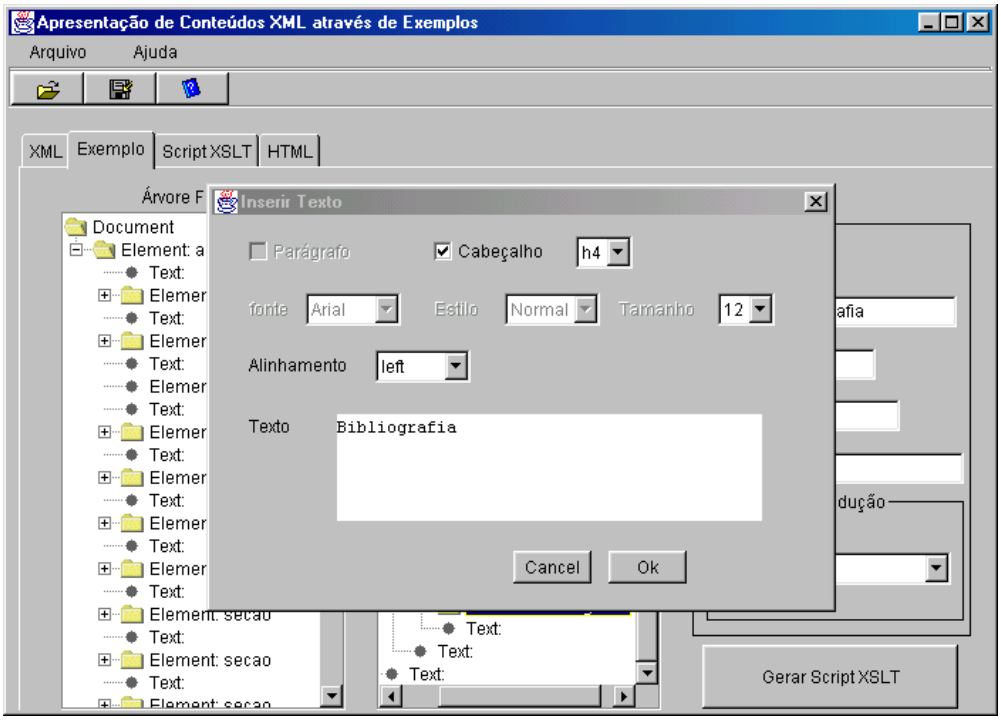


FIGURA 4.31- Inserção do Texto "Bibliografia" como cabeçalho para o elemento `bibliografia`

h) **para cada referência são apresentadas as informações id, obra e autores -** o usuário deverá inserir um elemento ***table***, através da operação **Inserir em Tabela** aplicada ao nodo <**referencia**> (figura 4.32), para indicar que cada linha deste elemento representa uma **referência.** Esta operação exibe a janela **"Inserir em Tabela"**, na qual o usuário deverá preencher as seguintes informações referentes ao elemento ***table***: as células, com os nodos: <**id**>, <**obra**> e <**autor**>**;** e os atributos com os valores: *border= "2"*, *cellpadding= "0"*, *cellspacing= "0"* e *width= "0"* (figura 4.33);

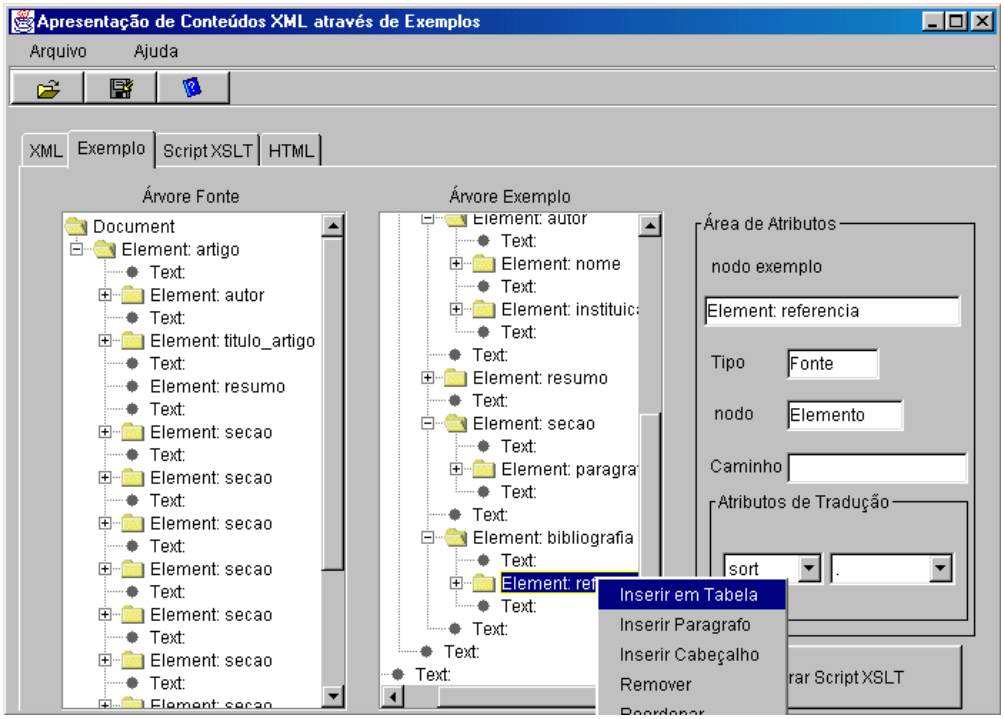


FIGURA 4.32 - Operação <Inserir em Tabela> para o elemento referencia

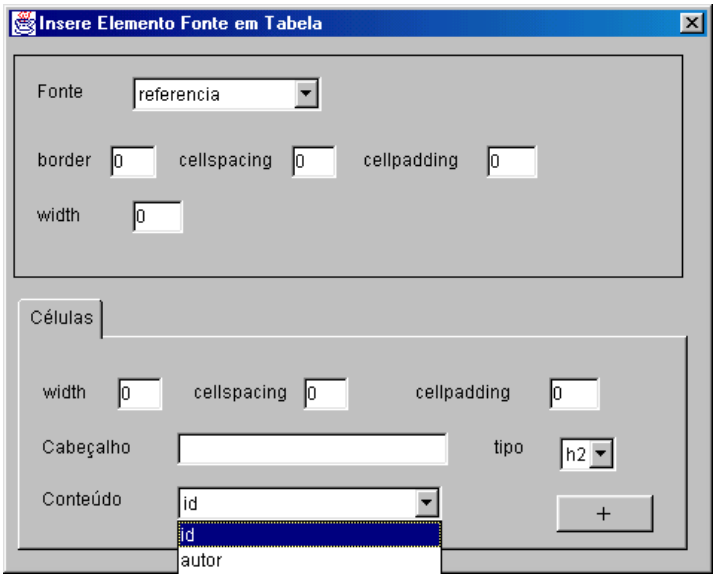


FIGURA 4.33 - Assistente Inserir em Tabela para o elemento referencia

i) **os nomes dos autores do artigo -** como esses elementos são filhos do elemento **<autor>**, é necessário incluir outro elemento *table***, mas**, desta vez, a operação **Inserir em Tabela** (figura 4.34) deverá ser aplicada ao elemento <**autor**>**.** Este elemento *table* deverá ser especificado somente com uma célula, que deve ser preenchida com o conteúdo do nodo <**nome**> e filho de <**autor**>. Os atributos de *table* devem ser preenchidos com os seguintes valores: *border= "0"*, *cellpadding= "0"*, *cellspacing= "0"* e *width= "0"* (figura 4.35).

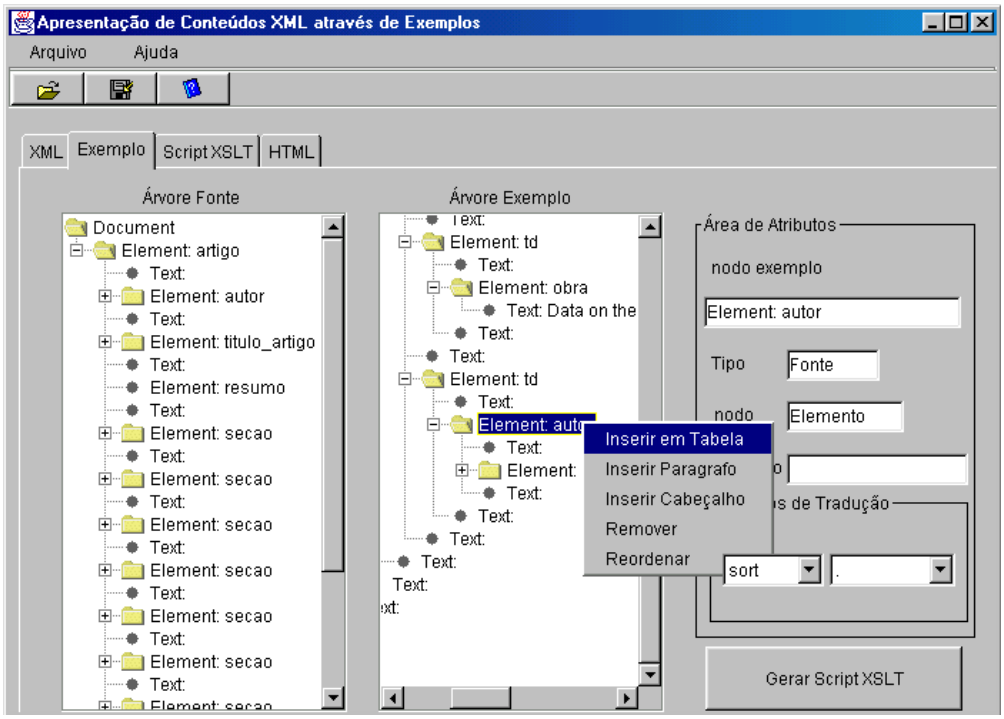


FIGURA 4.34 - Operação inserir em tabela para o elemento `autor`

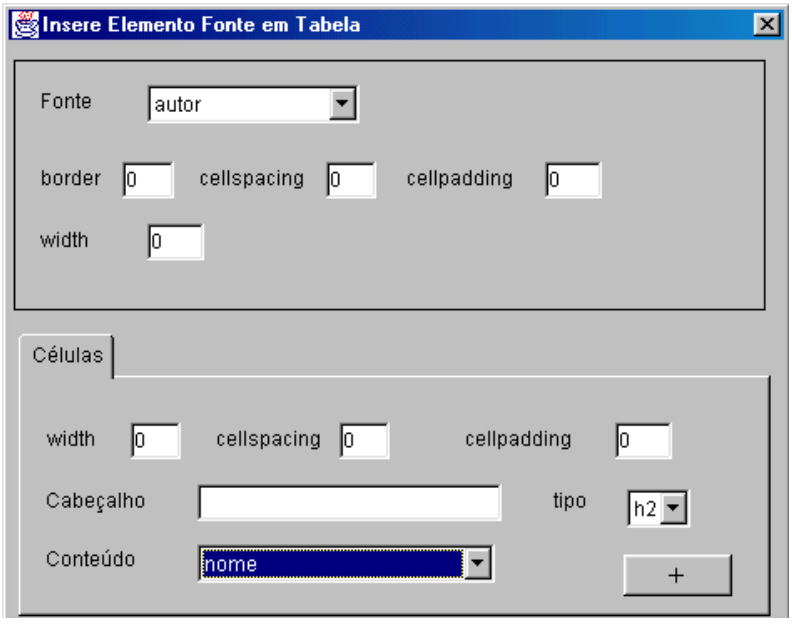


FIGURA 4.35 - Assistente Inserir em Tabela para o elemento `autor`

Com este passo, a composição do documento exemplo pode deve ser finalizada. A figura 4.36 exibe a árvore deste documento em seu estado final.

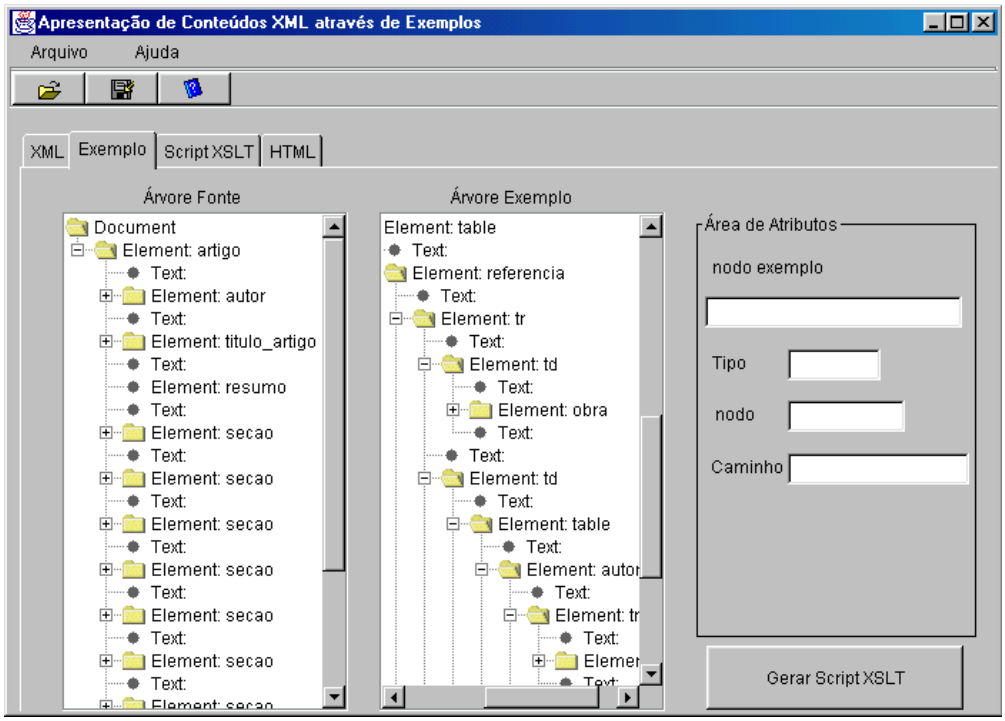


FIGURA 4.36 - árvore final do documento exemplo

Após a composição do documento exemplo, pode ser iniciado o processo de geração de regras. Este deve ser acionado através do botão <Gerar Script XSLT> presente na interface gráfica da ferramenta.

4.2.3 Geração do Script XSLT e apresentação do resultado

A partir do documento exemplo especificado e da folha de estilo inicial, o processo de geração de regras seguirá o algoritmo da ferramenta, descrito na seção 4.3. O resultado deste processo é exibido na página *Script* XSLT na interface gráfica da X2H (figura 4.37).

O *script* XSLT gerado pode ser executado através da opção de menu da X2H "Executar *Script*". O documento resultado HTML é apresentado na página HTML da interface gráfica.(figura 4.38)

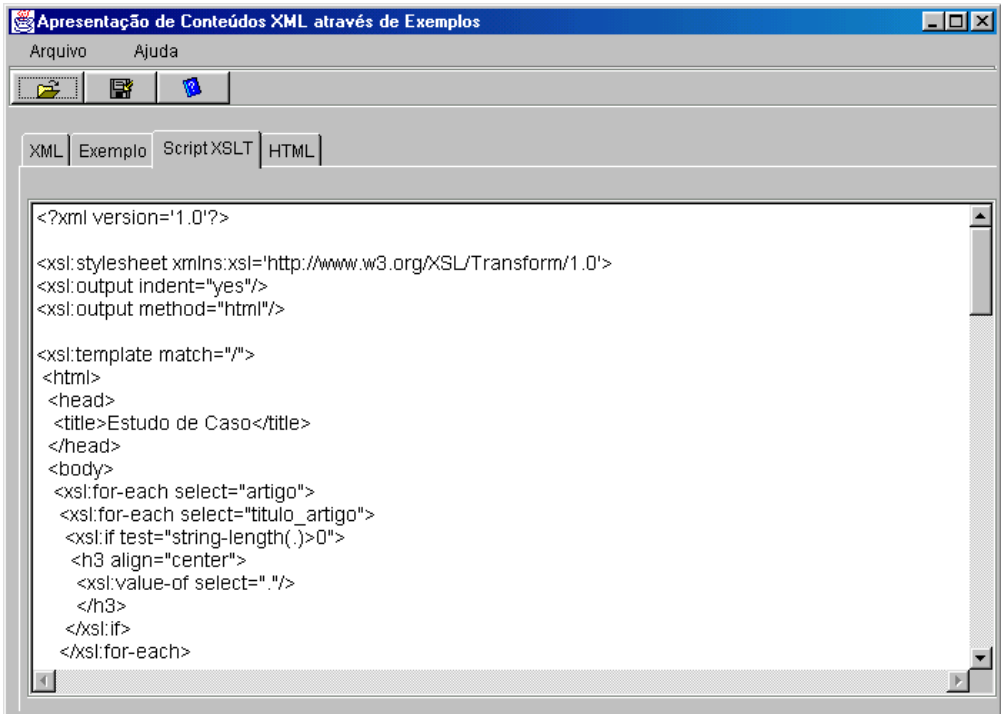


FIGURA 4.37 - folha de estilo gerada

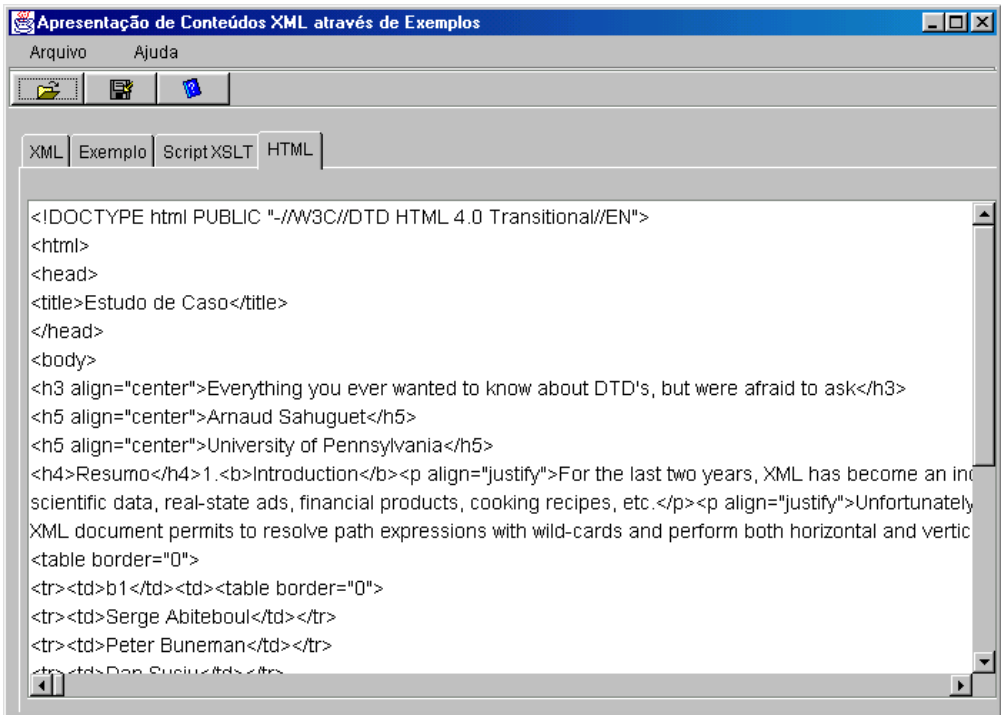


FIGURA 4.38 - Documento resultado HTML

5 Conclusões e Trabalhos Futuros

XSLT tem se destacado como uma linguagem de manipulação de dados XML. Esta linguagem é utilizada para transformar um documento XML em qualquer outro com formato baseado em texto. Apesar disso, algumas dificuldades foram identificadas em um ambiente de programação XSLT puro, o que compromete a produtividade de programação e a disseminação de uso dessa linguagem.

Tais dificuldades serviram de motivação para o desenvolvimento deste trabalho, que propõe a ferramenta X2H, que usa a abordagem orientada a exemplos para auxiliar usuários na apresentação de documentos XML em formato HTML. A X2H gera *script* XSLT a partir de uma especificação realizada por um usuário em uma interface gráfica.

Na interface gráfica da X2H, o documento fonte XML selecionado é apresentado no formato de árvore hierárquica, a partir da qual são selecionados nodos para a composição de uma árvore exemplo, que representa um documento exemplo XML, inicializado com a estrutura de um documento HTML. A composição do documento exemplo ocorre em duas fases. Na primeira fase, são inseridos os elementos exemplos selecionados na árvore fonte. Na segunda fase, são incluídos os elementos HTML.

Para facilitar a composição do documento exemplo, um conjunto de operações foi disponibilizado nas árvores da interface gráfica. Entre elas se destacam o "Assistente de Inserir em Tabela" para os nodos fontes e o "Assistente Inserir Tabela" válido para os nodos HTML. O documento exemplo composto serve de entrada para o módulo gerador de regras, que gera um *script* XSLT.

A opção pelo formato de saída HTML levou em consideração o fato de que, embora este seja o formato mais utilizado em XSLT, não havia sido contemplado pelas ferramentas, que utilizam como recurso para aumentar a produtividade de programação nesta linguagem, a geração de um *script* XSLT.

Como base para este trabalho foram realizados os seguintes estudos, descritos nos capítulos 2 e 3 respectivamente:

a) estudo sobre a linguagem XSLT, no qual foi apresentado o seu modelo de execução, sendo detalhado o modelo em árvore de um documento fonte XML, a estrutura de uma folha de estilo e como ocorre o processo de transformação quando um processador XSLT é acionado para a aplicar uma folha de estilo a um determinado documento fonte XML;

b) estudo de ferramentas que foram desenvolvidas sob a mesma motivação da X2H. Estas ferramentas foram descritas, comparadas e classificadas em dois grupos: as baseadas no reuso de transformações e as geradoras de *script* XSLT. No primeiro grupo foi classificado o sistema CoX. As ferramentas TIXP e *Visual XML Transformation* foram classificadas no segundo grupo. Foi verificado que, apesar do maior uso de XSLT seja para converter documentos XML em HTML, as ferramentas geradoras de *script* XSLT utilizam como formato de saída o XML.

Entre as restrições apresentadas pela X2H, podemos destacar as seguintes, que podem ser sugestões para trabalhos futuros:

a) a X2H gera HTML. Seria importante disponibilizar para o usuário a opção de selecionar um formato de saída entre os formatos possíveis para XSLT. O documento exemplo seria baseado na estrutura do formato selecionado como alvo;

b) os documentos XML fonte e exemplo são representados por árvores hierárquicas, que são estruturas de dados próximas ao modelo de dados de XML. Mas, seria melhor para o ser humano que estes documentos fossem apresentados em um formato mais amigável. Portanto, seria de grande valia o estudo de interfaces para apresentação de documentos XML. Uma sugestão a ser avaliada, seria a apresentação destes documentos em forma de tabelas aninhadas ou no formato apresentado pelo editor de XML do software XML Spy [ALT 2000a];

c) foram utilizados exemplos didáticos para descrever a ferramenta. Seria interessante aplicar à ferramenta casos de uso reais e analisar se resultados obtidos estão de acordo com os requisitos;

d) X2H é um protótipo, que está sendo desenvolvido na linguagem java, utilizando o ambiente Jbuider da Borland. É importante que este protótipo seja aperfeiçoado;

e) Foi utilizada uma classe adquirida em [ARM 2001], que exibe documentos XML como uma árvore DOM, porém esta interface não representa os nodos atributos da árvore XML. Portanto, esta classe deveria ser estendida para a visualização e manipulação destes tipos de nodos.

Após a conclusão desse trabalho, foi encontrada a ferramenta XSLT Designer, que é parte do pacote XML SPY [ALT 2002b] versão 4.3, lançada em fevereiro de 2002, que gera automaticamente *script* XSLT para transformar documentos XML em HTML.

A interface da XSLT Designer apresenta-se dividida em três páginas: Design, XSLT stylesheet e IE Preview. A especificação das transformações é realizada na página *Design*, na qual a estrutura do documento fonte XML é apresentada, a partir da qual os elementos podem ser selecionados e incluídos em elementos HTML em uma área que representa uma página HTML. As páginas XSLT stylesheet e IE Preview são utilizadas respectivamente para apresentar a folha de estilo XSLT gerada e o documento resultado HTML

Fazendo uma comparação entre esta ferramenta e a X2H, as seguintes considerações podem ser apresentadas:

a) a X2H auxilia na apresentação de qualquer documento XML em HTML, não sendo necessário que este esteja vinculado a um esquema;

b) é mais fácil estender a X2H para suportar a apresentação de documentos XML em qualquer outro formato diferente de HTML, uma vez que, nesta ferramenta, a estrutura do documento exemplo pode ser alterada para comportar estes outros formatos. Na XSLT *Designer*, a especificação das

transformações é feita em uma janela que representa uma página HTML, o que a torna muito dependente deste tipo de formato;

c) a XSLT *Designer* não apresenta elementos equivalentes aos atributos de tradução da X2H, o que limita as operações que podem ser realizadas pelos usuários na apresentação de determinados documentos;

d) a abordagem por exemplos utilizada pela X2H a torna mais intuitiva.

Bibliografia

[ALT 2002a] ALTOVA. **XML Editor with Intelligent Editor**. Disponível em: <http://www.xmlspy.com/features_editing.html>. Acesso em: 10 maio 2002.

[ALT 2002b] ALTOVA. **The XML SPY Company. XSLT Designer**. Disponível em: <http://www.xmlspy.com/products_xsl.html>. Acesso em: 10 maio 2002.

[ARM 2001] ARMSTRONG, Eric. **Working with XML, the Java API for Xml Processing (JAPX) Tutorial**. Disponível em: <http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/index.html>. Acesso em: 22 de ago. 2001.

[BRA 2000] BRADLEY N. **The XML Companion.** 2nd ed. Harlow: Addisson-Wesley, 2000.

[EDE 2001] EDER, Johann; STRAMETZ, Walter. Composition of XML-transformations. In EC-Web, INTERNATIONAL CONFERENCE ON ELETRONIC COMMERCE AND WEB TECHNOLOGIES, 2 , 2001. **Proceedings...** Munique: [s.n], 2001. p.72-80.

[EVA 2001a] EVANGELISTA FILHA, I. M. R. **Uma interface para consulta a dados semi-estruturados através de exemplos**. 2001. Dissertação (Mestrado em Ciência da Computação) – Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte.

[EVA 2001b] EVANGELISTA FILHA, I. M. R.; LAENDER, A H. F. ;SILVA, A S. Querying Semistructured Data By Example: The QSByE Interface. In WIIW INTERNATIONAL WORKSHOP ON INFORMATION ON WEB, 2001. **Proceedings...**Rio de Janeiro: [s.n], 2001. p.156 -163

[HEU 2000] HEUSER, Carlos A. **Curso de XML**. 2000. Disponível em: <http://metropole.inf.ufrgs.br/curso/xml/node2.html>. Acesso em: 5 jun. 2001.

[HOL 2000] HOLMAN. K. G. **XSL transformations (XSLT) 1.0**. W3C Recomendation, Nov. 1999. Disponível em: <http://www.CraneSoftwrights.com>. Acesso em: 4 jun. 2001

[HAR 2001] HAROLD E. R**. The *XML Bible, Second Edition*** : XSL Transformations. 2001. Disponível em: <http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html>. Acesso em: 10 jan. 2002

.

[KAD 2001] KADE. A. M. **Uma linguagem visual de consulta a XML baseada em ontologia**. 2001. 70 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[KAY 2000] KAY, Michael. **XSLT Programmer's Reference.** [S.l.]: Wrox Press, 2000.

[KAY 2001] KAY, Michael. **What kind of language is XSLT?** 2001. Disponível em: <http://www-106.ibm.com/developerworks/xml/library/x-xslt/?dwzone=xml> . Acesso em: 10 mar. 2002.

[LAU 99] LAU, Christina. **Visual XML tools: b**ridging business applications with XML. 1999. Disponível em: <http://www-106.ibm.com/developerworks/library/visualtools>. Acesso em: 10 out. 2001

[NAS 2001] NASCIMENTO, Alexandre. **Intercâmbio de dados entre aplicativos utilizando XML/XSLT**. 2001. 81f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[RIB 99] RIBEIRO NETO, B.; LEANDER, A. H. F.; SILVA, A. S. Extracting semi-structured data through examples. In: ACM INT. CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, CIKM, 1999, Kansas City, Missouri. **Proceedings....**. Kansas: [s.n], 1999. p 94-101.

[SIL 99] SILVA, E. S. **Extração de Dados Semi-Estruturados Baseada em Exemplos**. 1999. 80 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte.

[W3C 98a] W3C, World Web Consortium, Extension. **Document Object Model (DOM) Level 1 Specification, Version 1.0**. U.S.A, October 1st, 1998. W3C Reccomendation. Disponível em: <http://www.w3c.org/TR/REC-DOM-Level-1-1998-1001.pdf>. Acesso em: 10 ago. 2001

[W3C 98b] W3C, World Web Consortium, Extension. **Extensible Markup Language (XML 1.0)**. U.S.A, February 10st, 1998. W3C Reccomendation. Disponível em: <http://www.w3c.org/TR/REC-xml-19980210.html>. Acesso em: 20 maio 2001.

[W3C 99a] W3C, World Web Consortium, Extension. **XML Path Language (Xpath) Version 1.0** . U.S.A, November 16st, 1999. W3C Reccomendation. Disponível em: <http://www.w3c.org/TR/xpath-19991116.html>. Acesso em: 20 ago. 2001.

[W3C 99b] W3C, World Web Consortium, Extension. **XSL Transformation (XSLT) Version 1.0** . U.S.A, November 16st, 1999. W3C Reccomendation. Disponível em: <http://www.w3c.org/TR/xslt-19991116.html>. Acesso em: 22 maio 2001

[W3C 2000] W3C, World Web Consortium, Extension. **Stylesheet Language (XSL) Version 1.0**. U.S.A., March 27th , 2000. W3C Working Draft. Disponível em: <http://www.w3c.org/TR/2000/WD-xsl-20000327.html>. Acesso em: 15 mar. 2002.

[ZLO 75] ZLOOF M. Query by Example. In: NATIONAL COMPUTER CONFERENCE, 44., 1975. **Proceedings...**. Montvale: AFIPS, 1975. p. 431-438.