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

Intercâmbio de Dados entre Aplicativos utilizando XML/XSLT

por

ALEXANDRE ERASMO KROHN NASCIMENTO

Trabalho de conclusão submetido à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Informática

Prof. Carlos Alberto Heuser
Orientador

Porto Alegre, dezembro de 2001

CIP- CATALOGAÇÃO NA PUBLICAÇÃO

Nascimento, Alexandre Erasmo Krohn
Intercâmbio de Dados entre Aplicativos utilizando XML/XSLT / por Alexandre Erasmo Krohn Nascimento. - Porto Alegre: PPGC da UFRGS, 2001.
81f. il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Heuser, Carlos Alberto.

1. Intercâmbio de dados. 2. Integração de sistemas. 3. XML. 4. XSLT. I. Heuser, Carlos Alberto. II. Título.

Agradecimentos

Agradeço aos meus colegas e amigos José Henrique Amaral dos Santos, por dispender parte de seu tão escasso tempo na discussão de idéias que foram utilizadas no desenvolvimento deste trabalho, e Sérgio Henrique Ramos, pelas dicas e apoio na programação em Java. Ambos foram grandes incentivadores deste projeto.

Agradeço também ao meu orientador, Prof. Carlos Alberto Heuser, pela sua orientação, que foi feita sempre na forma de sugestões, e não de imposições. Obrigado, professor.

E agradeço a meu amigo e colega do mestrado Alexandre Braatz, por estar sempre disposto a ajudar.

Para Ivânia, por sua dedicação, amor e paciência.

Sumário

Lista de Abreviaturas ou Siglas.....5
Lista de Figuras.....6
Resumo.....7
Abstract.....8

1 Introdução.....9

2 Integração de sistemas heterogêneos.....11

3 XML utilizado no intercâmbio de dados.....14
3.1 XML.....14
3.2 Tecnologias de transformação do XML.....18
3.2.1 XSL.....18
3.2.1.1 Transformação da árvore.....19
3.2.1.2 Formatação.....19
3.2.1.3 Benefícios do XSL.....20
3.2.1.4 Um exemplo do XSL:.....21
3.2.2 XSLT.....23
3.2.2.1 Exemplo de XSLT.....24
3.2.3 XPath.....25
3.3 Outras possibilidades de utilização do XML.....27
3.4 API's para XML.....28
3.4.1 SAX.....28
3.4.2 DOM.....29

4 Framework para integração de aplicações heterogêneas.....31
4.1 Arquitetura geral do framework.....31
4.2 Exemplos da utilização do framework.....34
4.2.1 Troca de informações com diferenças de estruturas.....34
4.2.2 Troca de informações com mapeamento dos dados.....36
4.3 O Gerador de stylesheets.....38
4.4 TIXP – The Integrated XML Program.....39

5 Conclusões.....76
5.1 Trabalhos futuros.....77

Bibliografia.....79

Lista de Abreviaturas ou Siglas

API	Application Program Interface
DOM	Document Object Model
DTD	Document Type Definition
HTML	HiperText Markup Language
JDBC	Java DataBase Connectivity
JVM	Java Virtual Machine
OMG	Object Management Group
SAX	Simple API for XML
SGBD	Sistema Gerenciador de Bancos de Dados
SGML	Standard Generalized Markup Language
UML	Unified Modeling Language
WAP	Wireless Application Protocol
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XPath	XML Path
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation

Lista de Figuras

FIGURA 3.1 - Diagrama de um livro estruturado em árvore.....19
FIGURA 3.2 - Diferentes possibilidades de apresentação do mesmo documento XML. 20
FIGURA 3.3 - Árvore formada por DOM para um documento XML.....30
FIGURA 4.1 - Aplicativos trocando informações através de XML.....32
FIGURA 4.2 - O framework proposto para integração dos aplicativos.....33
FIGURA 4.3 - Tela principal do TIXP.....40
FIGURA 4.4 - Folder "XML".....41
FIGURA 4.5 - Os menus do TIXP.....42
FIGURA 4.6 - TIXP com o documento XML 1 já carregado.....42
FIGURA 4.7 - TIXP com os dois documentos XML carregados.....43
FIGURA 4.8 - As árvores XML expandidas.....44
FIGURA 4.9 - Representação textual do documento XML 1.....45
FIGURA 4.10 - O menu "Insert".....46
FIGURA 4.11 - Inserção do primeiro nodo na tabela de mapementos.....47
FIGURA 4.12 - Escolha do nodo correspondente no documento XML 2.....48
FIGURA 4.13 - O segundo nodo é inserido na tabela de mapeamentos.....49
FIGURA 4.14 - Seqüência de escolha dos nodos.....50
FIGURA 4.15 - A tabela de selected nodes preenchida.....51
FIGURA 4.16 - A opção "Generate XSLT – Row".....52
FIGURA 4.17 - A opção "Generate XSLT – Table".....53
FIGURA 4.18 - Visão reduzida do stylesheet gerado.....54
FIGURA 4.19 - Visão ampliada do stylesheet gerado.....54
FIGURA 4.20 - O botão "Apply XSLT" em ação.....55
FIGURA 4.21 - O documento gerado pelo stylesheet XSL.....56
FIGURA 4.22 - A visualização do arquivo convertido em forma de árvore.....57
FIGURA 4.23 - A opção "Create/Edit Conversion Table".....58
FIGURA 4.24 - Tabela de conversão de valores.....59
FIGURA 4.25 - A opção "Add Row" na tabela de conversão de valores.....60
FIGURA 4.26 - Uma tabela de conversão completa.....60
FIGURA 4.27 - Nova geração de templates XSL.....61
FIGURA 4.28 - Novo stylesheet XSL, com mapeamento de domínios.....62
FIGURA 4.29 - Documento resultante com valores substituídos.....63
FIGURA 4.30 - Documento XML resultante em forma de árvore hierárquica.....64
FIGURA 4.31 - Elementos não referenciados não são substituídos.....65
FIGURA 4.32 - A opção "Restart TIXP" do menu "File".....66
FIGURA 4.33 - A estrutura do documento exemplo paciente.xml.....67
FIGURA 4.34 - Documento XSL aberto pelo TIXP.....68
FIGURA 4.35 - A opção "Apply XSL Transformation" do menu "File".....69
FIGURA 4.36 - O documento transformado em forma textual.....70
FIGURA 4.37 - O documento transformado em forma de árvore hierárquica.....70
FIGURA 4.38 - A forma HTML interpretada do documento transformado.....71
FIGURA 4.39 - Stylesheet XSL com erro de sintaxe.....72
FIGURA 4.40 - Comunicação de erro encontrado no documento XSL.....73
FIGURA 4.41 - O menu "Look".....74
FIGURA 4.42 - A forma de apresentação "CDE/Motif", para UNIX com X-Windows. 74
FIGURA 4.43 - A forma de apresentação "Windows" para ambientes Microsoft.....75

Resumo

A integração de aplicações heterogêneas é uma tarefa constante entre empresas do mundo moderno. A grande quantidade de fornecedores de software, aliada à extensa variedade de técnicas e linguagens computacionais utilizadas, fazem desta integração uma tarefa trabalhosa e cara para as organizações. As alternativas existentes para a integração de sistemas de diferentes fornecedores podem variar, desde acesso compartilhado a uma base de dados comum, uso de replicadores de dados entre bases de dados distintas, troca de mensagens entre aplicações, ou o uso de programas exportadores/importadores, gerando arquivos em um formato previamente protocolado entre os desenvolvedores dos softwares envolvidos.

Este trabalho visa propor uma alternativa para a integração de sistemas heterogêneos, fazendo uso da tecnologia XML para representar os dados que são trocados entre os aplicativos. Para tanto, sugere um *framework* a ser utilizado no planejamento da arquitetura dos softwares. O objetivo principal da adoção de um *framework* é a utilização de uma metodologia previamente desenvolvida e certificada, economizando tempo de análise para a solução de um problema.

O *framework* proposto subtrai dos desenvolvedores a necessidade de alteração do código fonte dos seus programas cada vez que a integração com um novo fornecedor de software se faz necessária, ou que há alteração no formato dos dados trocados entre os aplicativos. Este efeito é conseguido através da utilização de XSLT para a conversão de formatos de documentos XML trocados pelos softwares. Tal conversão é realizada por um processador XSLT externo aos programas envolvidos.

Para simplificar o processo, foi desenvolvido o protótipo de uma ferramenta para a geração de *templates* XSLT. *Templates* são elementos da especificação XSLT capazes de realizar a transformação entre estruturas representadas em XML. O gerador de *templates* XSLT é uma ferramenta gráfica capaz de converter mapeamentos realizados entre estruturas XML em *templates* XSLT, podendo aplicar as transformações geradas a documentos XML, com a finalidade de teste ou transformação.

Palavras Chave: Intercâmbio de dados, Integração de sistemas, XML, XSLT.

TITLE: "DATA INTERCHANGE BETWEEN APPLICATIONS USING XSL/XSLT"

Abstract

The integration between heterogeneous applications has been a constant task at the corporations all over the world. The amount of software producers, allied to the variety of used computacional technics, and computer languages makes from this an arduous and expensive work to the organizations. Existent alternatives to different products integration may vary, from shared access of a common database, to data replication between databases, message exchange between applications, or export/import programs utilization, generating previously protocoled files by software makers.

This work has as objective to propose an heterogenean systems integration alternative, using XML technology to represent data being changed between applications. For to do so, it suggests a framework, which will be used on softwares architecture planning. The framework adoption main's objective is to use an already developed and certified metodology, saving problem solving analisys time.

The proposed framework subtract from developers the needed of source code changes, each time an integration to a new software producer makes it necessary, or there is format changes on data exchanged between the applications. This effect is got by XSLT utilization to format conversion of the XML documents used by the envolved softwares. An external XSLT processor is encharged of this conversion.

For to simplify the integration process, a prototype of a XSLT template generation tool was developed. Templates are XSLT especification elements, XML represented structure transform capable. The XSLT template generator is a graphical tool, that converts done mapping between XML structures to XSLT templates, and may apply this XSLT to the XML Documents, for tests or transformation.

Keywords: Data Interchange, Systems Integration, XML, XSLT.

1 Introdução

A idéia da realização deste trabalho surgiu de uma necessidade da empresa onde o autor trabalha, a InfoSaúde – Gestão Informatizada em Saúde. A empresa desenvolve sistemas de informação para instituições de saúde: hospitais, clínicas, empresas de planos de saúde de auto-gestão e afins. A InfoSaúde possui clientes de vários portes, distribuídos por todo o Brasil, Argentina e Uruguai.

Freqüentemente, os clientes da InfoSaúde solicitam à mesma que seus sistemas sejam integrados com sistemas já existentes, das próprias instituições, ou de terceiros. Essas integrações apresentam-se de formas variadas. Os softwares da empresa são desenvolvidos sob a forma de módulos, de forma que um cliente, no caso um hospital, pode informatizar seu centro cirúrgico através do Módulo Centro Cirúrgico, da InfoSaúde, e o ambulatório através do software de uma outra empresa. Para exemplificar, suponha-se o caso de um paciente que foi atendido no ambulatório, mas que devido a alguma complicação, deva ser operado imediatamente. Suas informações, introduzidas no sistema de ambulatório devem ser, de alguma forma, disponibilizadas para o sistema de informações do centro cirúrgico.

As variáveis existentes nessa transferência de informações são muitas. Os sistemas a ser integrados podem estar armazenados no mesmo banco de dados ou não; os Sistemas Gerenciadores de Bancos de Dados utilizados podem ser de fornecedores diferentes, com dialetos distintos de acesso às informações; os aplicativos geralmente não são desenvolvidos na mesma linguagem de programação; ou mesmo as plataformas computacionais utilizadas para o armazenamento das informações e/ou execução dos aplicativos pode vir a ser diferente. Na prática, a heterogeneidade computacional é a regra.

As necessidades de integração dos clientes devem ser atendidas, sob pena de gerar insatisfação dos mesmos. Porém, para cada cliente que requisita uma integração, analistas de sistemas e programadores devem ser alocados para a execução da adaptação do sistema existente, ou desenvolvimento de um módulo integrador, o que representa um custo que, dependendo do grau de integração necessário, pode suplantar mesmo o custo de venda ou locação dos módulos a ser integrados, visto que estes já encontram-se concluídos.

Alternativas para a integração de sistemas heterogêneos existem. É possível utilizar *gateways*, para permitir acesso direto às bases de dados utilizadas pelos aplicativos que se deseja integrar; pode-se utilizar replicadores de dados, ou algumas outras propostas que são abordadas mais adiante. Todas as soluções apresentam grau de complexidade considerável, bem como custo elevado.

Esse trabalho tem o objetivo de propor uma alternativa para a integração de sistemas heterogêneos, utilizando a tecnologia XML e XSLT, de modo a simplificar o desenvolvimento de programas com capacidade de integração com outros. Essa alternativa se constitui na utilização de um *framework* proposto, aliado a um pequeno módulo de software construído para ser utilizado como ferramenta de apoio.

Este documento está estruturado como segue.

No capítulo 2 é apresentado o problema da integração de sistemas heterogêneos, e as soluções mais comumente encontradas para sua resolução.

No capítulo 3, são apresentadas brevemente as tecnologias XML, XSLT e XPath, que são utilizadas na proposta de solução ao problema das integrações. São descritas também as duas API's utilizadas para o desenvolvimento utilizando XML, que são SAX e DOM.

No capitulo 4, é proposto um *framework* para a integração de sistemas, utilizando o binômio XML/XSLT. Neste capítulo é também apresentado o software construído como ferramenta de apoio ao *framework*.

O capítulo 5 apresenta as conclusões e trabalhos futuros.

2 Integração de sistemas heterogêneos

A popularização da informática, nos últimos anos, possibilitou o sugimento de uma quantidade considerável de empresas desenvolvedoras de software. O atual número de técnicas de desenvolvimento, aliado ao também elevado número de opções de plataformas computacionais, linguagens de programação e sistemas gerenciadores de bancos de dados (SGBD's), abriu uma gama de opções disponíveis para empresas que objetivam a informatização de seus processos.

Por motivos que fogem ao escopo desse trabalho explicar, é muito comum que empresas escolham, em seu processo de informatização, softwares de diferentes fornecedores, cada um com suas características próprias. A necessidade de integração entre esses diferentes sistemas surge então, como forma de automatizar processos, ou de fornecer informações consolidadas aos gestores das organizações.

É no momento que se começa a procurar alternativas para a realização da integração desses sistemas, que as dificuldades inerentes ao processo de integração começam a se apresentar.

Integrar sistemas significa, em primeira instância, fazer com que estes possam trocar informações entre si, de maneira a interoperar, visando atingir um objetivo comum. Como já foi dito, existe uma miríade de técnicas, linguagens e plataformas disponíveis para a confecção de sistemas. Essa miríade faz do processo de integração um problema com elevado número de variáveis, que dificultam a elaboração da solução do mesmo.

No Brasil, quando se trata de integrar sistemas através da utilização de softwares de terceiros, isto ainda é feito através de *middlewares*. *Middleware*, segundo [DIC 2001], é um software que funciona como uma camada de tradução e que está entre uma aplicação que reside num servidor e um grupo de clientes que querem acessar a essa aplicação. Nos Estados Unidos já é mais utilizado o conceito de troca de mensagens entre os aplicativos, através de softwares conhecidos como *message brokers*. [BAR 2000].

Nesse trabalho é abordada somente a integração de sistemas que acessam bancos de dados. Essas aplicações situam-se entre as mais amplamente utilizadas no mundo atual. Integração de sistemas não limita-se a esse tipo de sistema, pode-se, por exemplo, integrar sistemas de controle específicos para determinados componentes de hardware, ou esses controladores de hardware com sistemas que acessam bancos de dados. Enfim, embora outros sistemas, que não acessam bases de dados armazenados não sejam não sejam referenciados aqui, muito da problemática e a própria solução proposta nesse trabalho pode também ser aplicada na integração destes.

Devido ao alto custo de tecnologias de terceiros, tais como *middlewares* ou sistemas de mensagens inter-aplicativos, aliado a sua complexidade de configuração, é comum que empresas desenvolvam suas próprias soluções para integração de aplicativos, trocando dados entre esses.

O intercâmbio de informações entre programas aplicativos que acessam bancos de dados é realizado basicamente através de duas formas: através de arquivos de dados, geralmente texto, exportados por um dos softwares e importados por outro, ou através de acesso direto aos dados armazenados nos bancos.

Esta segunda opção se subdivide em dois modos: no primeiro caso, quando os dados estão armazenados no mesmo SGBD (Sistema Gerenciador de Banco de Dados), ambas as aplicações podem acessar as mesmas tabelas, sendo que para isso os desenvolvedores de cada uma devem conhecer as estruturas de dados utilizadas pela outra, ou ainda, uma estrutura comum deve ser definida como interface entre os aplicativos, dispensando assim, a necessidade de "divulgação" das estruturas de dados entre os envolvidos. No segundo caso, para intercâmbio de informações entre aplicativos acessando dados em diferentes SGBD's, pode ser utilizado um dos diversos *middlewares* disponíveis no mercado para a interconexão entre os SGBD's. Neste caso, os dados podem tanto ser replicados de um servidor para outro, como podem ser acessados localmente, pois há, entre os *middlewares* mais conhecidos no mercado, alguns que podem servir como *gateways*, permitindo que dados armazenados em diferentes bases sejam vistos como se estivessem residentes em mesmo local. Novamente apresenta-se então a necessidade de conhecimento da estrutura de dados ou a utilização de uma estrutura comum pré-acertada.

Segundo [SOA 95], *gateways* são computadores que interligam duas ou mais redes, de forma que os usuários as vejam como se fossem uma única rede. Expandindo-se esse conceito para a área de sistemas, tem-se que *gateways* são softwares capazes de fazer com que SGBD's distintos possam ser vistos com um único SGBD, obtendo transparência no acesso às informações residentes nos bancos de dados gerenciados por estes. A utilização de *gateways* implica na divulgação do modelo das estruturas de dados entre os desenvolvedores dos softwares a ser integrados.

Motivada pelos fatores expostos, a utilização de arquivos texto é a solução mais comum, estando presente em grande número de instituições que adquiriram softwares de diferentes fornecedores, e tiveram que integrá-los com os seus próprios ou com os de terceiros .

O ponto fraco dos arquivos texto reside no elevado grau de acoplamento entre os arquivos e os programas que os importam/exportam. Um programa capaz de importar arquivos de dados com a estrutura hipotética "A" deverá ser alterado para ser capaz de importar dados de uma arquivo com uma estrutura "B", diferente de "A".

A troca de dados através dos arquivos texto apresenta um outro problema: geralmente não só a estrutura dos dados sendo trocados é diferente, mas o domínio dos mesmos, bem como sua tipagem também é. Assim, um sistema pode ter seus *flags*, por exemplo, representados por uma variável booleana armazenando *true* ou *false*, um segundo sistema utiliza uma variável do tipo caractere com os valores "S" e "N" e ainda um terceiro sistema utilizando variáveis inteiras com valores 1 e 0.

Mesclando a questão das estruturas com a dos domínios, apresenta-se uma outra dificuldade na integração de sistemas: devido à diferenças de modelagem e de normalização entre estes, um mesmo conceito pode estar representado em um único

atributo em uma entidade de um sistema "A", e em mais de um atributo na entidade correspondente de um sistema "B". Isso leva à necessidade de mapeamentos não só estruturais, mas também de valores, para que a troca de informações entre os aplicativos seja possível.

E deve-se ainda considerar a diferença de técnica utilizada na representação de dados. Duas bases de dados , armazenadas em um mesmo tipo de SGBD, podem conter representações completamente diferentes para modelos do mundo real, na forma de seus diagramas de entidades e relacionamentos. (Em SGBD's diferentes essas diferenças acentuam-se, em função das peculiaridades dos mesmos) O mesmo conceito pode ter várias representações diferentes. Considerando-se também que um conceito pode estar representado utilizando a técnica da modelagem relacional, em um dos aplicativos, e em outro o mesmo conceito pode ter sido modelado através da técnica da modelagem orientada à objetos, vê-se que o processo de integração apresenta desafios consideráveis.

O surgimento do padrão XML representa um *"upgrade"* na utilização dos arquivos texto. Derivado do padrão SGML, o XML adiciona *tags* aos arquivos texto, podendo, desta forma, acrescentar semântica aos mesmos, ou seja, juntamente com os dados está informada a estrutura utilizada. Estando esta estrutura presente no arquivo texto, ela pode ser removida dos programas importadores/exportadores, permitindo o desenvolvimento de aplicativos genéricos.

Os dados podem ser representados em XML, e transformações XSL encarregam-se de realizar os mapeamentos de estruturas e valores, tirando essa responsabilidade dos programas que importam e exportam os dados. Com isso, obtém-se uma simplificação dos sistemas a ser integrados.

O próximo capítulo explicará como funcionam a tecnologia XML, bem como as tecnolgias que podem ser utilizadas, em conjunto a ela, na produção de aplicativos mais fáceis de integrar a outros.

3 XML utilizado no intercâmbio de dados

Nesse capítulo são apresentadas as tecnologias XML, XSL, XSLT e XPath, que podem ser utilizadas em conjunto na construção de sistemas com maior capacidade de integração, valendo-se dos benefícios que o padrão XML oferece. Além da apresentação dos ítens supracitados, são também apresentadas neste capítulo as API's utilizadas para construir sistemas baseados em XML, por considerar-se que o conhecimento destas apoia a compreensão do funcionamento do *framework* sugerido no capítulo 4.

3.1 XML

XML (eXtensible Markup Language) é um padrão proposto pela W3C (World Wide Web Consortium) utilizado para estruturar as informações na Web [W3C-2 98]. XML define a estrutura dessas informações através da tecnologia dos marcadores (*markups*), a mesma usada na linguagem HTML (HiperText Markup Language). Na realidade, tanto XML como HTML são derivados da mesma origem SGML (Standard Generalized Markup Language). Alguns autores afirmam que o HTML será substituído pelo XML, considerado como o HTML do futuro. A grande diferença entre as linguagens consiste na sua extensibilidade, ou seja, na flexibilidade quanto à definição dos marcadores. Enquanto o XML tem marcadores extensíveis (flexíveis), sendo que novos marcadores podem ser criados e utilizados de acordo com a necessidade do usuário, o HTML tem os mesmos fixos, como <P>, <TABLE>, <BODY>, etc., de forma que não se possam alterar e muito menos criar novos marcadores, além do que já foi definido na própria linguagem.

O código a seguir apresenta as diferenças entre ambos os padrões (HTML e XML), que utilizam a tecnologia dos marcadores. Pode-se observar que no padrão XML, novos marcadores ou *tags* foram criados para estruturar essas informações, dando semântica (significado) a elas e aos dados armazenados.

```
<!-Exemplo em HTML->
<hl>Catálogo de Automóveis</hl>
<p>Marca: FIAT
<p>Modelo: Palio EDX
<p>Ano:1999
<p>Cor:  Azul Santiago Metálico
<p>Preco: R$ 16.000

<!-Exemplo em XML-->
<Automóvel>
<Marca>FIAT</Marca>
<Modelo nome = 'Palio' categoria = 'EDX' />
<Ano>1999</Ano>
<Cor nome = 'Azul Marinho' pintura = 'Metálica'/>
<Preço moeda ='Real'>16.000</Preço>
</Automóvel>
```

Nota-se que novos marcadores foram criados no arquivo XML, como Automóvel, Marca, Modelo, Ano, etc., porém a utilização desses marcadores é regida

por regras e sintaxes que são validadas através do arquivo DTD (Document Type Definition). Em outras palavras, o arquivo DTD define o formato e a sintaxe de cada marcador que foi criado no arquivo XML. Cada arquivo XML poderá ter o seu próprio arquivo DTD, que definirá cada marcador criado anteriormente.

A seguir, é apresentado um exemplo de DTD para o arquivo XML codificado anteriormente:

```
<!- Exemplo de DTD para o arquivo XML ilustrado ->
<!ELEMENT Automóve1 (Marca, Modelo,Ano,Cor,Preço)>
<!ELEMENT Marca (#PCDATA)>
<!ATTLIST Modelo nome CDATA #REOUIRED>
<!ATTLIST Modelo categoria CDATA #REQUIRED>
<IELEMENT Ano (#PCDATA)>
<IATTLIST Cor nome CDATA #REQUIRED)>
<ATTLIST Cor pintura (Normal|Metálica|Perolizada)
"Metáica" #REOUIRED )>
<ATTLIST Preço moeda CDATA #REQUIRED>
```

Abaixo, são detalhadas as declarações do arquivo DTD acima:

- `<!ELEMENT Automóvel (Marca, Modelo, Ano, Cor, Preço)>` - O marcador Automóvel contém quatro subelementos nessa seqüência.
- `<!ELEMENT Marca (#PCDATA)>` - O elemento Marca é composto por uma cadeia de caracteres.
- `<!ATTLIST Modelo nome CDATA #REQUIRED>` - O elemento Modelo tem o atributo nome com formato caracter, exceto as *strings* "<", ">" e "&". Um valor deve ser fornecido.
- `<!ATTLIST Modelo categoria CDATA #REQUIRED)>` - O elemento Modelo tem o atributo categoria, que é urna cadeia de caracteres, exceto "<", ">" e "&". Um valor deve ser fornecido.
- `<!ELEMENT Ano (#PCDATA)>` - O elemento Ano tem o formato de uma cadeia de caracteres.
- `<!ATTLIST Cor nome CDATA #REQUIRED)>` - O elemento contém o atributo nome com o formato caracter, exceto "<", ">" e "&". O atributo é requerido.
- `<!ATTLIST Cor pintura (Normal | Metálica | Perolizada) "Metálica" #REQUIRED>` - O elemento Cor tem o atributo pintura, que pode ser "Normal", "Metálica" ou "Perolizada". O valor "Metálica" sempre deve ser fornecido, caso nenhum outro tenha sido definido.
- `<!ATTLIST Preço moeda CDATA #REQUIRED>` - O elemento Preço tem o atributo moeda, que é requerido e possui o formato de uma cadeia de caracteres, exceto "<", ">" e "&".

A parte introdutória deste texto, vista acima, foi retirada da revista Developer's Magazine [UEY 2000], e foi escolhida por mostrar claramente o principal ponto forte do XML, sua estrutura, bem como compará-lo ao HTML. A palavra chave é **semântica**. O XML separa, em sua sintaxe, os dados e sua estrutura. É esta separação que faz dele tão poderoso em relação ao HTML. Os *tags* do XML são definidos pelo criador do

documento, com base na estrutura definida no DTD. Com isto, a informação contida no documento XML passa a possuir um sentido, pois o DTD informa, de forma legível mesmo para humanos, o que representa a informação de um determinado campo. Ou seja, a estrutura dos dados é conhecida através do DTD, e sabemos portanto que a informação contida no *tag* `<Ano>1999</Ano>` representa o ano de um automóvel.

Nos documentos HTML, um *tag* informa apenas o formato de apresentação de um determinado dado, como a cor ou o tamanho da fonte que deve ser mostrada pelo browser. O XML concede significado às informações. Desta forma, pode-se, por exemplo, procurar por todos os documentos que contenham carros produzidos em 1999.

XML, por identificar que um determinado *tag* indica um ano de fabricação, pode facilmente ser varrido por um interpretador capaz de informar quais documentos atendem à requisição. Já no documento HTML, que não armazena a estrutura dos dados contidos, teria que ser varrido através de leitura exaustiva de seu conteúdo texto, e o máximo de informação que poderia ser encontrada é a *string* "1999" em seu interior, mas esta não necessariamente representaria o ano de fabricação do veículo. Em se tratando da concessão de um significado as informações texto nele armazenadas, o XML representa um marco tão grande na computação quanto a adoção do padrão ASCII para informar que o número binário 65 representa um "A" [BAL 2000].

O principal objetivo do XML é criar documentos inteligentes, que podem ser pesquisados de forma mais dinâmica que os convencionais, o que se obtém ao separar o conteúdo da apresentação dos documentos. Isto é conseguido porque os *tags* XML fornecem um meio de armazenar documentos estruturados em texto puro e renderizá-los em uma grande variedade de modos. Ou seja, a mesma informação é armazenada sempre em uma mesma forma, mas pode ser mostrada de diversas formas, o que pode ser realizado em diversos softwares, sistemas operacionais e plataformas de hardware diferentes. A formatação da apresentação dos dados fica por conta de uma tecnologia chamada XSL, que é apresentada mais adiante.

XML é uma das tecnologias computacionais de aplicabilidade maior do que a imaginada em sua concepção. Quando o XML Working Group, formado sob os auspícios do W3C em 1996, começou a desenvolver o XML, a idéia principal era a criação de um padrão mais potente que o HTML para a representação de dados para a Internet. O desacoplamento da definição dos dados e sua forma de apresentação possibilita que estes mesmos dados sejam utilizados em uma gama interminável de aplicações, que vão desde a utilização em Internet, para a qual o XML foi originalmente idealizado, até a conversão e integração de dados em sistemas legados.

Segundo o conceito da W3C [W3C-2 98], o XML descreve uma classe de objetos de dados chamados documentos XML, bem como descreve parcialmente o comportamento dos programas de computador que os processarão.

Esses programas são os processadores de XML. Um processador XML é um software capaz de entender um documento XML bem formado. Por sua vez um documento bem formado é aquele que segue a sintaxe do XML e se adequa a definição armazenada em um DTD. Simplificando: Um processador XML é capaz de ler um documento XML, seu DTD respectivo, e saber se o documento é um XML válido ou

não dentro de um contexto, e uma vez validado, pode ter seus dados utilizados por programas aplicativos. Uma vez que o XML não é uma especificação binária, seus processadores podem ser escritos em qualquer linguagem de programação, o que torna o padrão ainda mais flexível.

Apresentam-se então, como características básicas do XML [UEY 2000]:

- Simplicidade: O padrão XML é muito mais simples do que padrões binários para armazenamentos de dados. Dados armazenados em XML têm formato baseado em caracter, o que faz com que estes possam ser manipulados por qualquer editor de texto simples, bem como possam também ser lidos por seres humanos sem o uso de interpretadores. Além disso, sua especificação é bastante sucinta, resumindo-se à cerca de trinta páginas. Isto faz com que seja fácil aprender a utilizar XML. O número de características opcionais do padrão também é muito baixo.

- Extensibilidade: A separação dos dados e do formato de apresentação faz com que os dados possam ser escritos somente uma vez e gerados para leitura ou armazenamento em várias mídias e formatos diferentes. Pode-se por exemplo, formatar os dados e distribuí-los através de tecnologia WAP (*Wireless Application Protocol*) , para telefones celulares, ou gravá-los em um CD-ROM para consulta futura. Essa formatação é realizada através de XSL (*eXtensible Stylesheet Language*). Com o uso de XSL, pode-se apresentar dados contidos em um documento XML como uma tabela HTML, sem a necessidade de alterar o arquivo onde os dados se encontram.

- Descrição específica das informações e consultas mais dinâmicas: Uma vez que os *tags* do XML são definidos pelos projetistas dos documentos, o significado dos dados está armazenado junto com ele. Isso simplifica muito a busca de informações, uma vez que é necessário apenas, para realizar uma busca, localizar o *tag* desejado, e verificar o conteúdo da informação posicionada nele. Pode-se inclusive ordenar os dados pelos *tags*, fazendo com que as consultas sejam muito mais flexíveis.

- Interoperabilidade: Como XML é um padrão texto e não-binário, ele é capaz de circular sem problemas entre plataformas de hardware e sistemas operacionais completamente distintos, e de ser interpretado por programas de aplicação escritos em qualquer linguagem.

XML apresenta também, no entanto, alguns problemas. O maior deles, inerente ao XML, é [KIE 99] a falta de limitações de escopo dentro do qual um nome de *tag* deva ser único. Por exemplo, um *tag* pode identificar a localização de um cliente ou a localização de uma área geográfica. Esse problema está sendo endereçado através da especificação XML *Namespace*, que descreve como prefixos podem ser utilizados para identificar unívocamente um *tag* como pertencendo a um documento fonte XML em particular. Cada prefixo é suportado por metadados descritivos que dão a localização de cada conjunto de *tags* do documento fonte.

Essas características fizeram do XML muito mais que uma linguagem de marcadores. A capacidade de aglomerar sentido aos dados, e a possibilidade de

interoperabilidade tornaram o XML um padrão de intercâmbio de dados extremamente flexível, como veremos na próxima sessão.

3.2 Tecnologias de transformação do XML

Uma vez desacoplados os dados de sua apresentação, estes podem ser utilizados da forma que se julgar mais conveniente, o que abre caminho para aplicações que são apresentadas nesta seção. Na seção anterior os conceitos básicos do XML foram apresentados, e o foco em desacoplamento de estrutura e apresentação foi bastante insistente. Foi demonstrado como os dados são representados. Agora, antes que se possa discutir os usos do XML, deve-se explicar como é feita a etapa de apresentação das informações contidas nos documentos, que é o ponto onde entram o XSL, o XSLT e o XPath.

3.2.1 XSL

O XSL (*eXtensible Stylesheet Language*) é uma linguagem para expressar *Stylesheets* [W3C 2000]. A partir de uma classe de documentos ou arquivos XML arbitrariamente estruturados, *designers* utilizam um *Stylesheet* (*Lay-out*) XSL para expressar suas intenções sobre como o conteúdo estruturado deve ser apresentado, ou seja, como o documento deve ser formatado para sua apresentação na mídia desejada, seja esta um browser, um documento para impressão ou um computador portátil como um Palm Pilot.

Essa formatação é conseguida através de um processador de *stylesheets*. Um processador de *stylesheets* é um software que recebe um documento em XML e um *lay-out* XSL e produz a apresentação dos dados contidos no XML da forma intencionada pelo designer daquele *lay-out*. O processo de apresentação é composto de duas fases: a primeira, chamada *tree transformation* (transformação de árvore), que é composta da construção de uma árvore de resultado a partir da árvore fonte XML, e a segunda, chamada *formatting* (formatação), que é a interpretação dessa árvore resultado para produzir resultados ajustados à apresentação na tela em papel ou em qualquer outra mídia. O processo de formatação é realizado através de um formatador, que pode ser simplesmente um *engine* embutido no interior de um *browser*.

O modelo de processamento XSL é apenas conceitual, não havendo obrigatoriedade da separação das duas etapas citadas acima. Implementações são livres para processar o documento de qualquer forma desejada, desde que o resultado seja igual aquele que seria obtido usando-se o modelo de processamento de XSL conceitual.

Todo documento XML é organizado em forma de árvore. Assim sendo, todo o documento XML começa em uma entidade chamada raiz. Para exemplificar, suponha-se um livro. Este livro é organizado em capítulos, e seus capítulos em seções, que por sua vez dividem-se em parágrafos. Estruturando-se um livro em uma árvore obteríamos algo como o diagrama abaixo:

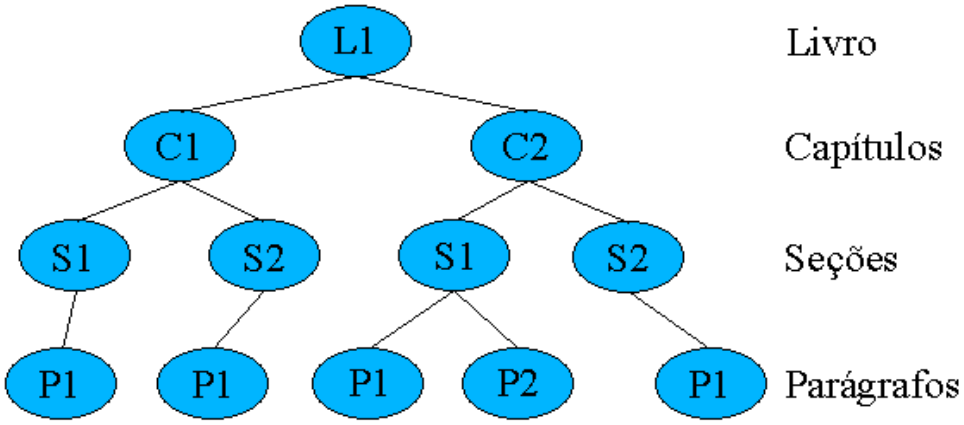

FIGURA 3.1 - Diagrama de um livro estruturado em árvore

O nodo L1 é o nodo *root* do documento XML. Só pode haver uma única entidade livro em um documento que representa um livro, porém essa entidade pode subdividir-se em diversos capítulos, parágrafos e seções. Essa explicação sobre a estrutura de um documento XML é necessária para a melhor compreensão de como o XSL funciona.

3.2.1.1 Transformação da árvore

Na fase da transformação de árvore, a árvore fonte contida no documento XML é lida e pode ou não ser transformada, segundo critérios do desenvolvedor do *stylesheet*. Assim sendo, o resultado apresentado é capaz de agregar mais semântica à informação armazenada. Por exemplo, tomando-se como exemplo o livro, não há neecessidade de armazenar-se o índice de nomes de capítulos, uma vez que se pode "filtrar" a árvore XML para se demonstrar apenas essa informação. Até aqui o que se obtém é uma árvore resultado, ainda sem a adição de qualquer caracter de formatação.

Durante a transformação da árvore XML, nodos, e até mesmo níveis podem ser acrescentados à arvore XML, alterando sua estrutura.

3.2.1.2 Formatação

O passo seguinte é a formatação da árvore resultado para sua apresentação, no dispositivo escolhido. A semântica de formatação é expressa em termos de um catálogo de classes de objetos de formatação. Os nodos da árvore resultado são objetos que serão formatados utiilizando essas classes, que representam abstrações tipográficas como páginas, parágrafos, tabelas, etc. Existe também uma classe de objetos chamados propriedades de formatação, para ajuste fino do formato, tais como controle de hifenação, tamanho e espaçamento de letras e outros desse gênero. Pode-se dizer que, de forma simplificada, a formatação é aplicação de uma máscara à informação armazenada. Esse processo é também chamado de renderização.

O processo de formatação se dá em três etapas.

Na primeira, cada elemento e atributo obtido da árvore resultado XML é convertido em um objeto via uma transformação XSLT. Esse processo é basicamente a

conversão de elementos da árvore em objetos nodos, e os atributos em especificações de propriedades. O resultado obtido aqui é a árvore de objetos a serem formatados.

Na segunda etapa, é realizada a refinação da árvore, para obter a árvore refinada de objetos a ser formatados. Nessa etapa são realizados os mapeamentos de propriedades para resultados. Aqui é realizada a expansão de propriedades individuais, mapeamento de propriedades correspondentes, determinação de valores computados e herança.

Na terceira e última etapa é realizada a geração de uma árvore de áreas geométricas, chamada de árvore de áreas. As áreas geométricas são posicionadas em uma seqüência de uma ou mais páginas. Cada área tem uma posição na página, uma especificação do que será apresentado em qual área e pode possuir fundo, preenchimento e bordas. Por exemplo, a formatação de um simples caracter deve gerar uma área suficientemente grande para conter o desenho do caracter, utilizado para representar o caracter visualmente, mais o caracter a ser apresentado. Estas áreas devem ser aninhadas. Por exemplo, o caracter pode ser posicionado em uma linha, dentro de um bloco, dentro de uma página. A partir desta etapa, as informações contidas no documento XML já estarão formatadas e prontas para a apresentação na mídia desejada.

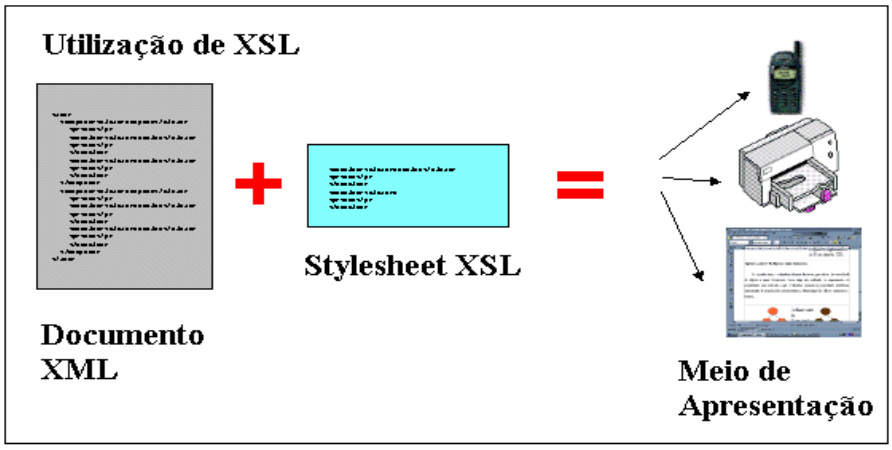

FIGURA 3.2 - Diferentes possibilidades de apresentação do mesmo documento XML

3.2.1.3 Benefícios do XSL

Diferentemente do HTML, o XML não possui semântica de formato de apresentação intrínseca ao documento, ou seja, em um documento HTML a informação sobre a apresentação das informações, tais como fonte, cor, tamanho e posição fica armazenada no próprio documento HTML, juntamente com as informações em si. No XML, a apresentação dos dados é de responsabilidade do XSL, armazenado em arquivo separado. É desta forma que os dados e sua forma se separam. Pode-se então, trocando-se os *stylesheets*, obter apresentações diferentes para um mesmo dado. Da mesma forma, o dado armazenado no XML pode ser utilizado sem necessariamente utilizar-se XSL. Porém, características bastante poderosas fazem deste uma ferramenta bastante útil [W3C 2000]:

- Paginação e *Scrolling*: A construção de documentos roláveis, que podem ser navegados em janelas ou páginas cria um grau de complexidade bastante grande para o conteúdo do XML. Controles como espaçamento e paginação tornam-se bastante importantes. Em HTML, todos esses controles são de tal forma interligados a informação ali armazenada que, abrindo-se o documento em um editor de textos simples, fica bastante difícil compreender o que está gravado ali. O XSL resolve esse problema tomando para si o controle do rolamento.

- Seletores e construção de árvores: XSL utiliza XSLT e XPath para a construção de suas árvores, o que fornece alto grau de flexibilidade e independência em relação aos dados armazenados. Pode-se apresentar os dados de qualquer maneira desejada. Uma vez trabalhando-se com árvores, pode-se fazer por exemplo, que todos os terceiros parágrafos de cada capítulo de um livro sejam mostrados em negrito. Já que o livro é estruturado em árvore como já mostrado na figura 1, basta navegar pelos nodos da árvore representando os parágrafos desejados e manipulá-los como quiser.

- Um modelo de *lay-out* de páginas extendido: As opções para formatação de texto do XSL decrevem tanto a estrutura de uma página ou *frame* (Cabeçalhos, rodapés, tamanhos) quanto as regras para colocar o conteúdo do XML nesses receptáculos.

- O modelo de áreas é compreensível: Os desenvolvedores do XSL tomaram o cuidado de fazer com que sua definição de áreas fosse simples, basendo-a em abstrações como blocos, linhas e regiões de página. O modelo de áreas fornece um vocabulário para descrever os relacionamentos e ajuste de espaçamento entre letras, palavras, linhas e blocos.

- Modos de escrita e internacionalização: XSL trabalha com *frames* relativos para a escrita. Isto por que culturas orientais não escrevem como nós, de cima para baixo e da esquerda para a direita. Uma vez que existem essas diferenças, para que um formato seja válido em todo o planeta, ele deve ser capaz de tratar qualquer tipo de escrita.

- *Links*: O XML não possui *links* embutidos, como HTML. Os *links* são definidos através de um objeto de formatação XSL, de dupla função, apresentar o conteúdo do documento XML e criar o apontador para o alvo a ser visitado.

3.2.1.4 Um exemplo do XSL

Levando-se em conta um documento XML representando um livro, um possível *lay-out* XSL seria:

Documento XML representando livro:

```
<doc>
  <chapter><title>Chapter</title>
    <p>Text</p>
    <section><title>Section</title>
    <p>Text</p>
    </section>
    <section><title>Section</title>
```

```
    <p>Text</p>
    </section>
  </chapter>
  <chapter><title>Chapter</title>
    <p>Text</p>
    <section><title>Section</title>
    <p>Text</p>
    </section>
    <section><title>Section</title>
    <p>Text</p>
    </section>
  </chapter>
</doc>
```

Note-se que o livro é um documento dividido em capítulos e seções, sendo que para uma dessas divisões existe um título.

XSL *Stylesheet*:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format"
                version='1.0'>

<xsl:template match="chapter">
  <fo:block break-before="page">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="chapter/title">
  <fo:block text-align="center" space-after="8pt"
            space-before="16pt" space-after.precedence="3">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="section/title">
  <fo:block text-align="center" space-after="6pt"
            space-before="12pt" space-before.precedence="0"
            space-after.precedence="3">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="p[1]" priority="1">
  <fo:block text-indent="0pc" space-after="7pt"
            space-before.minimum="6pt" space-before.optimum="8pt"
            space-before.maximum="10pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="p">
  <fo:block text-indent="2pc" space-after="7pt"
            space-before.minimum="6pt" space-before.optimum="8pt"
            space-before.maximum="10pt">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

```
</xsl:stylesheet>
```

O que se pode ver no exemplo é como o XSL funciona. Procura-se por um *template* `<xsl:template match="chapter/title">`, representando um título de capítulo e, um vez encontrando-o, aplica-se a informação contida no *tag* os comandos abaixo, que centralizam o texto do título e ajustam os espaços antes e depois deste.

```
<fo:block text-align="center" space-after="8pt"
          space-before="16pt" space-after.precedence="3">
  <xsl:apply-templates/>
</fo:block>
```

No exemplo fica clara a separação entre elementos estilísticos de apresentação do documento (XSL) e a estrutura dos dados (XML).

3.2.2 XSLT

O XSLT (XSL Transformation) é uma linguagem para transformar documentos XML em outros documentos XML [W3C-2 99]. Embora possa ser utilizada independentemente do XSL, o XSLT foi desenvolvido para trabalhar em conjunto com o XSL, realizando as transformações nos documentos XML para sua apresentação. Uma transformação XSLT está representada em um documento XML bem formado, que pode tanto conter elementos XSLT quanto não-XSLT.

Uma transformação descrita em XSLT descreve as regras para transformação de uma árvore fonte em uma árvore resultado XML. Esta transformação é obtida através da associação de padrões com *templates*. Um padrão é comparado com elementos na árvore fonte, e um *template* é instanciado para a criação de parte da árvore resultado. As duas árvores são separadas e podem possuir estruturas completamente diferentes. Na construção da árvore resultado, elementos da árvore fonte podem ser filtrados e reordenados, bem como novos elementos arbitrários podem ser inseridos.

Uma transformação expressa em XSLT é chamada de *stylesheet*. Isto se deve ao fato de que, quando XSLT é utilizada em conjunto com XSL, a transformação ocorrida faz as vezes de um *stylesheet*. Um *stylesheet* contém um conjunto de regras de *templates*. Uma regra de *template* tem duas partes: Um padrão que deve ser comparado com os nodos da árvore fonte e um *template* que deve ser instanciado para formar parte da árvore resultado. Isto permite ao *stylesheet* ser aplicável a uma grande classe de documentos que possuam estruturas de árvores similares.

Um *template* é instanciado para um elemento fonte em particular para criar parte da árvore resultado. Um *template* pode conter elementos que especificam estrutura de elementos resultados literais. Um *template* pode também conter elementos do *namespace* XSLT que são instruções para criar fragmentos da árvore resultado. Quando um *template* é instanciado, cada instrução é executada e substituída por um fragmento da árvore resultado que ele criou. Instruções podem selecionar e processar elementos fonte descendentes. Este processamento resulta na criação de um fragmento na árvore resultado através do encontro de uma regra de *template* e sua posterior instanciação.

Note que os elementos somente são processados quando são selecionados pela execução de uma instrução. A árvore resultado é construída encontrando-se a regra de *template* para o nodo *root* e instanciando-se o seu *template*.

3.2.2.1 Exemplo de XSLT

O documento XML a seguir representa uma tabela periódica com apenas dois elementos químicos, hidrogênio e hélio [HAR 99].

Documento XML:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="14-2.xsl"?>
<PERIODIC_TABLE>

  <ATOM STATE="GAS">
    <NAME>Hydrogen</NAME>
    <SYMBOL>H</SYMBOL>
    <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
      0.0899
    </DENSITY>
  </ATOM>

  <ATOM STATE="GAS">
    <NAME>Helium</NAME>
    <SYMBOL>He</SYMBOL>
    <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
    <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
    <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
    <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
    <DENSITY UNITS="grams/cubic centimeter"><!-- At 300K -->
      0.1785
    </DENSITY>
  </ATOM>

</PERIODIC_TABLE>
```

O que veremos a seguir é a transformação deste documento XML em um documento HTML, através de XSLT. Primeiro é mostrado o código XSLT, e depois o documento HTML resultante [HAR 99].

Código XSLT:

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

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

  <xsl:template match="ATOM">
```

```
    <P>
      <xsl:apply-templates/>
    </P>
  </xsl:template>

</xsl:transform>
```

Temos aqui código XSLT que, quando processado, procura pela ocorrência da *string* "PERIODIC_TABLE", e encontrando-a, ativa um *template* para inserir no documento gerado os marcadores <html> e </html>. De forma semelhante, outro *template* é acionado pelo encontro do *string* "ATOM", inserindo para as linhas aninhadas por este *tag* os marcadores <P> e </P>.

Documento HTML resultado:

```
<html>
  <P>
    Hydrogen
    H
    1
    1.00794
    20.28
    13.81
      0.0899
  </P>
  <P>
    Helium
    He
    2
    4.0026
    4.216
    0.95
      0.1785
  </P>
</html>
```

3.2.3 XPath

XSLT define quais alterações serão realizadas sobre um elemento, quando ele for encontrado. Encontrar tal elemento, diferenciando-o de outro qualquer em um documento XML é responsabilidade do XPath. XPath é a especificação de uma linguagem capaz de endereçar elementos, reconhecendo-os dentro de uma estrutura XML [W3C-1 99].

A especificação XPath é a base para uma série de outras especificações, como XSLT e XPointer [SUN 2001]. Os *templates* desenvolvidos em XSLT utilizam padrões (*patterns*) para encontrar elementos no documento XML e aplicar as transformações definidas. É através do XPath que distingue-se, por exemplo, um elemento <rua><numero> de um elemento <apto><numero>. Uma vez interpretada a árvore respresentada em um documento XML, o XPath consegue encontrar um elemento <numero> "filho" de um elemento <rua>, e saber que este não é o elemento <numero> "filho" do elemento <apto>.

O padrão XPath não se refere somente a elementos. A especificação define um documento abstrato com sete tipos diferentes de nodos:

- *root*
- *element*
- *text*
- *attribute*
- *comment*
- *processing instruction*
- *namespace*

O funcionamento do XPath baseia-se na navegação sobre a estrutura em árvore de um documento XML. Esta estrutura é hierárquica, como uma árvore de diretórios. Uma forma análoga ao *path* de uma árvore de diretórios é utilizada para especificar qual elemento da árvore está sendo procurado. As semelhanças são muitas:

- A barra (/) é utilizada como separador de *path's*.
- Um caminho absoluto a partir do nodo *root* começa com /.
- Um caminho relativo a partir de um nodo começa com qualquer outro nome.
- Ponto ponto (..) indica o nodo "pai" de um dado nodo.
- Um ponto (.) indica o nodo corrente.

Seguindo o padrão de diretórios de sistemas UNIX, a nomenclatura de elementos é sensível a letras maiúsculas e minúsculas, logo, um elemento `<teste>` não será encontrado por uma expressão XPath procurando pelo padrão `<Teste>`.

Quando se realiza uma busca por batimento de padrões (*pattern matching*) em XSLT, a especificação /h1/h2 seleciona todos os elemetos h2 que residem sob um elemento h1. Para especificar um determinado elemento h2, utilizam-se colchetes ([ ]) para indexar o elemento desejado, da mesma forma como a maioria das linguagens de programação representam vetores. Dessa forma, para identificar-se o terceiro elemento h2, sob o segundo h1, a sintaxe é `/h1[2]/h2[3]`.

A especificação XPath refere-se sempre a um elemento XML. Para representar um atributo, deve-se preceder seu nome com uma arroba (@). Por exemplo, a expressão `pessoa/@sexo` refere-se ao atributo `sexo` do elemento `pessoa`.

XPath também utiliza coringas (*wildcards*), para ampliar o escopo da busca. Por exemplo, o caracter asterisco (*), indica a busca por qualquer elemento, e arroba asterico (@*), por qualquer atributo. Voltando ao exemplo `<numero>`, a expressão `*/numero` identifica todos os números, indepententemente destes serem filhos de `<rua>` ou de `<apto>`.

Expressões XPath retornam sempre um conjunto de nodos, um string, um valor booleano ou um número. XPath conta também com um conjunto de operadores e funções que permitem a manipulação destes resultados. Essas funções dividem-se em funções posicionais, de manipulação de strings, de números, booleanas, de conversão e funções de *namespaces*. O número de funções é alto e explaná-las todas não faz parte do

escopo deste trabalho. Informações mais detalhadas podem ser encontradas na seção 4 da especificação XPath do W3C.

Para exemplificar, apresenta-se a seguir um exemplo de *template* XSLT:

```
<xsl:template match="/Funcionarios/Funcionario/Admissao">
<DataEntrada><xsl:value-of select="../Admissao"/></DataEntrada>
</xsl:template>
```

A seqüência XPath `"/Funcionarios/Funcionario/Admissao"` refere-se ao elemento sendo procurado, que é o elemento `<Funcionarios> <Funcionario> <Admissao>` , este *template* XSLT define que, quando tal elemento for encontrado, o conteúdo do elemento deve ser expresso no documento XML resultante, sob o *tag* `<DataEntrada>`.

3.3 Outras possibilidades de utilização do XML

Não é somente no intercâmbio de informação pura que o XML tem mostrado grande utilidade. O advento da Internet fez com que os sistemas computacionais adotassem estruturas diferentes. Assim, da programação estruturada chegamos à programação orientada à objetos, e esta evolui fortemente em direção ao desenvolvimento de aplicações com objetos distribuídos. Isto porque a Internet baseia-se na utilização conjunta de diversos hardwares, e trabalhando-se com sistemas distribuídos pode-se aproveitar ao máximo as potencialidades de cada um destes.

Os dois principais modelos de computação utilizando objetos distribuídos atualmente, com capacidade de interoperar com diversas das linguagens de programação, dentre as mais utilizadas comercialmente, são o **Microsoft DCOM** e o **OMG CORBA**. Apesar da grande similaridade funcional entre estes, pois ambos operam trabalham e fornecem serviços de comunicação ao estilo cliente-servidor [CHU 97], - ao requisitar um serviço, um cliente invoca um método implementado por um objeto remoto, que age como o servidor no modelo cliente-servidor - estes modelos não conversam entre si, pelo fato de suas interfaces serem diferentes. O XML fornece um meio rápido e conveniente de permitir a sistemas como esse interagirem um com o outro, através de seu padrão [GAR 99] . O OMG (Object Management Group) está trabalhando na maneira de realizar a inserção do XML em seus *Brokers*.

Outro trabalho que a OMG está desenvolvendo, neste caso "alavancada" diretamente pela IBM e Unisys é a criação de um novo padrão aberto que combine os benefícios do XML com os benefícios da UML (Unified Modeling Language), que é uma especificação do OMG que fornece aos desenvolvedores uma linguagem comum para especificação, visualização, construção e documentação de objetos distribuídos e modelos de negócios [IBM 2000]. Este padrão é o XMI (XML Metadata Interchange).

O XMI especifica um padrão aberto de intercâmbio de informação que objetiva dar aos desenvolvedores, trabalhando com tecnologia de objetos, a habilidade de trocar dados de programação através da Internet de maneira padronizada. Trazendo consistência e compatibilidade à aplicações criadas em ambientes de colaboração.

Através do estabelecimento de um padrão de mercado para armazenar e compartilhar informação sobre objetos, as equipes de desenvolvimento utilizando ferramentas variadas, de fornecedores diversos podem colaborar entre si no desenvolvimento de aplicações. O padrão proposto permitirá aos programadores utilizar a Web para trocar dados entre ferramentas, aplicações e repositórios e criar aplicações distribuídas seguras construídas em um ambiente de equipes de desenvolvimento.

Resumindo, então: as possibilidades de utilização do XML são inúmeras. A possibilidade de armazenar dados estruturados em um meio legível em diversas plataformas diferentes o promove a uma condição de *middleware* universal, pois ele pode portar dados, projetos de telas, modelos de projetos e afins. A consolidação deste como padrão de intercâmbio é questão de tempo.

3.4 API's para XML

O próximo passo deste estudo é uma breve introdução as duas API's mais usadas no trânsito dos dados entre o XML e os programas aplicativos. Estas API's (Application Program Interfaces - Interfaces de Programação de Aplicações), dividem-se em duas categorias [LAD 2000], as API's orientadas a árvores, na qual enquadra-se o DOM, e API's orientadas a eventos, como o SAX. API's orientadas a árvores compilam um documento XML em uma estrutura interna de árvore, permitindo então que programas de aplicação naveguem essa árvore. Já as API's orientadas a eventos reportam eventos da análise do documento, como por exemplo o encontro do início ou fim de um documento, diretamente para a aplicação através de chamadas, e normalmente não constróem árvores internas. A aplicação manipula essas chamadas, de forma similar ao disparo de eventos em aplicações baseadas em interfaces gráficas para usuários.

3.4.1 SAX

SAX (Simple API for XML), como já mencionado, é uma API baseada em eventos [MEG 2000]. SAX é implementada em Java, e deve funcionar com qualquer SGBD para o qual exista acesso JDBC. SAX, como qualquer interface baseada em eventos, analisa o documento XML e envia chamadas para ser processadas pela aplicação. Sua principal vantagem é o fato de, por não construir uma árvore interna, necessitar de muito menos recurso computacional para trabalhar, especialmente se o documento XML for muito grande. Além disso, algumas aplicações necessitam construir suas próprias árvores internas, e seria ineficiente construir uma árvore, através de um parser, para depois convertê-la em outra, com estrutura diferente. As vantagens de SAX ficam evidentes quando se analisa o seguinte caso: Se o objetivo da análise de um documento XML grande, com digamos, 20 Mb, for a busca do *string* "UFRGS", a menor árvore gerada para a análise deste documento teria no mínimo 30 Mb. Utilizando-se SAX, o documento é lido seqüencialmente, sem a formação de uma árvore, simplesmente procurando-se por ocorrências da *string* informada. As partes já lidas do documentos podem ser descartadas, e a memória alocada a elas, liberada.

A seguir é demonstrado um exemplo de um pequeno documento XML e o resultado de sua análise por SAX:

```
<?xml version="1.0"?>
<doc>
<para>Hello, World!</para>
</doc>
```

Análise por SAX:

```
start document
start element: doc
start element: para
characters: Hello, World!
end  element: para
end element: doc
end document
```

Assim sendo, uma aplicação gerenciaria os eventos encontrados da mesma forma como manipula eventos de interface, sem a necessidade de armazenar o documento inteiro em memória. É importante salientar que, se desejado, pode-se utilizar os eventos reportados por SAX para construir uma árvore, e analisar o documento como a outra interface que iremos ver, o DOM.

3.4.2 DOM

Ao contrário de SAX, que teve seu desenvolvimento realizado de forma independente, DOM (Document Object Model) é uma especificação do W3C [W3C-1 98], e é uma API que pode ser usada tanto para documentos HTML como XML. DOM define a estrutura lógica dos documentos e o modo como o documento é acessado e manipulado. Na especificação DOM, o termo documento é utilizado em um amplo sentido, e tudo, desde o XML até diferentes tipos de informação armazenada são vistos como documentos. XML apresenta seus dados como documentos, e DOM pode ser utilizado para manipular estes dados.

A interface DOM constitui-se de um *parser* que monta árvores a partir de estruturas de documentos. Uma vez de posse da árvore, programadores podem construir documentos, navegar na estrutura, adicionar, modificar e remover elementos e conteúdo. Qualquer informação encontrada em um documento XML ou HTML pode ser manipulado através da utilização de DOM. Vejamos o exemplo de um documento em XML e a árvore gerada por DOM para este:

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

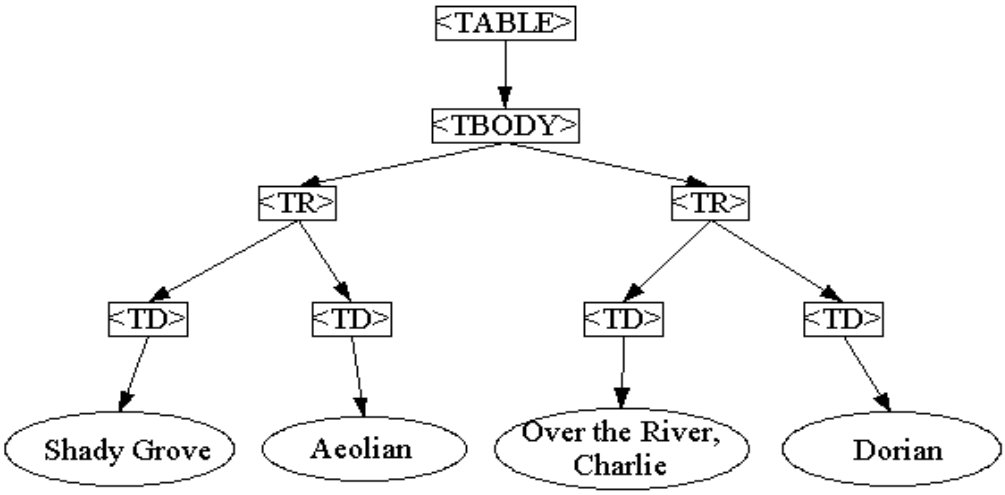

FIGURA 3.3 - Árvore formada por DOM para um documento XML [W3C-1 98]

A estrutura dos documentos em DOM é uma floresta de árvores, pois pode conter mais do que uma árvore. DOM não especifica que documentos precisem ser implementados como árvores ou florestas, nem especifica as relações entre os objetos a ser implementados. DOM é um modelo lógico que pode ser implementado da maneira que se julgar mais conveniente.

Em sua especificação, o termo modelo de estrutura é utilizado para descrever a representação em forma de árvore utilizada. DOM possui isomorfismo estrutural, o que significa que não importa a implementação utilizada, a representação gerada para um mesmo documento deve ser igual. DOM, como modelo lógico, representa a estrutura e o comportamento dos objetos dos quais um documento é composto. Ele identifica:

- As interfaces entre os objetos utilizados para representar e manipular um objeto.
- A semântica dessas interfaces e objetos, incluindo comportamento e atributos.
- Os relacionamentos e colaborações entre estas interfaces e objetos.

Documentos SGML, e em nosso caso específico XML, são representados através de um modelo de objetos abstrato (existe um padrão para o modelo, mas não uma especificação binária deste) , e não por um modelo de objetos. Em um modelo abstrato, o modelo é centrado nos dados. Em linguagens de programação orientadas a objetos, o dado em si é encapsulado em objetos que escondem os dados, protejendo-os de manipulação externa direta. As funções associadas com esses objetos determinam como os objetos podem ser manipulados, e como eles fazem parte do modelo de objetos.

Por se tratar de um modelo abstrato, DOM pode ser implementado em qualquer linguagem de programação orientada à objetos.

4 *Framework* para integração de aplicações heterogêneas

O problema de integrar sistemas, com suas dificuldades intrínsecas, foi exposto no capítulo 2. No capítulo 3, foram apresentadas a tecnologia XML e as tecnologias auxiliares que operam em conjunto a ela, para conferir-lhe a flexibilidade e poder necessários para sua utilização na implementação de softwares integradores.

Este capítulo apresentará a sugestão de uma alternativa para a integração de sistemas heterogêneos entre si. Essa alternativa se apresenta na forma de um *framework* que faz uso de tecnologia XML para a comunicação de dados entre os aplicativos, de maneira que as adequações necessárias aos dados sejam externas aos aplicativos, sendo realizadas através de transformações executadas por um processador XSLT.

Segundo Deutsch, citado por [GAM 2000], um *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma específica classe de software.

Já Fayad, em [FAY 99], dá duas outras definições para *framework*: "Um *framework* é um design reutilizável do todo ou de parte de um sistema, que é representado por um conjunto de classes abstratas e o modo como suas instâncias interagem" e "um *framework* é um esqueleto de uma aplicação que pode ser customizada por um desenvolvedor de aplicação". Ambas as definições não são opostas. São, na verdade, complementares.

O objetivo desse trabalho é propor esse "esqueleto" de aplicação, para ser utilizado por desenvolvedores de software em seus projetos de integração. A seguir é demonstrada a arquitetura do *framework* proposto.

4.1 Arquitetura geral do *framework*

Possuindo-se como ponto de partida dois softwares quaisquer, ambos com a capacidade de gerar e interpretar documentos XML, pode-se esquematizar um *framework* onde ambos os softwares possam passar e receber dados um do/para o outro. A figura 4.1 ilustra a troca de documentos XML entre os aplicativos.

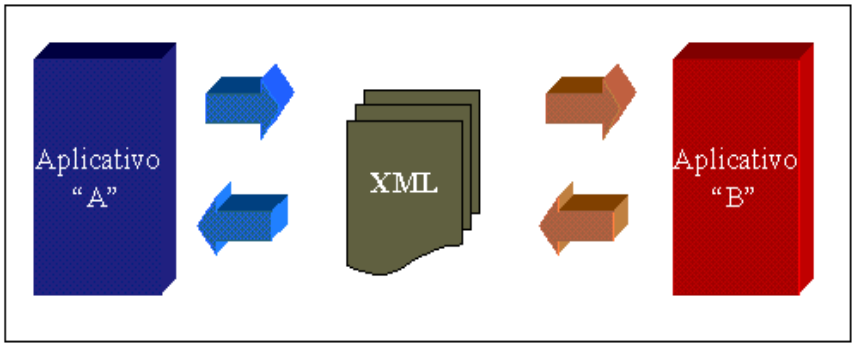

FIGURA 4.1 - Aplicativos trocando informações através de XML

Ambos os softwares são capazes de interpretar documentos XML. XML é padrão único. Porém, cada programa está preparado para reconhecer um número limitado de documentos XML. Por exemplo, uma aplicação bancária, mesmo portadora de um *parser* XML, não estará apta para manipular um documento XML contendo informações a respeito do resultado de um exame de sangue, gerado por uma aplicação laboratorial. Embora o XML defina a forma, o conteúdo é definido pelo criador do documento, e se o significado de cada informação contida no documento não for conhecida por ambas as partes envolvidas no processo, estas continuarão sem "conversar".

Uma vez que ambas as aplicações conhecem a semântica das informações contidas nos documentos, a necessidade de codificá-los em XML poderia tornar-se discutível, pois a interpretação XML dos dados representaria apenas um passo a mais de processamento para os aplicativos. O fato é que existem vantagens no uso de XML, e essas vantagens são discutidas a seguir.

Não utilizar XML para realizar esta integração faz com que seja necessário codificar nos programas o formato dos arquivos, e qualquer alteração nestes formatos força a necessidade de manutenção dos aplicativos. Além disso, antes do começo do desenvolvimento dos programas, as partes interessadas deverão entrar em acordo em relação aos conteúdos que serão enviados e recebidos em ambas as aplicações, desenvolvendo uma espécie de protocolo.

Suponha-se agora a entrada de um terceiro programa a ser integrado nesta mesma conjuntura. O protocolo já está combinado. Seria necessário, para integrá-lo, a alteração do protocolo, e portanto dos outros dois programas já existentes. E se um deles não puder ser alterado, por tratar-se de um padrão imposto por algum órgão público, ou se for necessária a introdução de um quarto ou quinto programa no conjunto, pode-se ter em mãos um problema de difícil resolução.

A codificação das estruturas de dados nos programas faz com que essa estrutura torne-se muito rígida , e pouco passível de alteração. Faz-se necessária uma abordagem com maior flexibilidade, capaz de suportar todas as mudanças que podem vir a surgir nesse cenário.

Dentro deste contexto é que a utilização do XML e das suas tecnologias de transformação têm o poder para agir como *middlewares* universais. Explica-se: Se cada aplicativo for capaz de interpretar XML, basta definir quais documentos cada um será capaz de ler e gerar, fazendo destes caixas-pretas, como funcões em linguagens de programação. Define-se a interface de contato destes aplicativos através de documentos XML. Isto elimina a necessidade de alteração dos mesmos para cada nova estrutura XML utilizada.

Persiste ainda a necessidade de mapeamentos entre diferentes estruturas XML utilizadas por diferentes programas. Esta necessidade é facilmente endereçada através da utilização de XSLT para realizar as conversões necessárias. Desta forma, a necessidade de alteração nos programas é reduzida drasticamente, pois estes deixam de ser adequados a novos documentos XML. Pelo contrário, agora, através de XSLT, os documentos é que são adequados a estes.

Existe, no entanto a necessidade do desenvolvimento de *stylesheets* XSL para a realização das conversão de documentos. Para tanto, foi desenvolvida uma ferramenta de apoio para criação de *stylesheets* para realizar as modificações dos documentos envolvidos, através da transformação de suas árvores DOM.

Fica definido portanto, o *framework* de integração, conforme pode ser visualizado na figura 4.2.

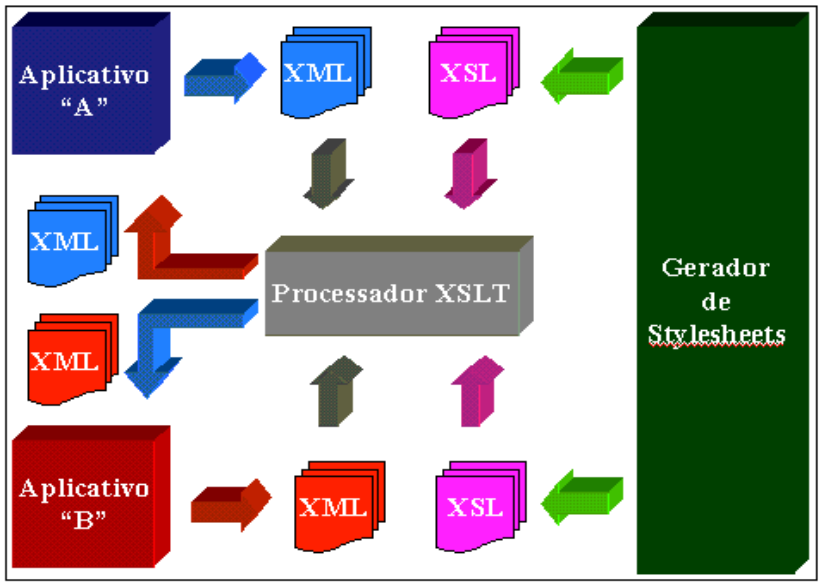

FIGURA 4.2 - O *framework* proposto para integração dos aplicativos

Os programas envolvidos geram e recebem documentos XML de formatos fixos, pré-definidos e conhecidos pelos participantes. Uma ferramenta de apoio gera as *stylesheets* que serão utilizadas para conversão entre os formatos, e as conversões são realizadas por todos os participantes antes da leitura dos documentos XML, utilizando um processador XSLT.

4.2 Exemplos da utilização do *framework*

Com a finalidade de demonstrar como o *framework* de integração é utilizado, são apresentados dois exemplos, em forma de códigos XML e XSLT.

4.2.1 Troca de informações com diferenças de estruturas

No primeiro exemplo, suponha-se que duas aplicações tenham que trocar informações a respeito de seus cadastros de funcionários. Vamos chamá-las simplesmente de aplicações "alfa" e "bravo". Ambas aplicações têm o conceito funcionário representado em forma de entidades bastante semelhantes, com variações somente na nomenclatura das entidades e atributos. Ambas podem gerar e reconhecer documentos XML, cada uma com seu DTD próprio. Exemplos dos documentos XML de ambas são mostrados a seguir.

Documento XML da aplicação "alfa":

```
<?xml version="1.0"?>
<Funcionarios>
     <Funcionario>
          <Codigo>35</Codigo>
          <Nome>Maria Helena Andrade</Nome>
          <Setor>Financeiro</Setor>
     </Funcionario>
</Funcionarios>
```

Documento XML da aplicação "bravo":

```
<?xml version="1.0"?>
<Empregados>
     <Emp>
          <cod_emp>567</cod_emp>
          <nome_emp>Jonas Carvalho</nome_emp>
          <depto_emp>Almoxarifado</depto_emp>
     </Emp>
</Empregados>
```

Como pode-se observar, as estruturas são idênticas, excluindo-se questões de nomenclatura. Porém, a aplicação "alfa" não é capaz de reconhecer os documentos da aplicação "bravo", devido à diferença dos *tags* utilizados por esta. O uso do *framework* proposto permite a integração das duas aplicações sem alteração dos códigos de nenhuma destas. Esse objetivo é alcançado através da aplicação de transformações XSL (XSLT) aos documentos gerados por cada uma das aplicações.

Se ao documento da aplicação "alfa" for aplicada a transformação XSL abaixo, o resultado será um documento reconhecível pela aplicação "bravo":

Transformação XSL de "alfa" para "bravo":

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

```
<xsl:output method="xml" indent="yes"/>
   <xsl:template match="/Funcionarios">
      <Empregados>
      <xsl:apply-templates />
      </Empregados>
   </xsl:template>
   <xsl:template match="/Funcionarios/Funcionario">
      <Emp>
      <xsl:apply-templates />
      </Emp>
   </xsl:template>
   <xsl:template match="/Funcionarios/Funcionario/Codigo">
      <cod_emp><xsl:value-of select="../Codigo"/></cod_emp>
   </xsl:template>
   <xsl:template match="/Funcionarios/Funcionario/Nome">
      <nome_emp><xsl:value-of select="../Nome"/></nome_emp>
   </xsl:template>
   <xsl:template match="/Funcionarios/Funcionario/Setor">
      <depto_emp><xsl:value-of select="../Setor"/></depto_emp>
   </xsl:template>
<xsl:template match="text()">
</xsl:template>
</xsl:stylesheet>
```

Aplicada a transformação anterior ao documento "alfa", obtemos como resultado o seguinte documento:

Documento XML gerado pela aplicação da transformação de "alfa" para "bravo":

```
<?xml version="1.0" encoding="UTF-8"?>
<Empregados>
     <Emp>
          <cod_emp>35</cod_emp>
          <nome_emp>Maria Helena Andrade</nome_emp>
          <depto_emp>Financeiro</depto_emp>
     </Emp>
</Empregados>
```

O documento acima contém os dados gerados pela aplicação "alfa", no padrão que a aplicação "bravo" é capaz de reconhecer. Note-se aqui a grande vantagem da utilização do *framework* de integração: não houve nenhuma alteração no código interno dos programas envolvidos. As adequações necessárias aos dados sendo trocadas foram realizadas externamente aos programas, por um processador XSLT.

A transformação dos dados da aplicação "bravo" para "alfa" é obtida da mesma forma. A única alteração é o *stylesheet* utilizado, que é demonstrado a seguir:

Transformação XSL de "bravo" para "alfa":

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

```
      <xsl:apply-templates />
      </Funcionarios>
   </xsl:template>
   <xsl:template match="/Empregados/Emp">
      <Funcionario>
      <xsl:apply-templates />
      </Funcionario>
   </xsl:template>
   <xsl:template match="/Empregados/Emp/cod_emp">
      <Codigo><xsl:value-of select="../cod_emp"/></Codigo>
   </xsl:template>
   <xsl:template match="/Empregados/Emp/nome_emp">
      <Nome><xsl:value-of select="../nome_emp"/></Nome>
   </xsl:template>
   <xsl:template match="/Empregados/Emp/depto_emp">
      <Setor><xsl:value-of select="../depto_emp"/></Setor>
   </xsl:template>
<xsl:template match="text()">
</xsl:template>
</xsl:stylesheet>
```

Este primeiro exemplo demonstra a transformação de documentos XML com informações muito semelhantes, representadas em estruturas também bastante semelhantes. O próximo exemplo demonstrará um caso de uso onde o domínio dos dados representados varia, bem como a estrutura destes, e como tal caso pode ser tratado com a utilização de XSLT.

4.2.2 Troca de informações com mapeamento dos dados

Neste exemplo, são utilizadas duas outras aplicações hipotéticas, "charlie" e "delta". A peculiaridade apresentada na integração entre essas duas aplicações diz respeito a uma informação manipulada por estas. Ambas as aplicações, da mesma forma que as do exemplo anterior , geram e reconhecem documentos XML, cada uma com seu próprio DTD. Ambas as aplicações possuem também cadastros de clientes, que desejam trocar entre si. Porém, há um atributo em ambas as entidades que recebeu um tratamento diferente durante a etapa de modelagem. Esse atributo é o sexo do cliente, que para a aplicação "charlie" é expresso em termos da letra "M", para representar sexo masculino, e "F", para feminino. A aplicação "delta" representa o sexo masculino com o número inteiro 1, e o feminino como 2.

Portanto, para a integração destas aplicações, além do mapeamento de suas estruturas XML, é necessário também o mapeamento dos valores sexo, onde "M" é igual a 1, e "F" é igual a 2. Os documentos utilizados no exemplo são mostrados a seguir:

Documento XML da aplicação "charlie":

```
<?xml version="1.0"?>
<Clientes>
  <Cliente>
     <Codigo>78</Codigo>
     <Nome>Antonio Silva</Nome>
```

```
      <Sexo>M</Sexo>
   </Cliente>
   <Cliente>
      <Codigo>79</Codigo>
      <Nome>Luiza Pereira</Nome>
      <Sexo>F</Sexo>
   </Cliente>
</Clientes>
```

Documento XML da aplicação "delta":

```
<?xml version="1.0"?>
<CadCliente>
   <cliente>
      <cod_cli>95</cod_cli>
      <nome_cli>Ana Sampaio</nome_cli>
      <sexo_cli>2</sexo_cli>
   </cliente>
   <cliente>
      <cod_cli>96</cod_cli>
      <nome_cli>Marcelo Almeida</nome_cli>
      <sexo_cli>1</sexo_cli>
   </cliente>
</CadCliente>
```

Para que os dados gerados por "charlie" sejam reconhecidos por "delta", o documento XML deve ser submetido ao XSLT que segue:

Transformação XSL de "charlie" para "delta":

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
   <xsl:template match="/Clientes">
      <CadCliente>
      <xsl:apply-templates />
      </CadCliente>
   </xsl:template>
   <xsl:template match="/Clientes/Cliente">
      <cliente>
      <xsl:apply-templates />
      </cliente>
   </xsl:template>
   <xsl:template match="/Clientes/Cliente/Codigo">
      <cod_cli><xsl:value-of select="../Codigo"/></cod_cli>
   </xsl:template>
   <xsl:template match="/Clientes/Cliente/Nome">
      <nome_cli><xsl:value-of select="../Nome"/></nome_cli>
   </xsl:template>
   <xsl:template match="/Clientes/Cliente/Sexo">
      <xsl:choose>
         <xsl:when test="text()='M'">
            <sexo_cli>1</sexo_cli>
         </xsl:when>
         <xsl:when test="text()='F'">
```

```
            <sexo_cli>2</sexo_cli>
         </xsl:when>
         <xsl:otherwise>
            <sexo_cli><xsl:value-of
select="../Sexo"/></sexo_cli>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>
<xsl:template match="text()">
</xsl:template>
</xsl:stylesheet>
```

A aplicação da transformação gerará o documento XML a seguir:

Documento XML gerado pela aplicação da transformação de "charlie" para "delta":

```
<?xml version="1.0" encoding="UTF-8"?>
<CadCliente>
     <cliente>
          <cod_cli>78</cod_cli>
          <nome_cli>Antonio Silva</nome_cli>
          <sexo_cli>1</sexo_cli>
     </cliente>
     <cliente>
          <cod_cli>79</cod_cli>
          <nome_cli>Luiza Pereira</nome_cli>
          <sexo_cli>2</sexo_cli>
     </cliente>
</CadCliente>
```

Podemos observar portanto, que a utilização de XSLT retira do interior dos programas a funcionalidade de adaptação dos dados sendo integrados, de forma a simplificar, em muito, o desenvolvimento dos aplicativos. A parte mais extensa do trabalho de integração torna-se responsabilidade do desenvolvedor dos *stylesheets* XSL que realizarão a transformação nos documentos XML quando necessário.

Retirar o processo de transformação dos dados do interior da aplicação, e colocá-lo em um processador XSLT reduz o trabalho do desenvolvedor do aplicativo, mas gera a necessidade do desenvolvimento de *stylesheets* XSL para realizar as transformações. Para que esse desenvolvimento fosse facilitado, foi desenvolvido um software capaz de gerar, via mapeamento gráfico, os documentos XSL necessários. Esse gerador de *stylesheets* é apresentado em detalhes na seqüência.

4.3 O Gerador de s*tylesheets*

O ponto central do *framework* proposto para a integração de sistemas é o gerador de *stylesheets*. Para tanto, foi desenvolvido uma aplicação denominada TIXP – The Integrated XML Program. O TIXP foi desenvolvido completamente em Java, utilizando a biblioteca JAXP 1.1, da Sun. JAXP é uma implementação das API's DOM, SAX e da especificação XSL [SPE 2000].

O propósito inicial do projeto do TIXP era desenvolver um software capaz de gerar os *templates* XSL necessários à transformação de uma árvore XML, com uma estrutura "A", em uma árvore com a estrutura "B". No projeto inicial não estava contemplada nem mesmo uma interface gráfica. Porém, a medida que o software ia sendo projetado e implementado, ficou evidente que uma ferramenta para manipular documentos XML e XSL seria muito mais útil se fosse capaz de demonstrar representações gráficas dos documentos envolvidos. Assim, foi implementada a representação dos documentos tanto em formato texto quanto em árvores hierárquicas.

Gerar semi-automaticamente *templates* XSL levou também a outra questão. Para testar as transformações XSL, era necessária a utilização de um processador XSLT adicional, externo ao programa, o que não era prático. Portanto, a capacidade de aplicar as *stylesheets* criadas aos documentos XML foi também incorporada ao programa.

Por último, como o programa já possuía a capacidade de gerar, editar, salvar *stylesheets* XSL, e submetê-las à documentos XML, pareceu interessante a possibilidade de carregar para dentro do programa, para manipular ou apenas aplicar à documentos XML, *stylesheets* gerados externamente ao mesmo. Isto transformou o TIXP em uma ferramenta completa para o desenvolvimento em XSL. De toda essa capacidade, integrada em um único programa, surgiu o nome TIXP – The Integrated XML Program.

Na sequência é apresentado o TIXP, com a seqüência de execução de um estudo de caso, simplificado, criado apenas à título de demonstração das capacidades da ferramenta, e de como a mesma encaixa-se no contexto do *framework* proposto.

4.4 TIXP – The Integrated XML Program

O sistema TIXP é um software de apoio ao trabalho com XML e XSLT. Através do TIXP, pode-se:

- Examinar documentos XML, na forma textual ou de árvore hierárquica;
- Gerar *templates* XSLT para converter um arquivo de uma estrutura XML para outra, podendo esses *templates* alterar também o conteúdo dos documentos;
- Fazer alterações nos *templates* gerados, para customização destes;
- Aplicar as transformações XSL aos documentos XML, podendo observar os resultados gerados na forma de:
  - Árvores hierárquicas XML,
  - Documentos texto e
  - HTML;
- Editar documentos XSL gerados por outros programas;
- Salvar os documentos XML e XSL manipulados.

Visando a utilização em escala global, o sistema TIXP teve toda a sua interface desenvolvida em lingua inglesa. Esta interface foi dividida em duas partes, organizadas na forma de folders. No primeiro folder, o folder "Tree", o espaço foi dividido em três áreas. Nas duas primeiras, da esquerda para a direita, são apresentadas as representações em forma de árvore para os documentos XML para os quais estará se desenvolvendo *stylesheets*. Na terceira parte, é apresentada uma tabela com os nodos da árvore selecionados para ser mapeados de uma estrutura para a outra, e os *templates* XSL gerados.

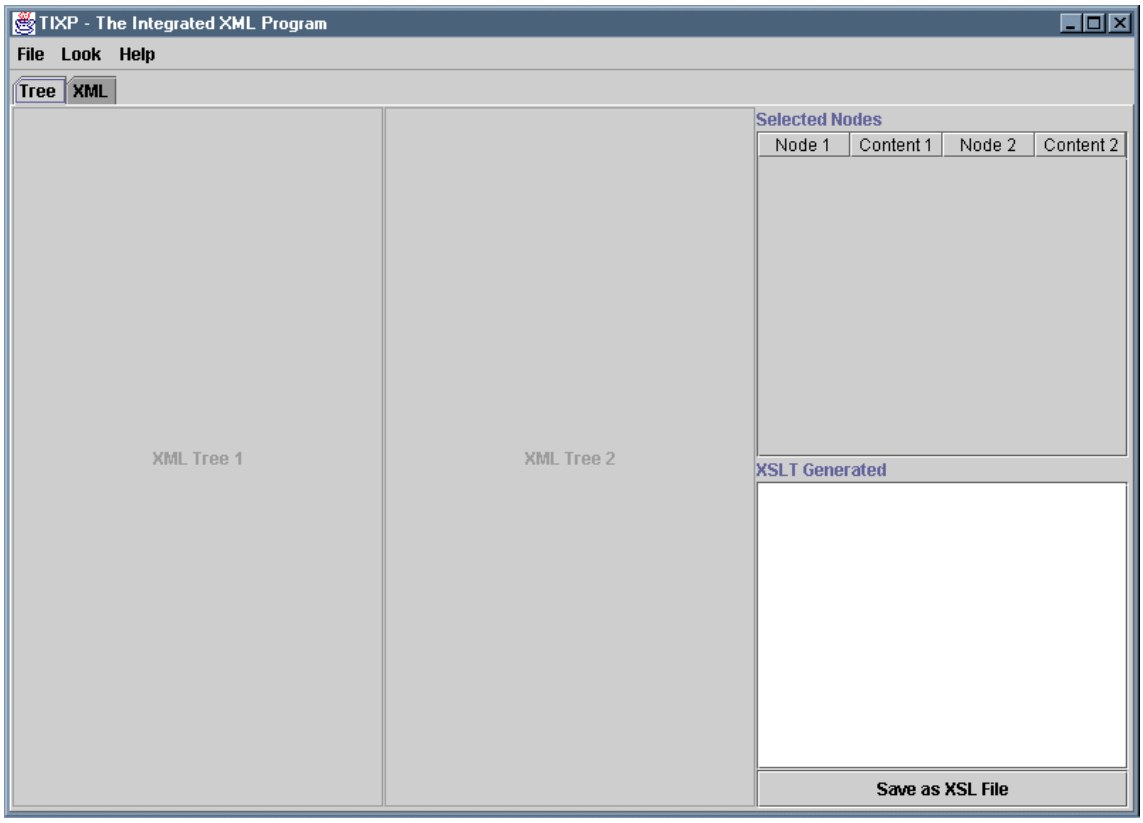

FIGURA 4.3 - Tela principal do TIXP

Houve uma preocupação constante, durante o desenvolvimento do aplicativo, em fazer com que a interface fosse o mais simples possível, para facilitar a utilização do aplicativo, sem fazer com que este viesse a tornar-se mais um fator elevador do nível de complexidade em um contexto já bastante complexo.

O segundo folder, denominado "XML", está também distribuído em três áreas, sendo que a primeira apresenta a representação XML do documento apresentado na árvore XML do primeiro objeto do folder demonstrado na figura anterior, o segundo o documento XSL, que pode ter sido gerado pelo TIXP, ou carregado do meio externo, e a terceira representa o documento XML resultante da transformação resultante da aplicação do conteúdo do *stylesheet*.

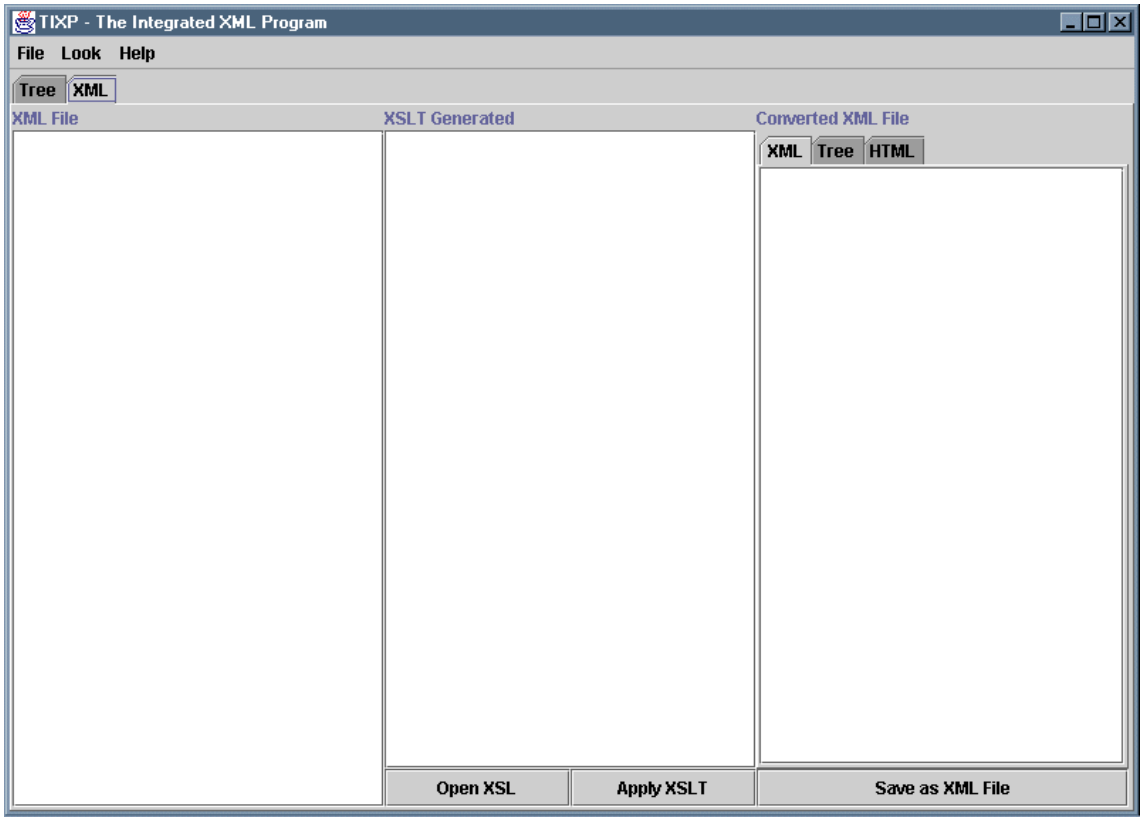

FIGURA 4.4 - Folder "XML"

Os documentos XML resultantes podem ser observados em três formas: Em formato texto, na forma de árvore hierárquica, ou na forma de HTML. Esta última forma foi inserida no projeto, devido ao fato de uma das maiores linhas de utilização do XML ser a representação de informações que serão "formatadas" posteriormente para distribuição via Web, na forma de documentos HTML. Pensando dessa forma, e visualizando o TIXP como uma ferramenta de apoio ao trabalho com XSL, a inclusão de um objeto capaz de renderizar documentos HTML acrescentaria uma funcionalidade bastante desejada ao mesmo.

Atendendo à padrões de interface Windows, é possível acessar as funções básicas do aplicativo através de menus.

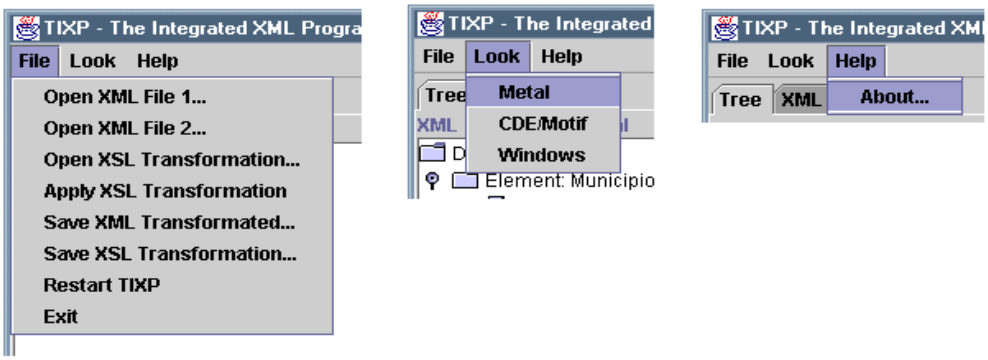

FIGURA 4.5 - Os menus do TIXP

Começa-se a utilização do TIXP escolhendo um arquivo XML, do qual deseja-se mapear os dados para outra estrutura. Isso é feito através da escolha da função "Open XML File 1", no menu "File". Uma caixa de diálogo onde pode-se escolher por um arquivo XML para manipular, será mostrada. Escolhe-se então um documento XML, que será apresentado no programa como segue:

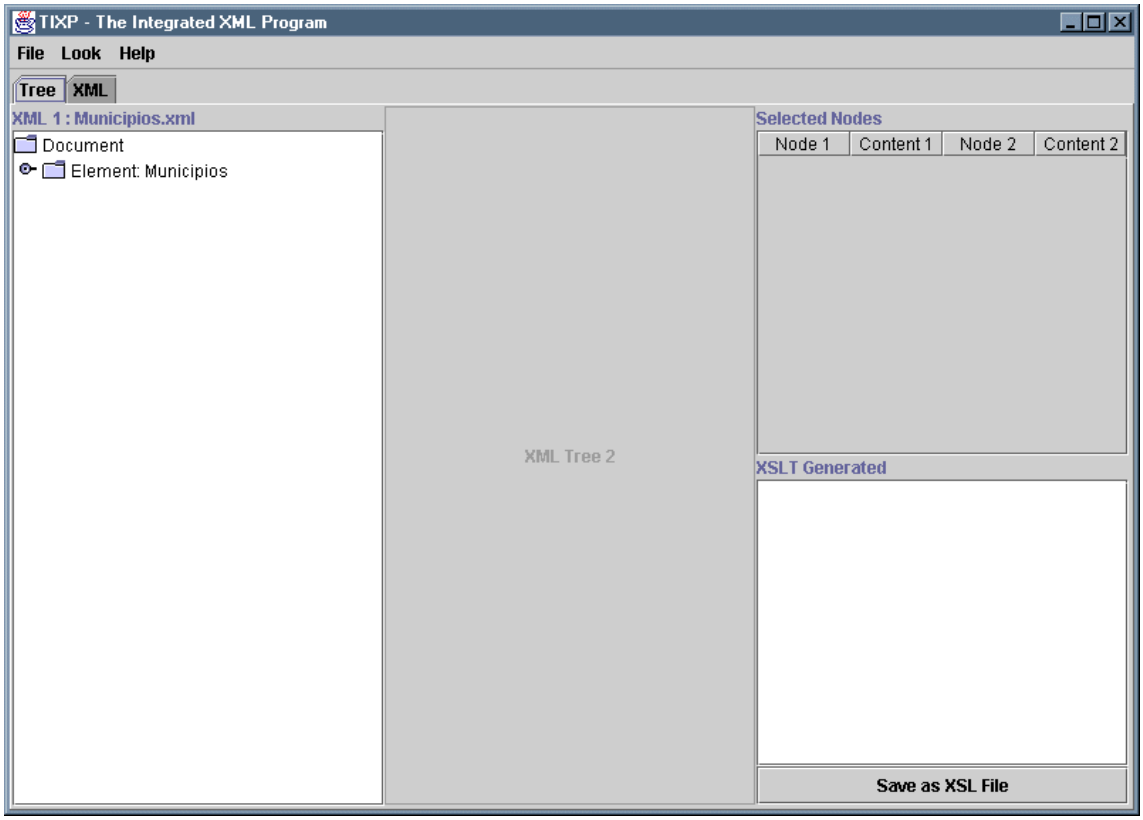

FIGURA 4.6 - TIXP com o documento XML 1 já carregado

Em nosso exemplo, vamos mapear as informações constantes de um documento XML chamado "municipios" em outro, de estrutura diferente, chamado "cidades". Para tanto, escolhe-se o como primeiro documento o arquivo "municipios.xml".

O documento escolhido é então apresentado no TIXP, na forma de árvore hierárquica. Note-se que, por padrão, as árvores são apresentadas em sua forma reduzida, sendo necessário, se desejado, aplicar cliques do mouse sobre a mesmas para uma visualização expandida de seus conteúdos.

Carrega-se então o segundo documento, "cidades.xml", através da escolha da função "Open XML File 2", no menu "File". Como na ação anterior, será mostrada novamente uma caixa de diálogo para a escolha do arquivo XML. Escolhe-se então o documento, que será apresentado no TIXP como demonstrado:

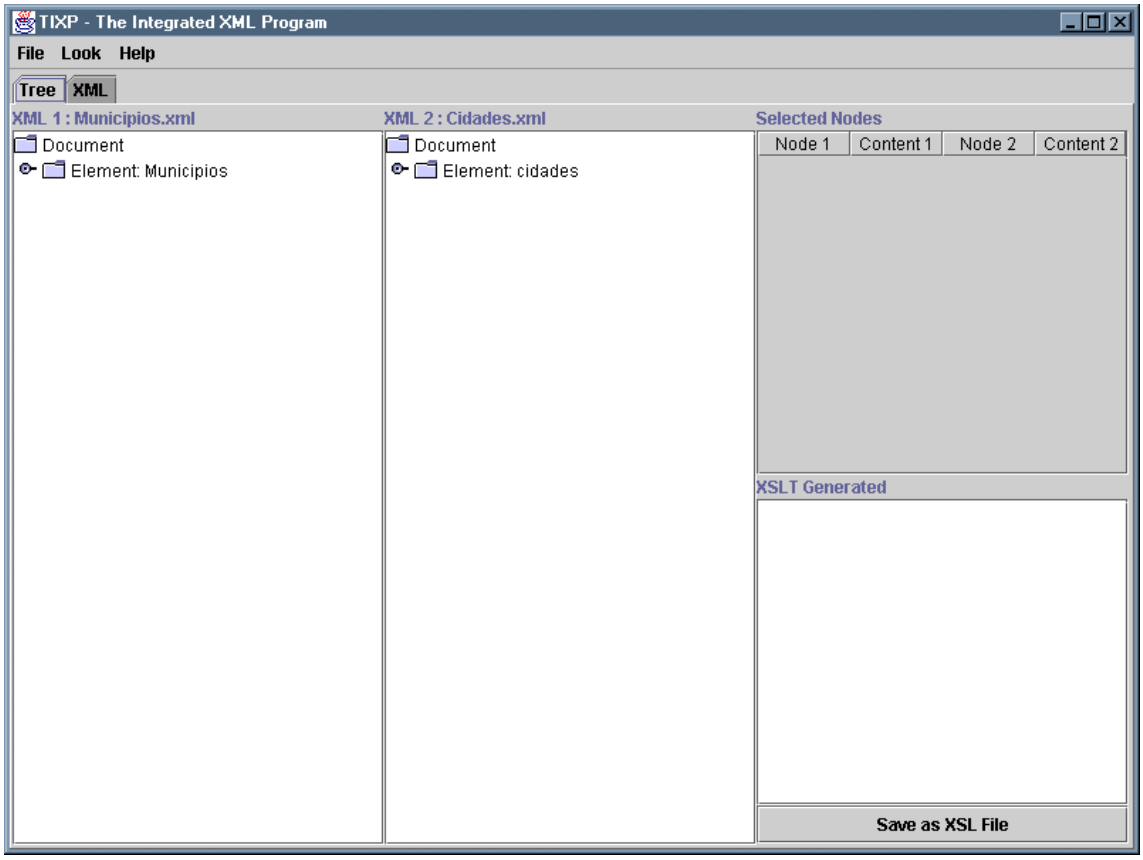

FIGURA 4.7 - TIXP com os dois documentos XML carregados

As árvores representando os documentos XML podem então ser expandidas, simplificando a viualização das informações contidas nos mesmos. Essa expansão das árvores, mesmo que parcialmente, é necessária para a operação de mapeamento estrutural que será realizada na seqüência.

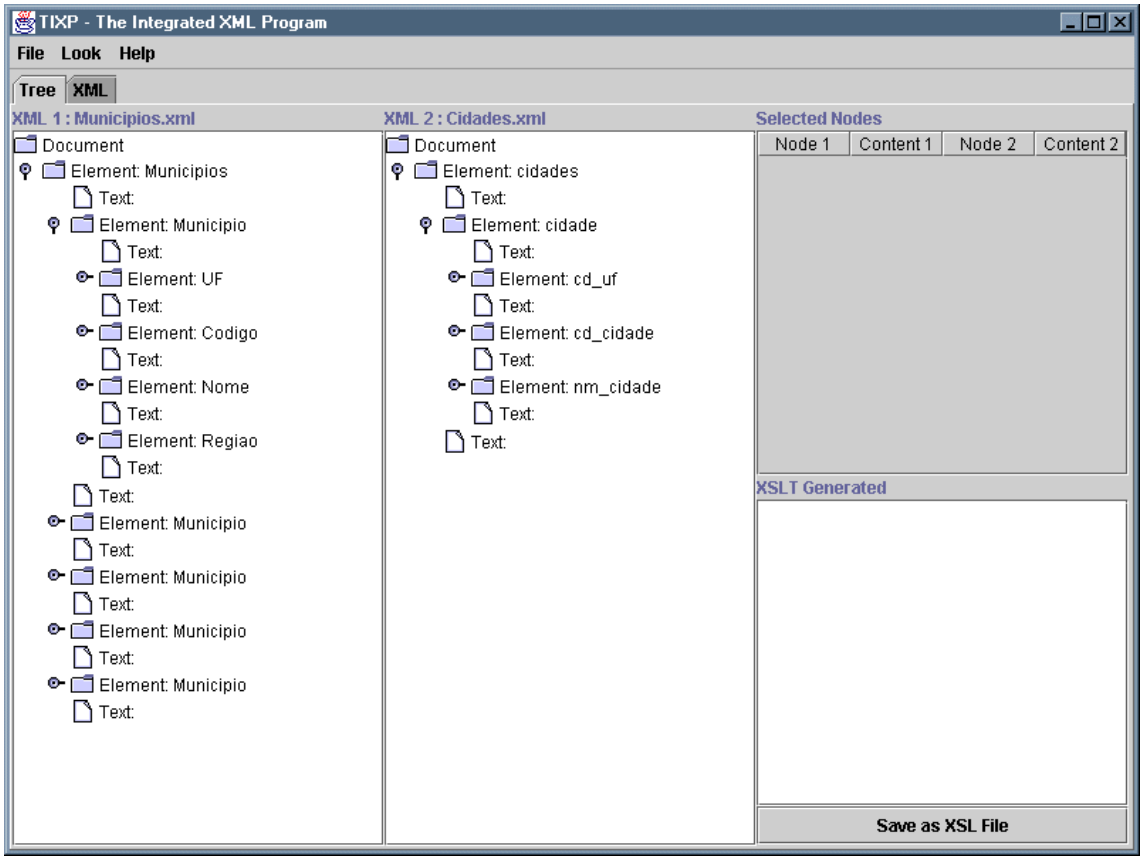

FIGURA 4.8 - As árvores XML expandidas

Antes, porém, é demonstrado que o conteúdo do documento XML escolhido como origem dos dados pode ser visualizado também na forma textual do XML, através do primeiro objeto do folder XML. Nessa visualização não é permitida a edição do documento para alteração de seus conteúdos.

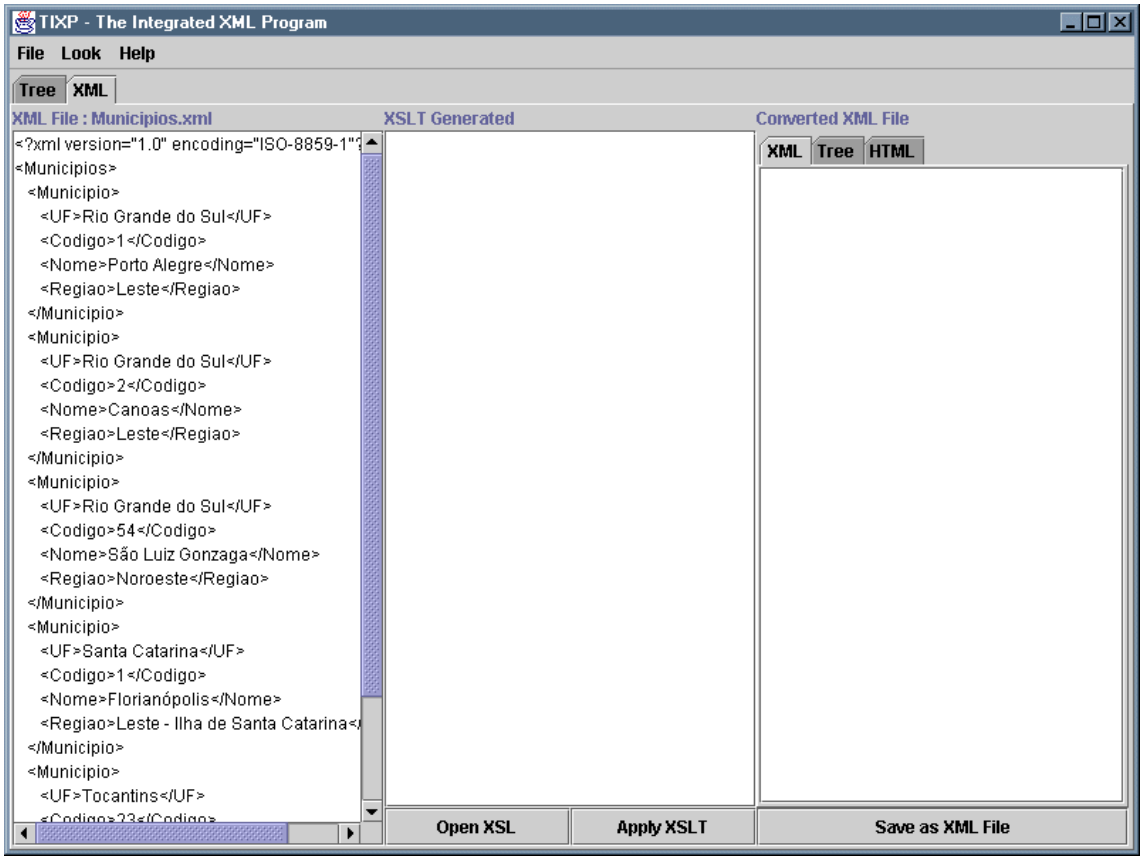

FIGURA 4.9 - Representação textual do documento XML 1

A representação textual do documento XML 2 não é demonstrada, uma vez que o o conteúdo do mesmo não é levado em consideração no momento da realização dos mapementos. Aproveita-se portanto, somente a estrutura do documento XML 2 na criação dos *stylesheets*.

Começa-se então o processo de mapeamento do documento XML 1 para o documento XML 2. Isto é feito através da opção "Insert" de um menu acionado pelo botão direito do mouse sobre o nodo desejado.

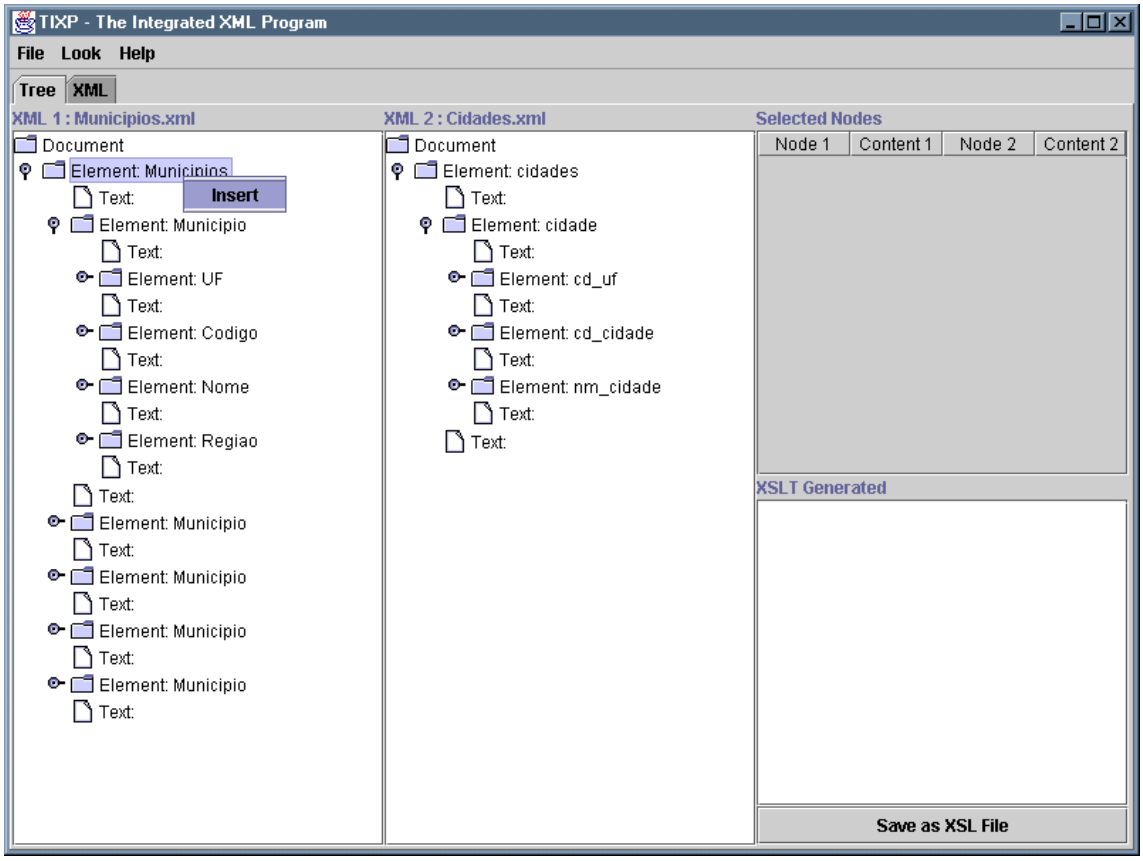

FIGURA 4.10 - O menu "Insert"

No momento do mapeamento dos nodos que compõe um documento XML, faz-se necessário sempre o mapeamento do elemento raíz (*root*) do documento XML 1 para o elemento raíz do documento XML 2, para a geração de um *stylesheet* válido, seguindo a especificação do XSL.

O nodo selecionado é então inserido em uma tabela de nodos para mapeamento, sob a coluna "Node 1".

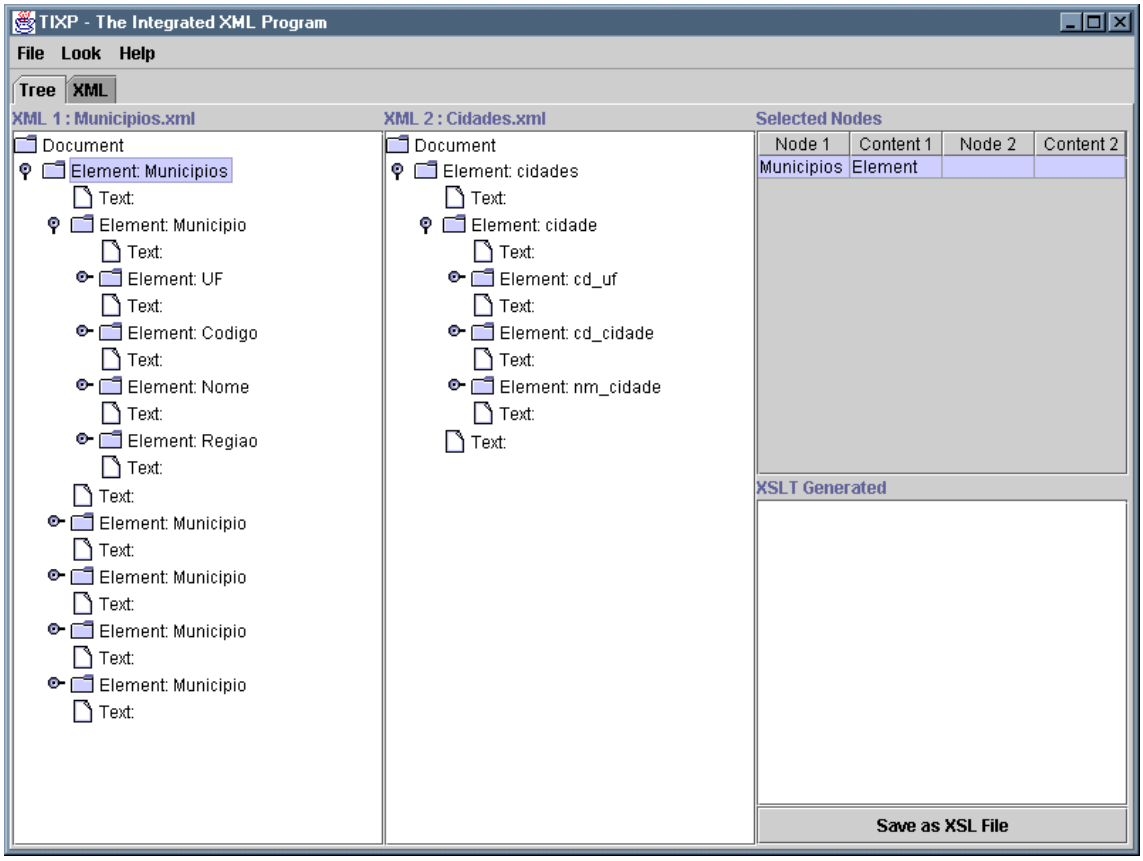

FIGURA 4.11 - Inserção do primeiro nodo na tabela de mapementos

Uma vez inserido o primeiro nodo do documento XML 1, escolhe-se o nodo do documento XML 2 para o qual o primeiro conteudo será mapeado.

De forma análoga à escolha do primeiro nodo do documento XML 1, a escolha do nodo correspondente do documento XML 2 é realizada através do acionamento do botão direito do mouse sobre o nodo desejado, e a posterior escolha da opção "Set/Change".

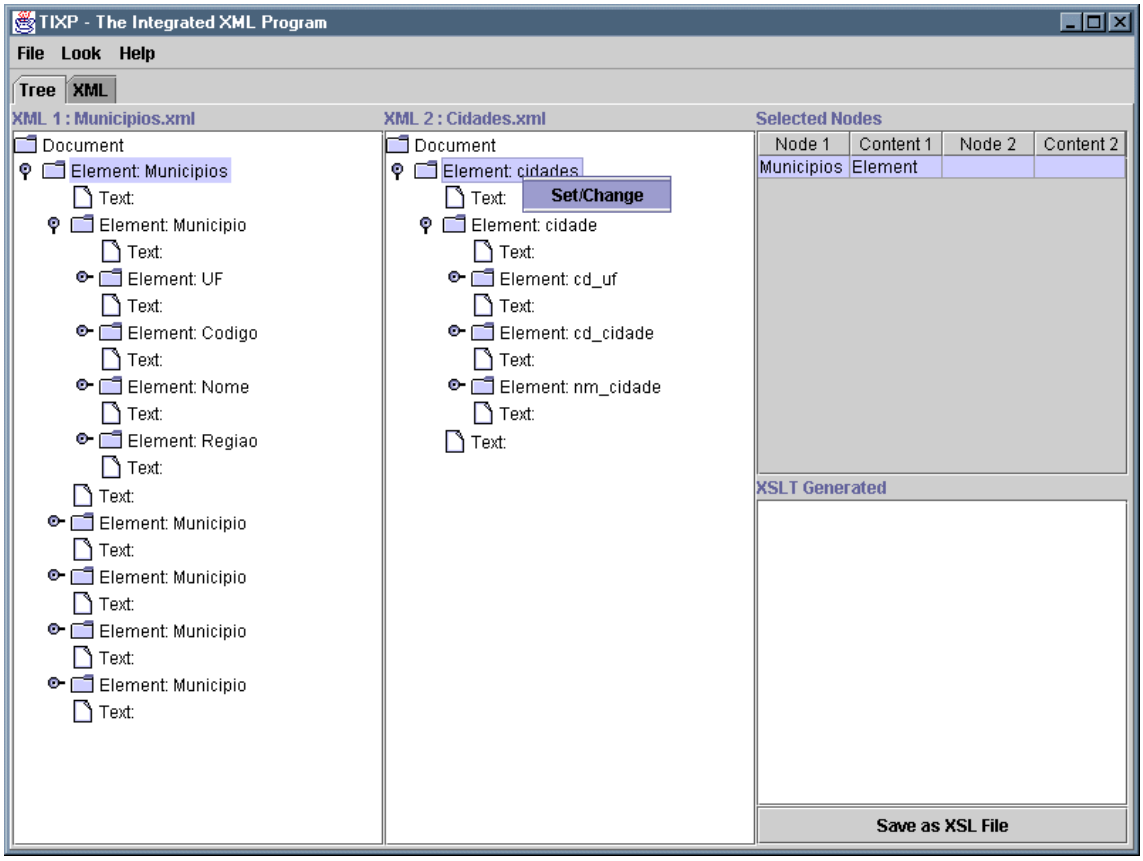

FIGURA 4.12 - Escolha do nodo correspondente no documento XML 2

O opção apresentada é "Set/Change", ao invés de "Insert", como no documento anterior, porque pode-se escolher um nodo do XML 2 para corresponder ao nodo do XML 1 já escolhido, e depois voltar atrás e escolher outro. É portanto, possível alterar opções já realizadas sobre mapeamentos.

O nodo escolhido é inserido na coluna Node 2 da tabela "Selected Nodes".

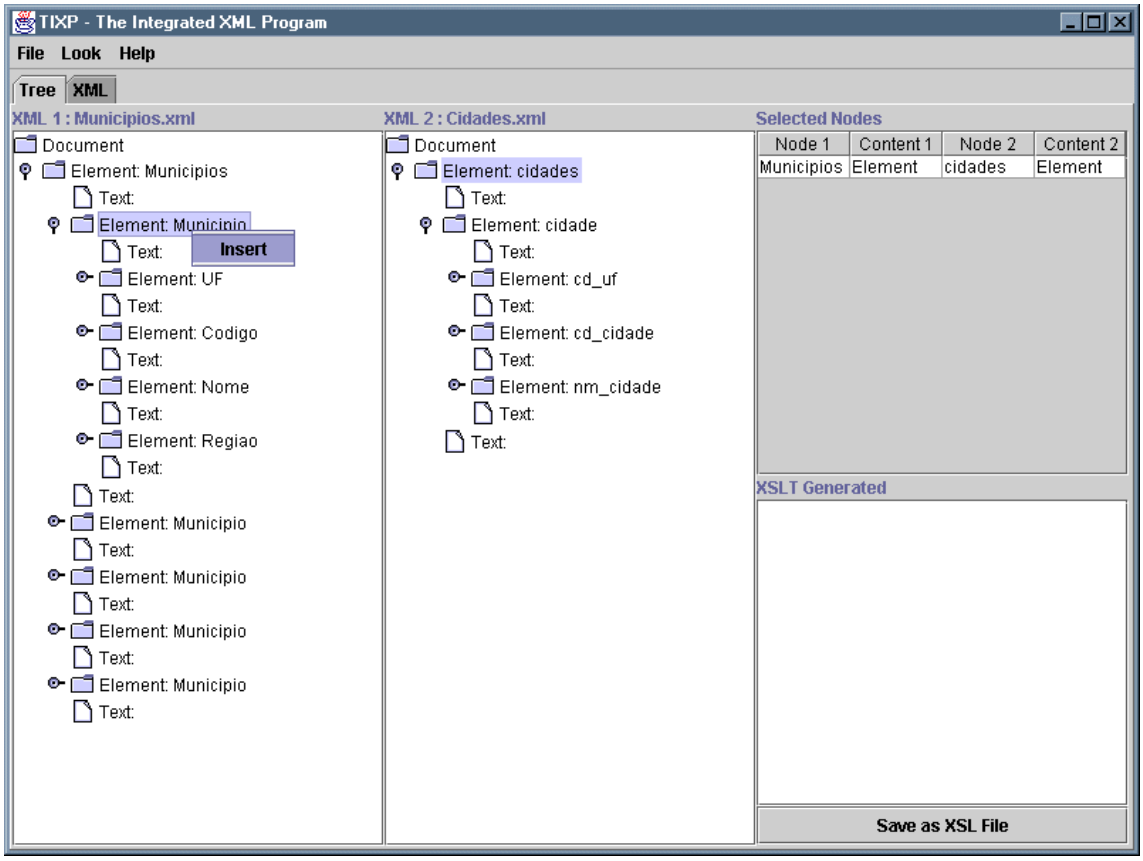

FIGURA 4.13 - O segundo nodo é inserido na tabela de mapeamentos

Note-se que os nodos inseridos foram para as colunas "Node 1" e "Node 2" da tabela de mapeamentos. As colunas "Content 1" e "Content 2" contém apenas exemplos de valores armazenados nos nodos dos documentos XML, e não tem função no mapeamento das informações além de ilustrar os conteúdos dos elementos.

Repete-se então o processo, escolhendo os nodos desejados como origem das informações, bem como os nodos destino. Levando-se em conta que a operação de escolha do nodo raíz do documento XML 1 foi a operação de número 1, e a escolha do nodo raíz do documento XML 2 foi a operação de número 2, pode-se seguir a seqüência numérica indicada nos círculos em vermelho na ilustração abaixo para preencher a tabela "Selected Nodes"

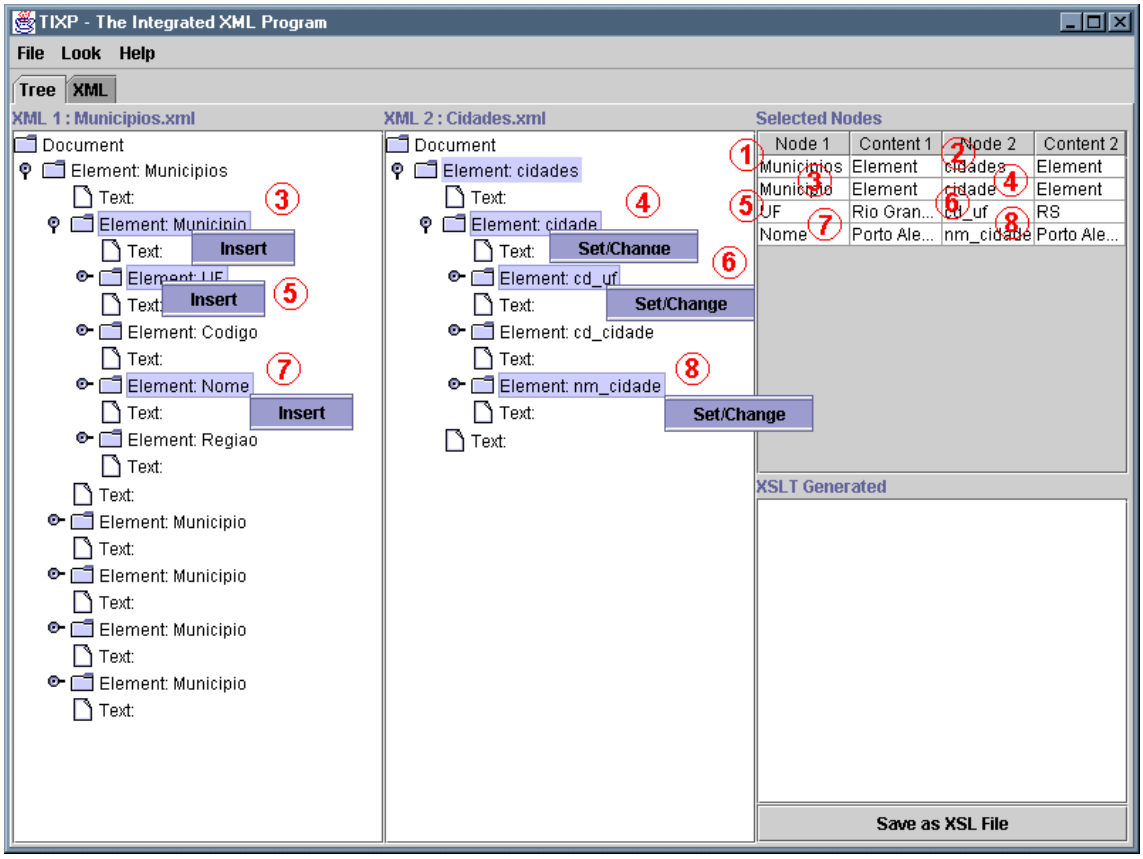

FIGURA 4.14 - Seqüência de escolha dos nodos

(Nota : Os círculos em vermelho são apenas ilustrações com fim didático e não fazem parte da implementação do TIXP.)

Uma vez preenchida a tabela de mapeamento, seu resultado deverá parecer-se com o apresentado a seguir:

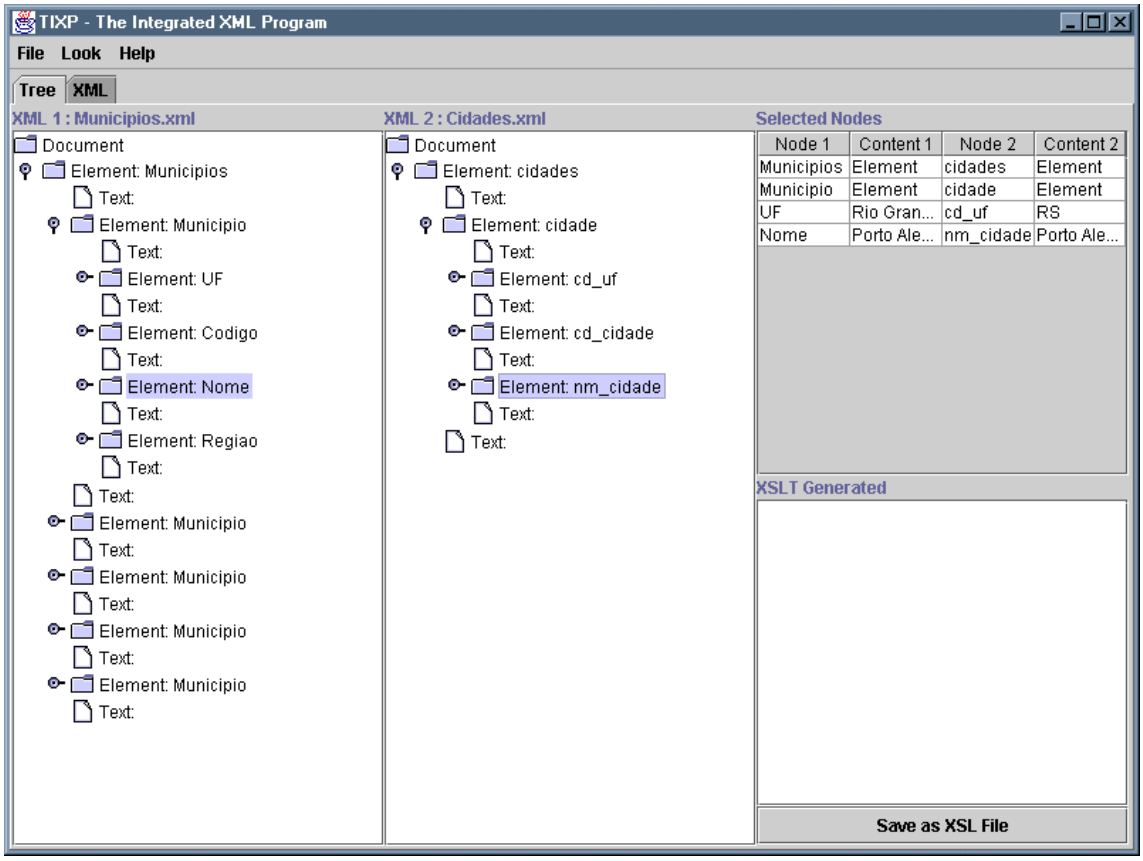

FIGURA 4.15 - A tabela de selected nodes preenchida

Uma vez preechida a tabela de nodos para mapeamento, já se pode criar *templates* ou *stylesheets* completos para aplicação. Antes ainda da geração destes *templates*, é possível definir tabelas para mapeamento de valores, que os *stylesheets* gerados sejam capazes de alterar não só a estrutura dos documentos XML, como também seus conteúdos. Um exemplo de mapeamento de valores é demonstrado mais adiante.

Pode-se gerar um *template* para um mapeamento aleatório, escolhido da tabela de mapeamentos. Para tanto, dá-se um clique do mouse sobre a linha desejada, para selecioná-la, e, em seguida, escolhe-se, no menu acionado pelo botão direito do mouse, a opção "Generate XSLT-ROW".

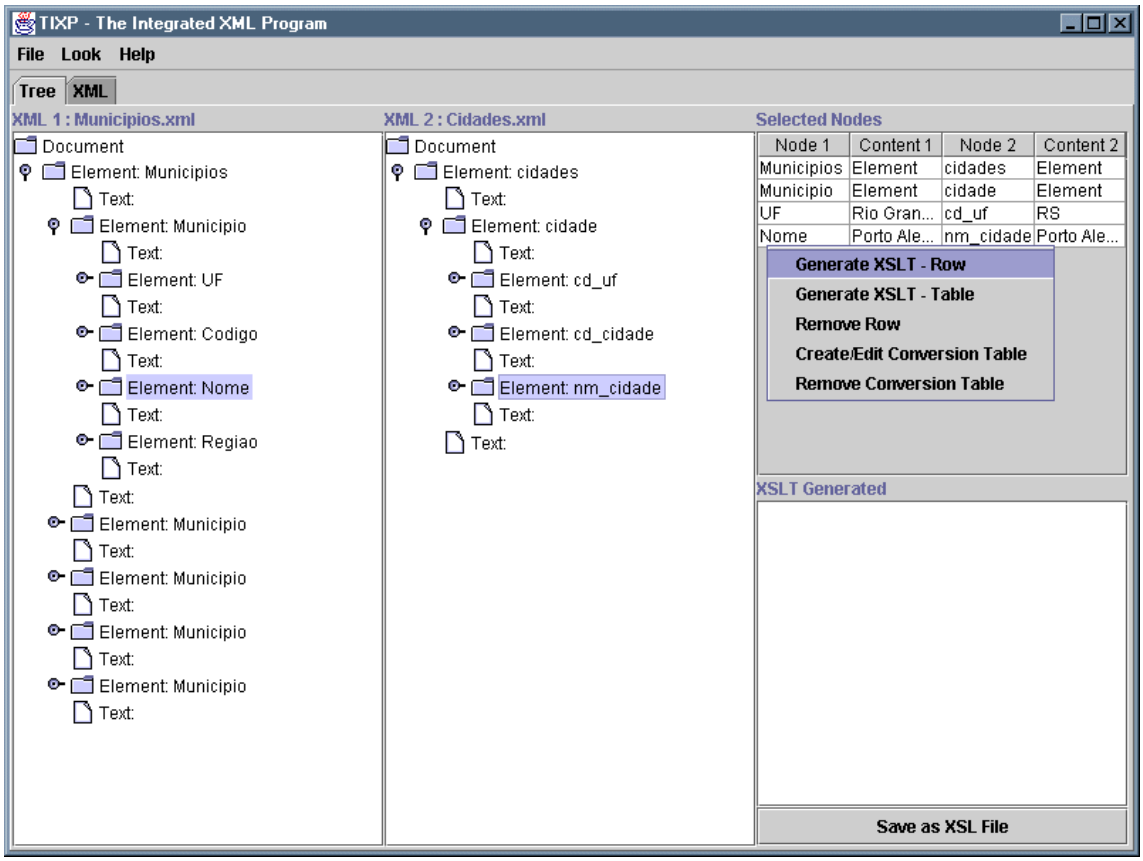

FIGURA 4.16 - A opção "Generate XSLT – Row"

Note-se que a geração de um *template* para apenas uma linha da tabela de mapeamentos serve somente como apoio ao desenvolvimento manual de *stylesheets* utilizando TIXP, uma vez que sem os cabeçalhos que identificam um arquivo como sendo um *stylesheet* XSL, este não será corretamente interpretado pelo processador XSLT.

Para a geração de um *stylesheet* completo, capaz de realizar a transformação de um documento XML, utiliza-se a opção "Generate XSLT – Table", que processa todas as linhas da tabela de mapeamentos, gerando um documento XSL com todas as transformações mapeadas.

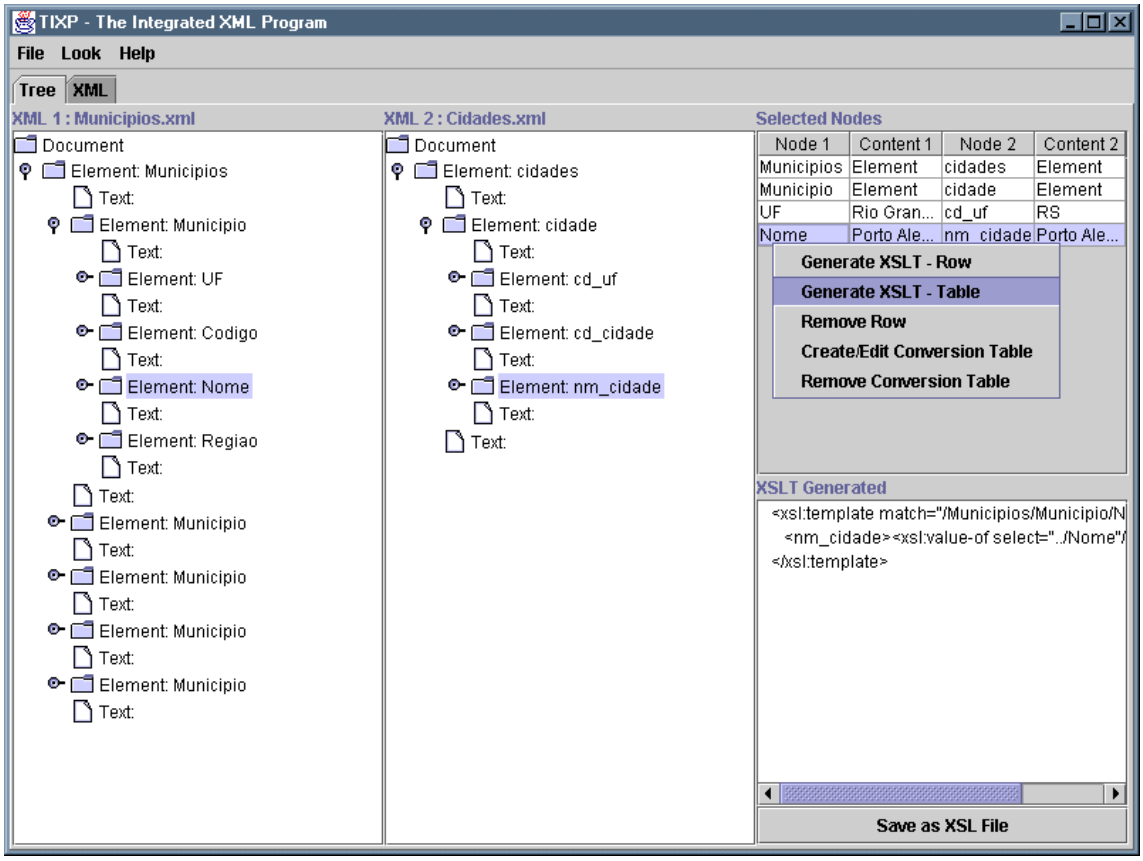

FIGURA 4.17 - A opção "Generate XSLT – Table"

Pode-se observar na ilustração acima o *template* gerado pela opção anterior, "Generate XSLT – Row". *Templates* gerados são demonstrados logo abaixo da tabela de mapeamentos, e também no objeto central do *folder* "XML".

Nas próximas duas ilustrações pode-se ver o conteúdo do documento XSL gerado, primeiro no objeto reduzido, e depois na versão ampliada, em um objeto de maior dimensão.

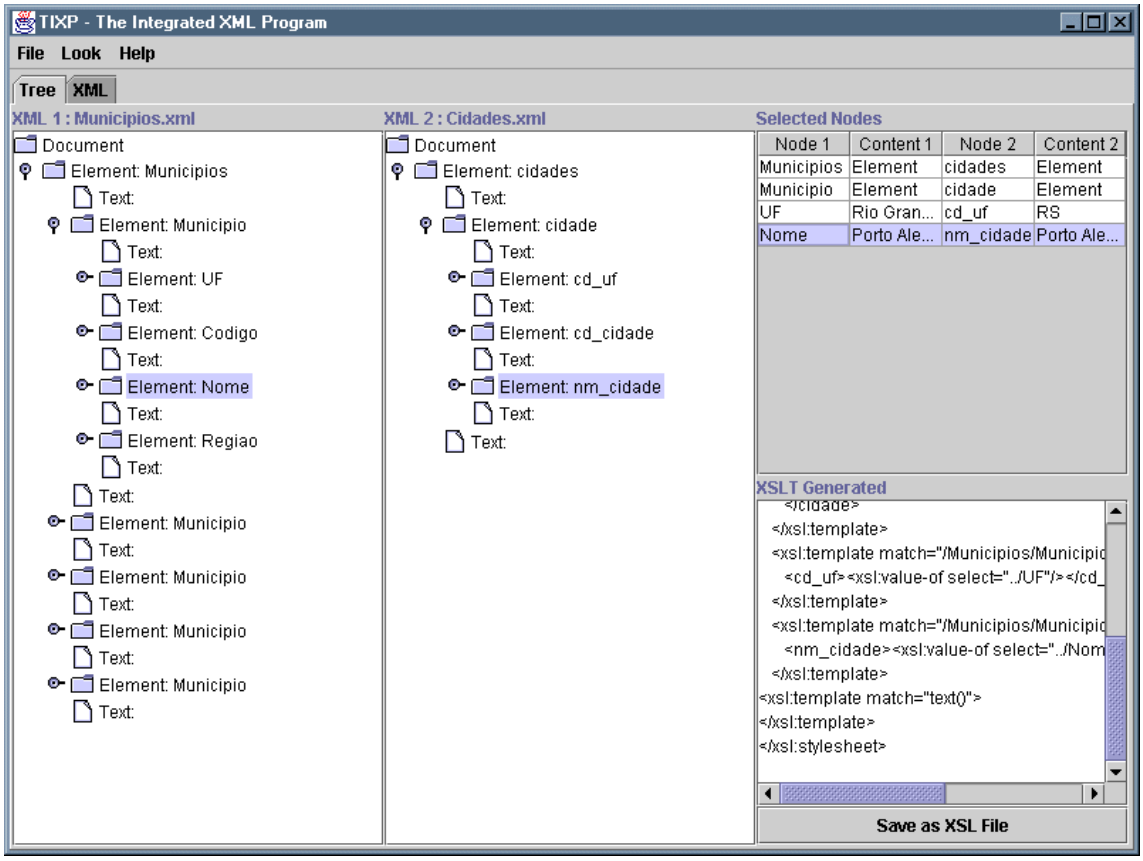

FIGURA 4.18 - Visão reduzida do *stylesheet* gerado

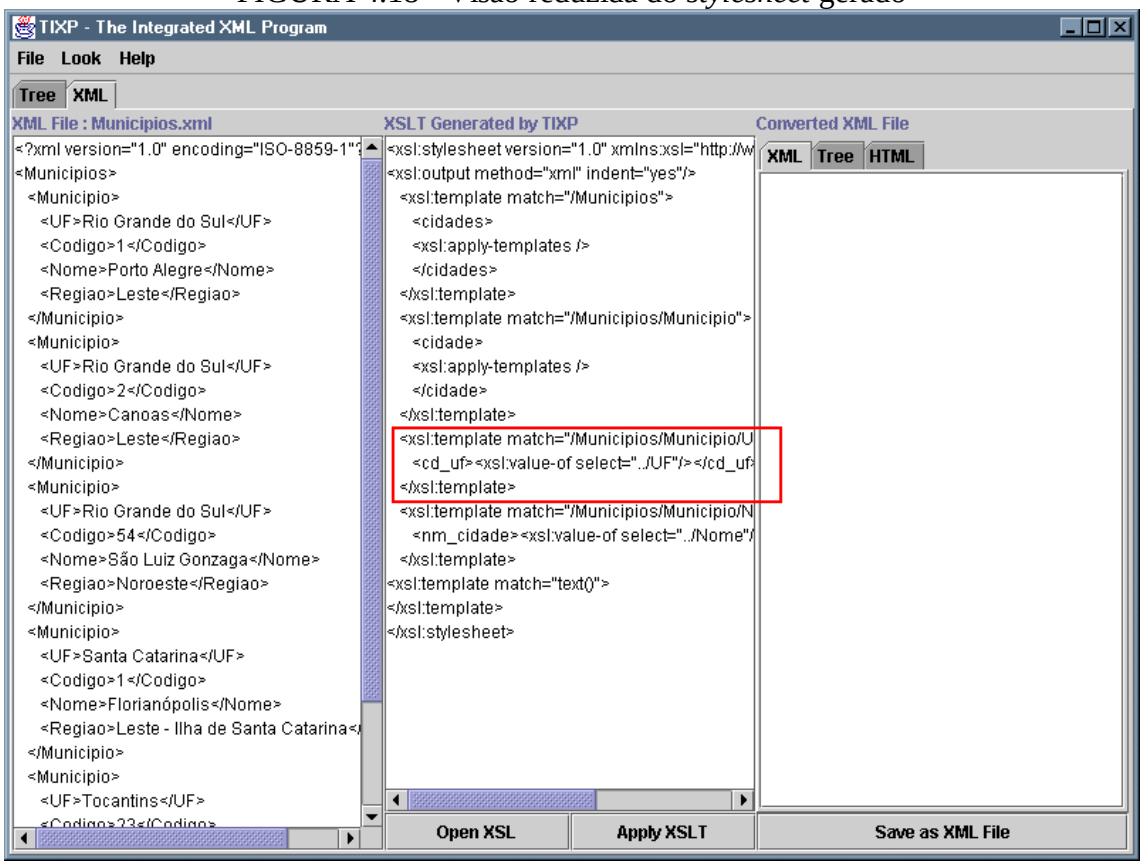

FIGURA 4.19 - Visão ampliada do *stylesheet* gerado

O *styleshet* gerado não possui a capacidade de alterar o conteúdo dos dados representados no documento XML 1. Isto é visível na parte grifada em vermelho na figura 4.19. O *template* gerado para o elemento UF é idêntico ao abaixo:

```
<xsl:template match="/Municipios/Municipio/UF">
   <cd_uf>
      <xsl:value-of select="../UF" />
   </cd_uf>
</xsl:template>
```

Este *template* simplesmente lê o valor armazenado sob o *tag* `<UF>` e o transfere para o *tag* `<cd_uf>`. Isto é feito através da instrução `xsl:template match`, do XSLT, que utiliza a expressão `"/Municipios/Municipio/UF"`, do padrão XPath, para encontrar os nodos contendo informações sobre `"UF's"`, dentro de `"Municípios"`. A instrução `<xsl:value-of select="../UF" />` transfere então o valor encontrado para o interior do par de *tags* `<cd_uf> e </cd_uf>`.

Aplica-se então a transformação XSL ao documento XML pressionando o botão "Apply XSLT", como ilustrado a seguir:

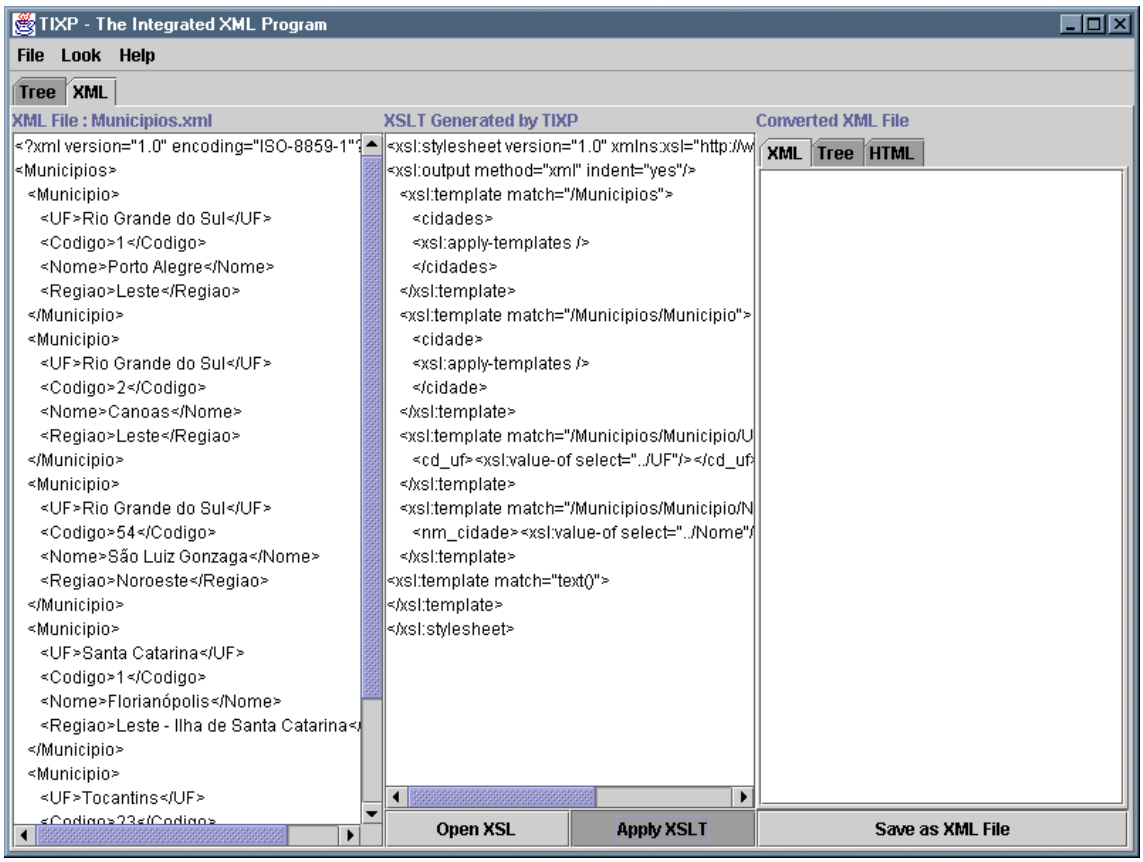

FIGURA 4.20 - O botão "Apply XSLT" em ação

Alternativamente pode-se também utilizar a opção de menu "Apply XSL Transformation", no menu "File". O Documento gerado resultante pode ser observado na figura 4.21.

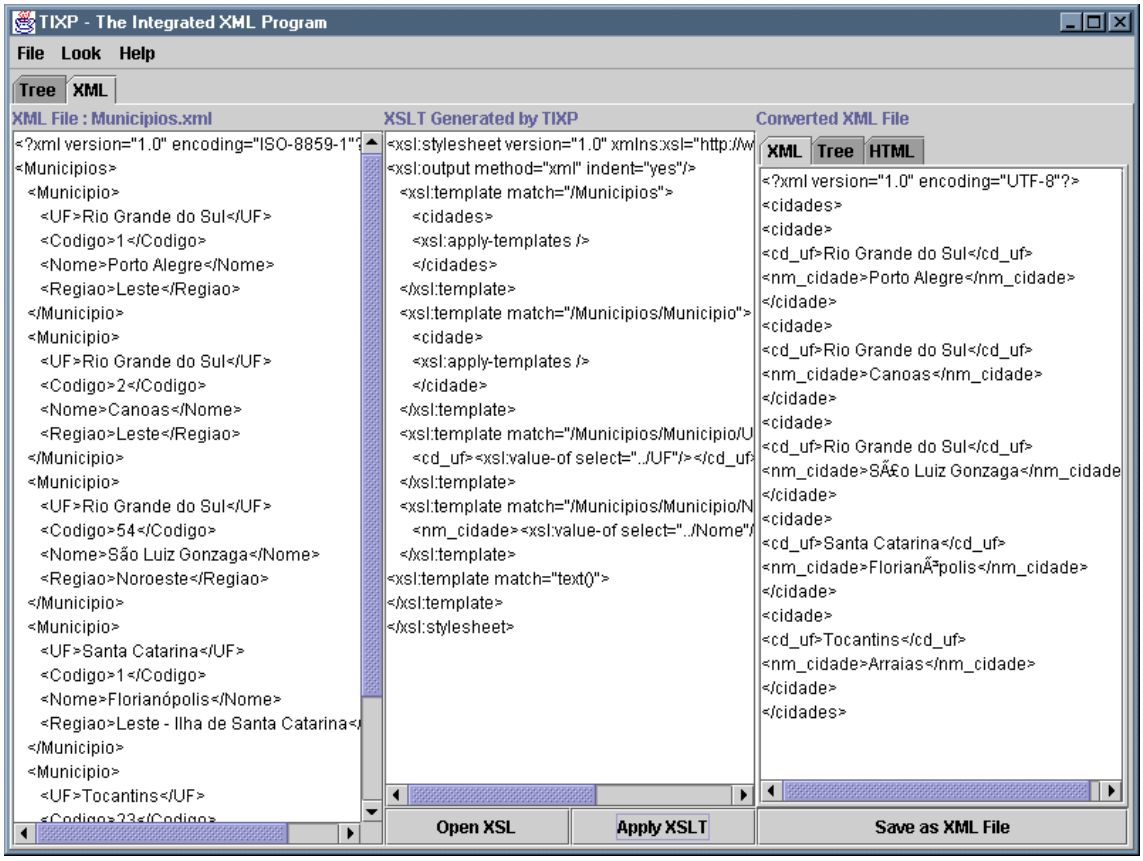

FIGURA 4.21 - O documento gerado pelo *stylesheet* XSL

O primeiro modo de visualização que o TIXP proporciona do documento gerado é a visualização na forma textual XML.

De outra forma, é possível visualizar o documento gerado na forma de árvore hierárquica, o que é feito acionando-se a opção "Tree", sob a opção "Converted XML File".

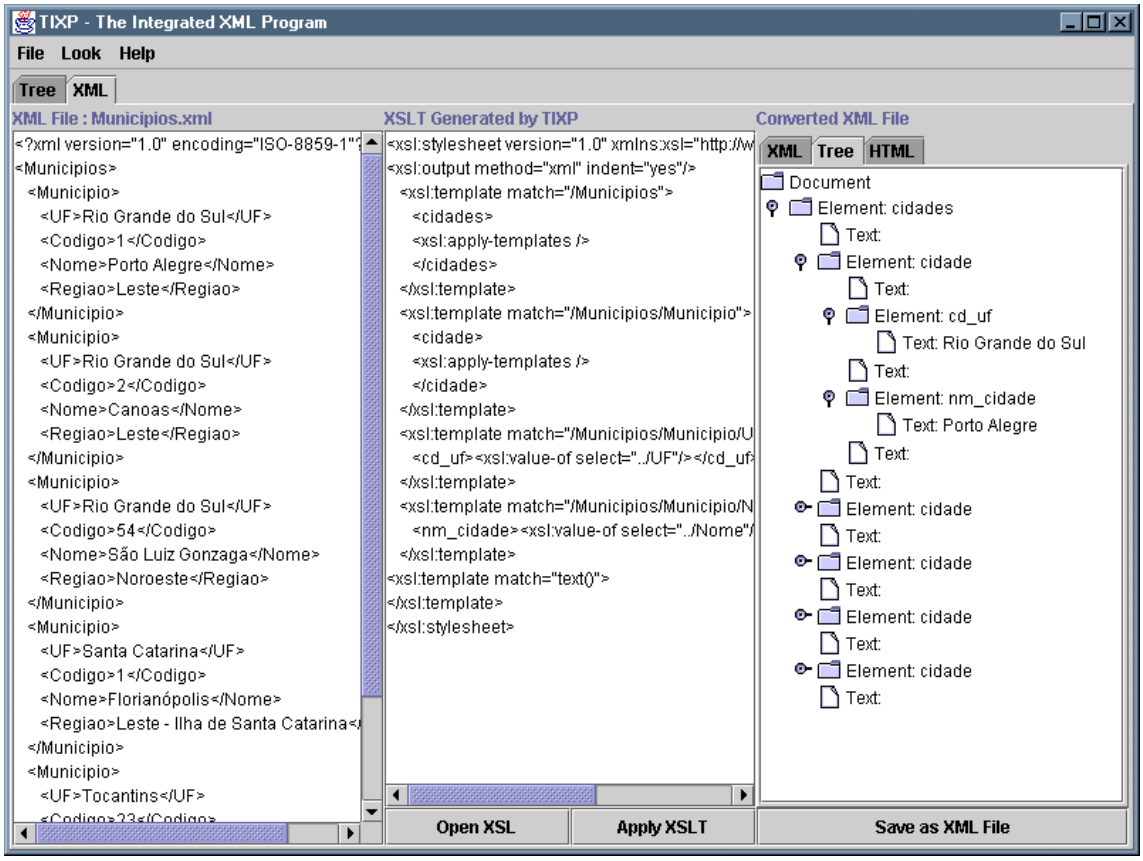

FIGURA 4.22 - A visualização do arquivo convertido em forma de árvore

Utilizando-se o TIXP da forma apresentada até aqui, pode-se realizar a transformação de uma árvore XML, representando uma determinada hierarquia para outra, independentemente do número de níveis de ambas. Essa pode ser considerada uma transformação de caráter estrutural, pois somente as estruturas que dão semântica aos dados são mudados, mas seus conteúdos são transferidos de uma árvore à outra sem qualquer alteração.

Outra forma de transformação bastante necessária, e portanto desejada em um aplicativo de apoio, é a capacidade de realizar conversões no conteúdo dos dados, através de mapeamento de domínios. Para exemplificar, imagine-se a integração de dois sistemas distintos, com informações sobre pessoas, onde no primeiro o sexo dessas é representado através de uma letra "M" para masculino ou "F" para feminino, e no segundo os sexos são representados por um número 1 ou 2.

Para a realização desse tipo de conversão, foi implementada no TIXP a capacidade de manipular tabelas de conversão de valores, que são utilizados na criação de *templates* capazes de substituir os valores encontrados no documento origem por valores fornecidos nas tabelas de conversão.

A manipulação destas tabelas é feita escolhendo-se a opção "Create/Edit Conversion Table", no menu acionado quando o botão direito do mouse é pressionado sobre a tabela de nodos selecionados. Isto é visto na figura a seguir:

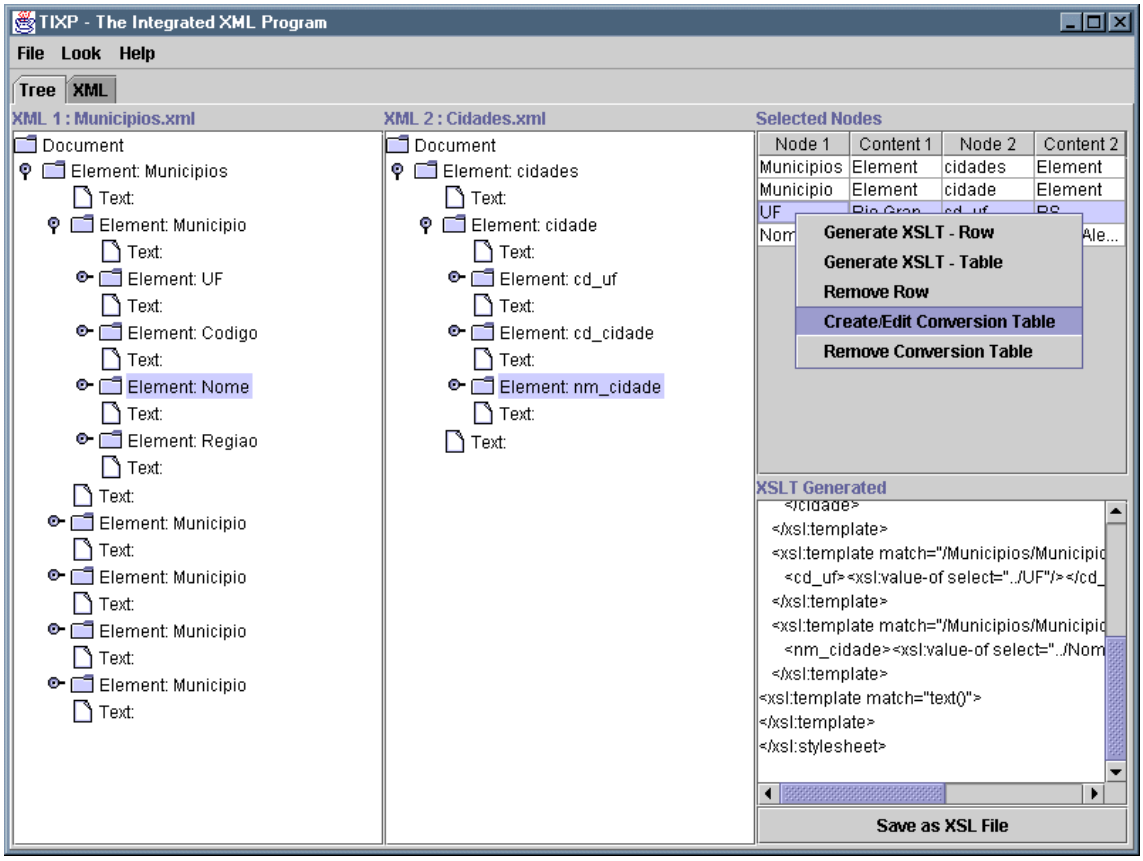

FIGURA 4.23 - A opção "Create/Edit Conversion Table"

Escolhendo-se esta opção, é mostrada uma tabela de conversão para os nodos selecionados, conforme pode ser visto na figura 4.24.

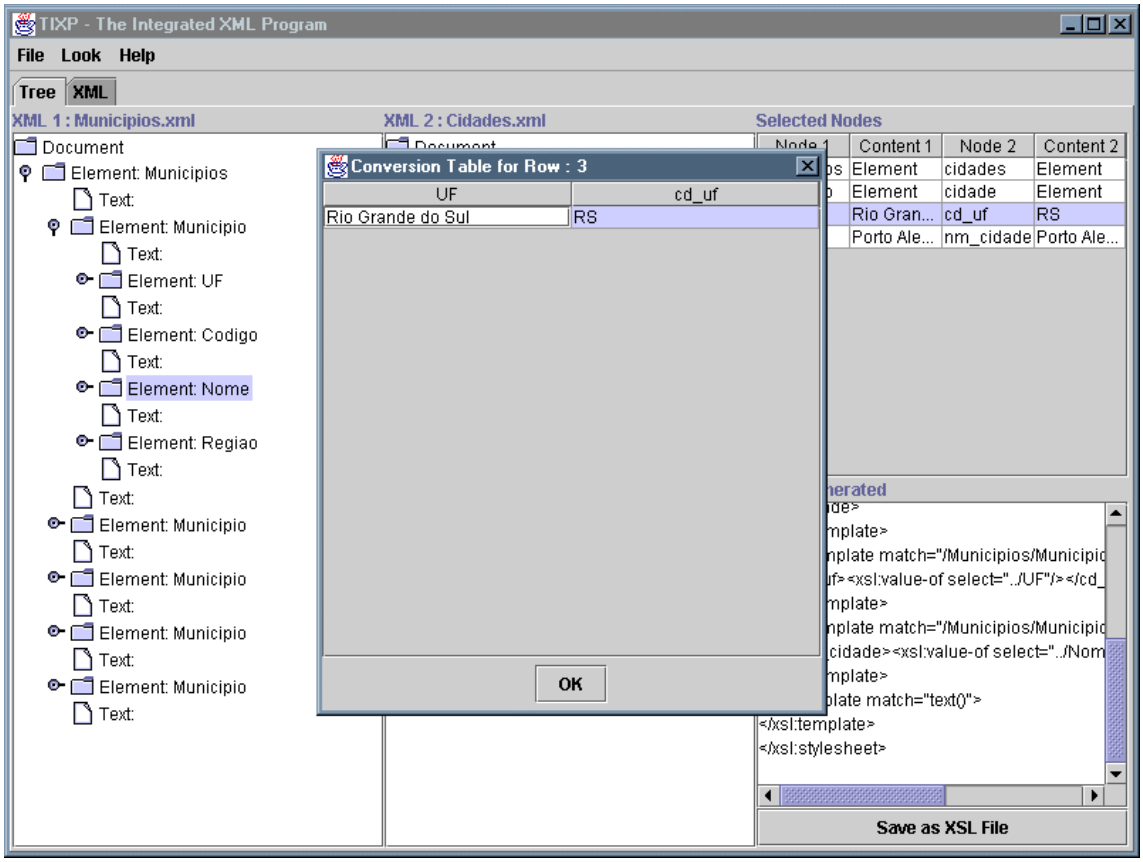

FIGURA 4.24 - Tabela de conversão de valores

A tabela de conversão de valores é criada com apenas uma linha. Preenche-se o os valores "From" e "To" da tabela, que indicam respectivamente qual o dado a ser substituído e qual o valor de substituição. Para criar mais linhas, utiliza-se a opção "Add Row" do menu suspenso acionado pelo pressionamento do botão direito do mouse sobre as linhas da tabela de conversão. Isso é demonstrado na figura 4.25.

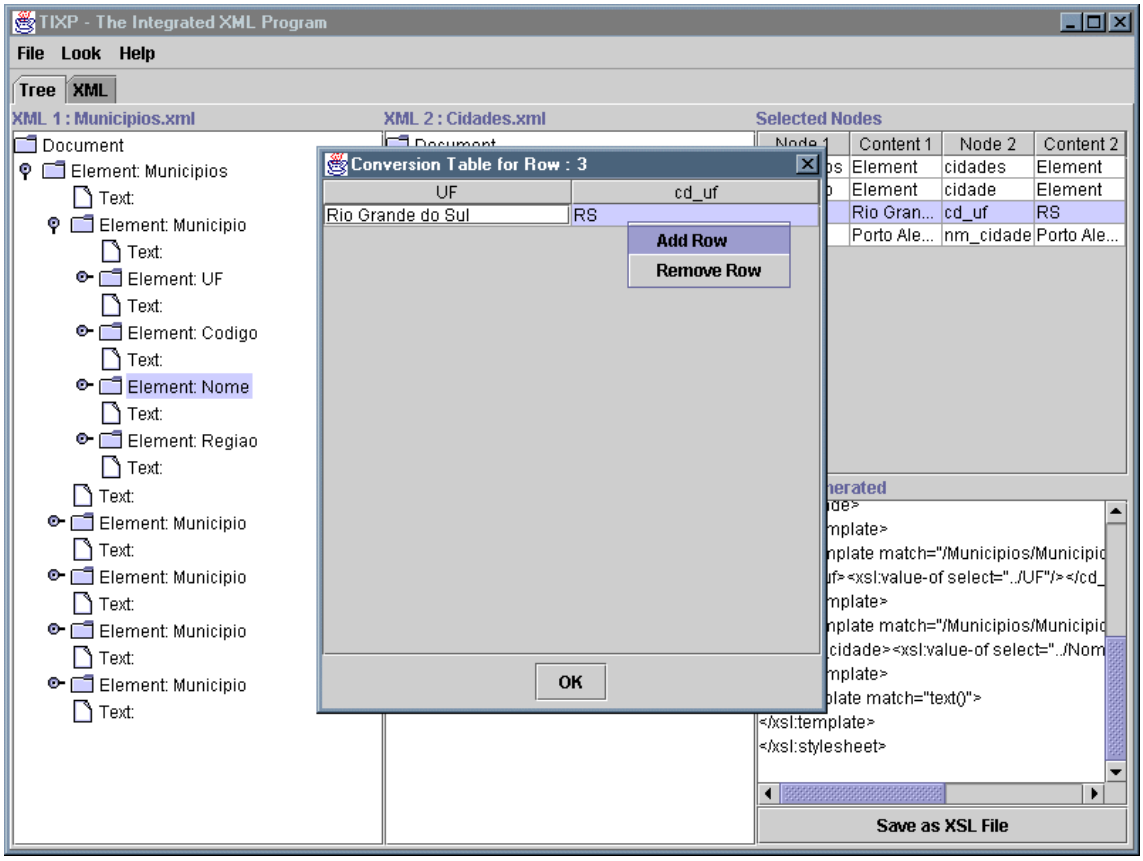

FIGURA 4.25 - A opção "Add Row" na tabela de conversão de valores

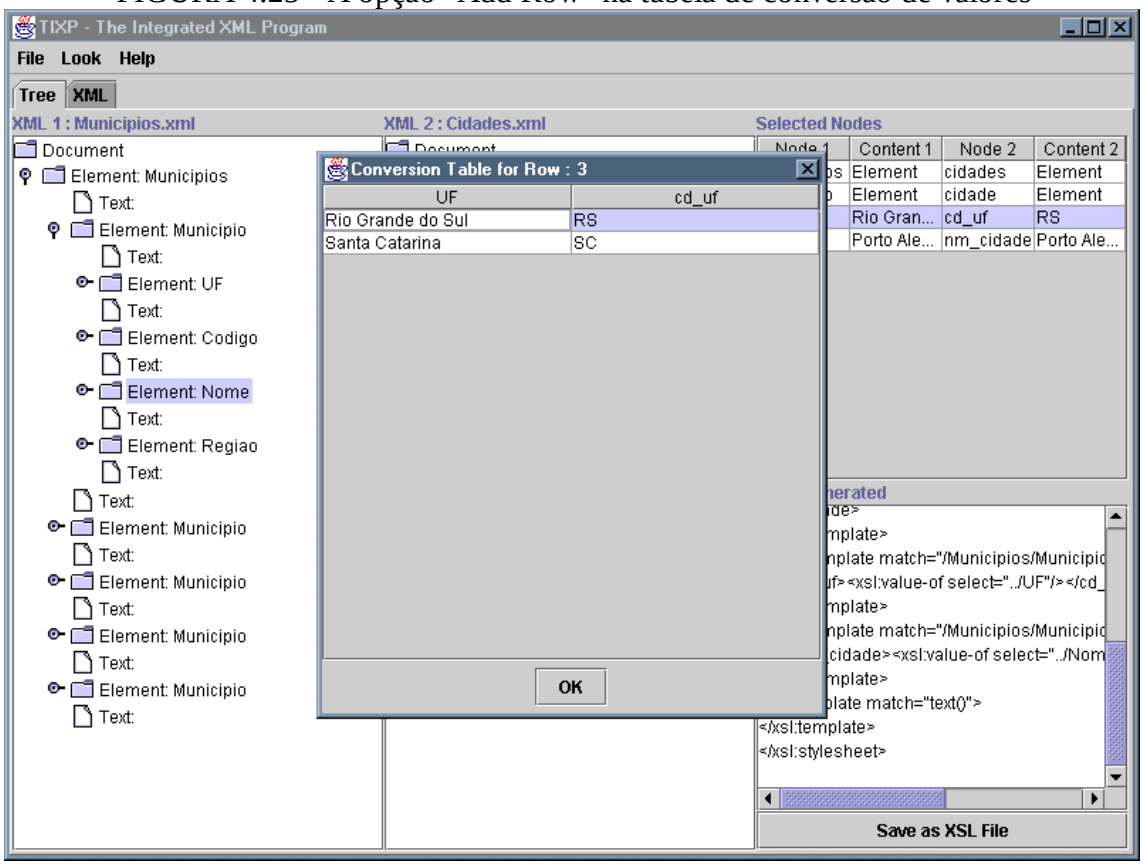

FIGURA 4.26 - Uma tabela de conversão completa

Estando preenchida a tabela de conversão, pressiona-se o botão OK. A tabela é ocultada e é possível agora gerar novamente os *templates* XSL. Novamente escolhe-se opção "Generate XSLT – Table", no menu suspenso acionado sobre a tabela de nodos selecionados.

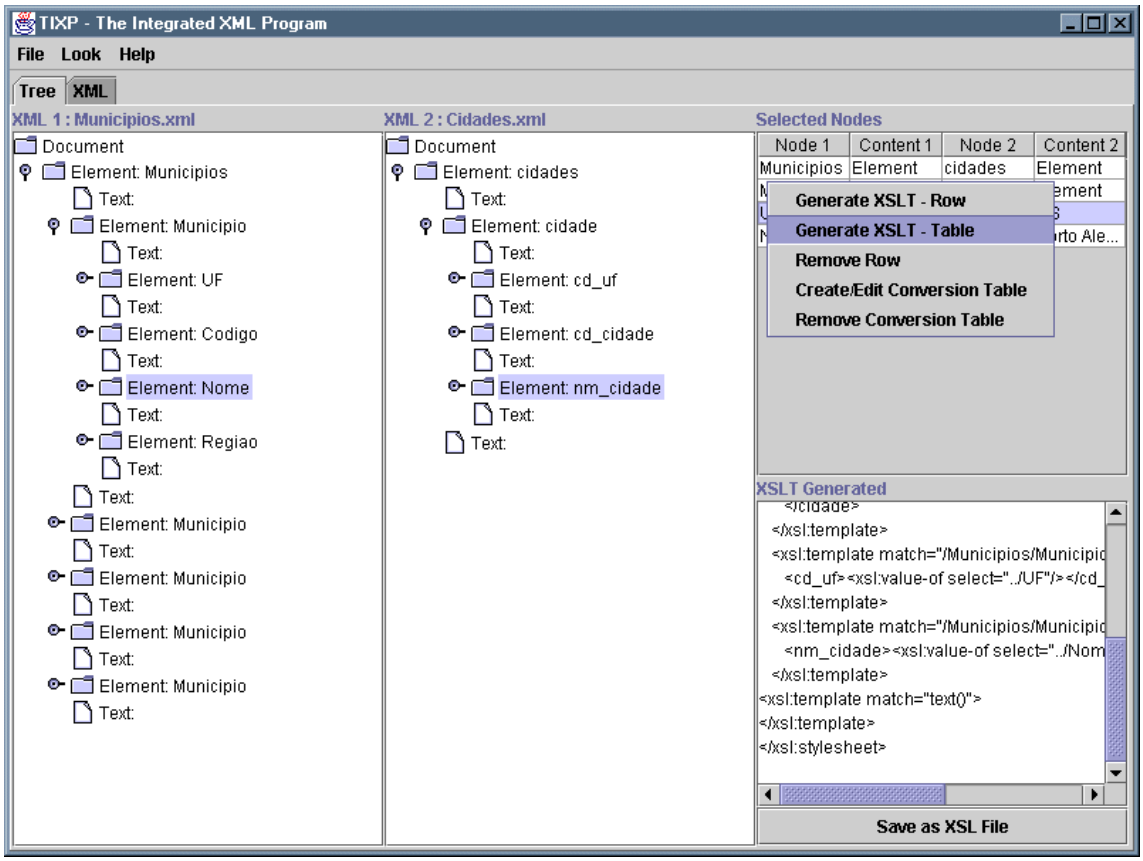

FIGURA 4.27 - Nova geração de *templates* XSL

São gerados então novos *templates*, contemplando a substituição de valores.

Dentro da área realçada em vermelho na figura 4.28, é possível observar o uso de uma construção `<xsl:choose>` para substituir os valores, realizando assim o mapeamento de domínios de dados.

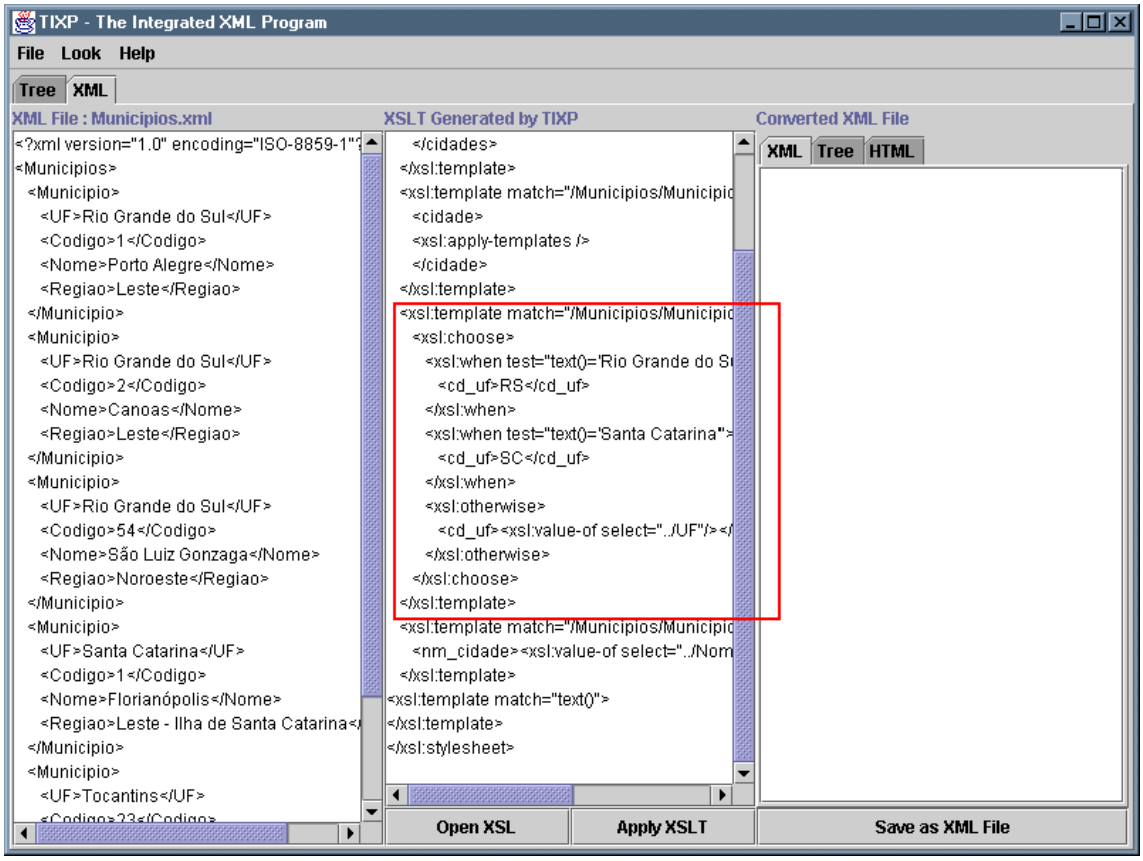

FIGURA 4.28 - Novo *stylesheet* XSL, com mapeamento de domínios

É utilizado um comando da especificação do XSLT para a realização de múltiplas escolhas, o `<xsl:choose>`. O *template* gerado assemelha-se ao abaixo:

```
<xsl:template match="/Municipios/Municipio/UF">
   <xsl:choose>
      <xsl:when test="text()='Rio Grande do Sul'">
         <cd_uf>RS</cd_uf>
      </xsl:when>
      <xsl:when test="text()='Santa Catarina'">
         <cd_uf>SC</cd_uf>
      </xsl:when>
      <xsl:otherwise>
         <cd_uf><xsl:value-of select="../UF"/></cd_uf>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

É possível então aplicar a transformação XSL e verificar o documento resultante, onde as substituições de valores foram realizadas.

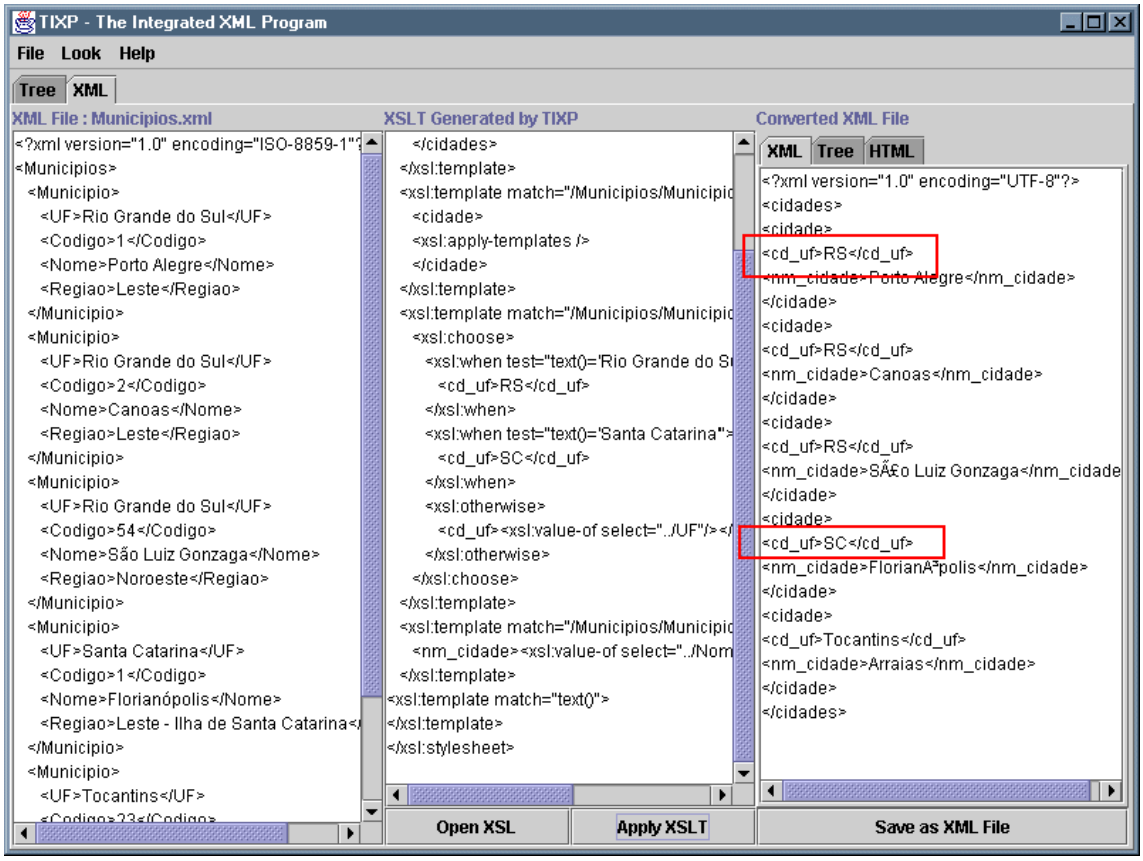

FIGURA 4.29 - Documento resultante com valores substituídos

Da mesma forma como mostrado anteriormente, o documento XML resultante pode ser visualizado tanto em forma textual quanto em forma de árvore hierárquica.

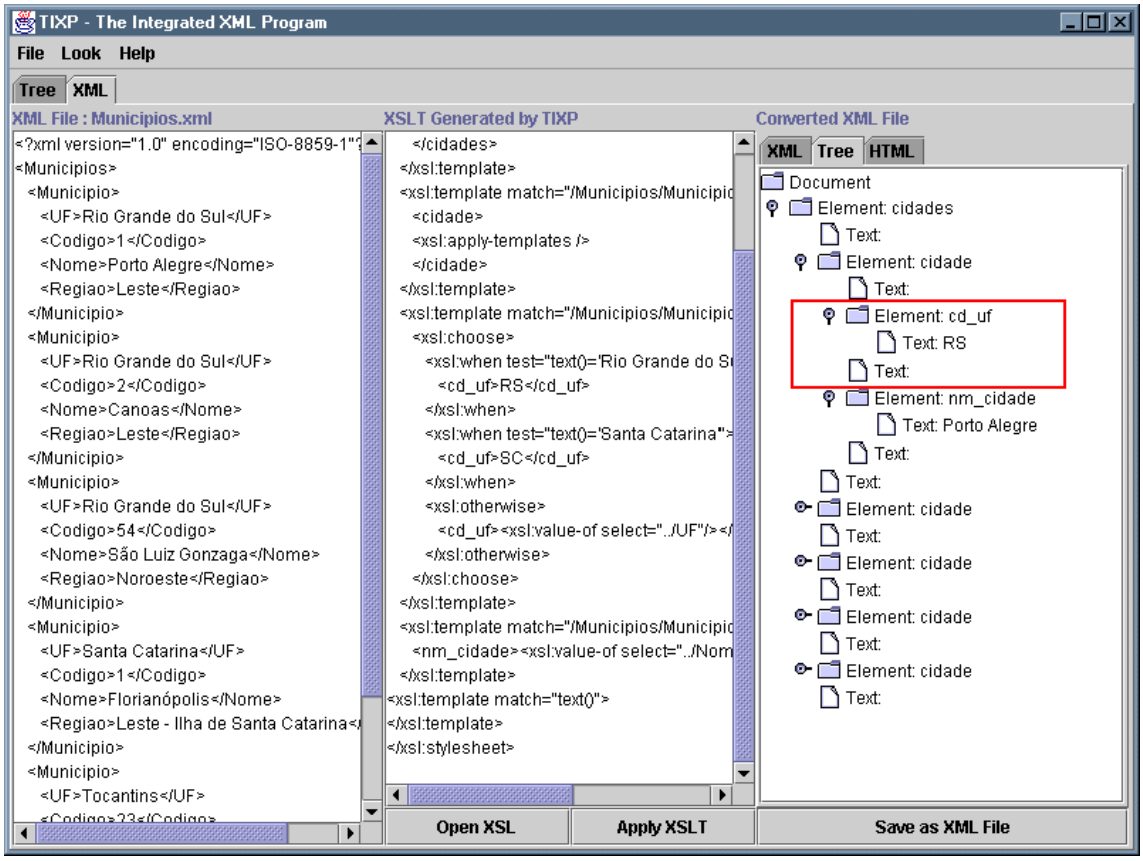

FIGURA 4.30 - Documento XML resultante em forma de árvore hierárquica

O TIXP foi programado para gerar *templates* de substituição somente para os valores informados nas tabelas de conversão, mantendo os valores dos dados nos casos onde estes valores não são referenciados.

Na próxima figura, é possível observar que o valor "Tocantins" para o *tag* `<cd_uf>` foi preservado, pois não estava referenciado na tabela de conversão para aquele elemento.

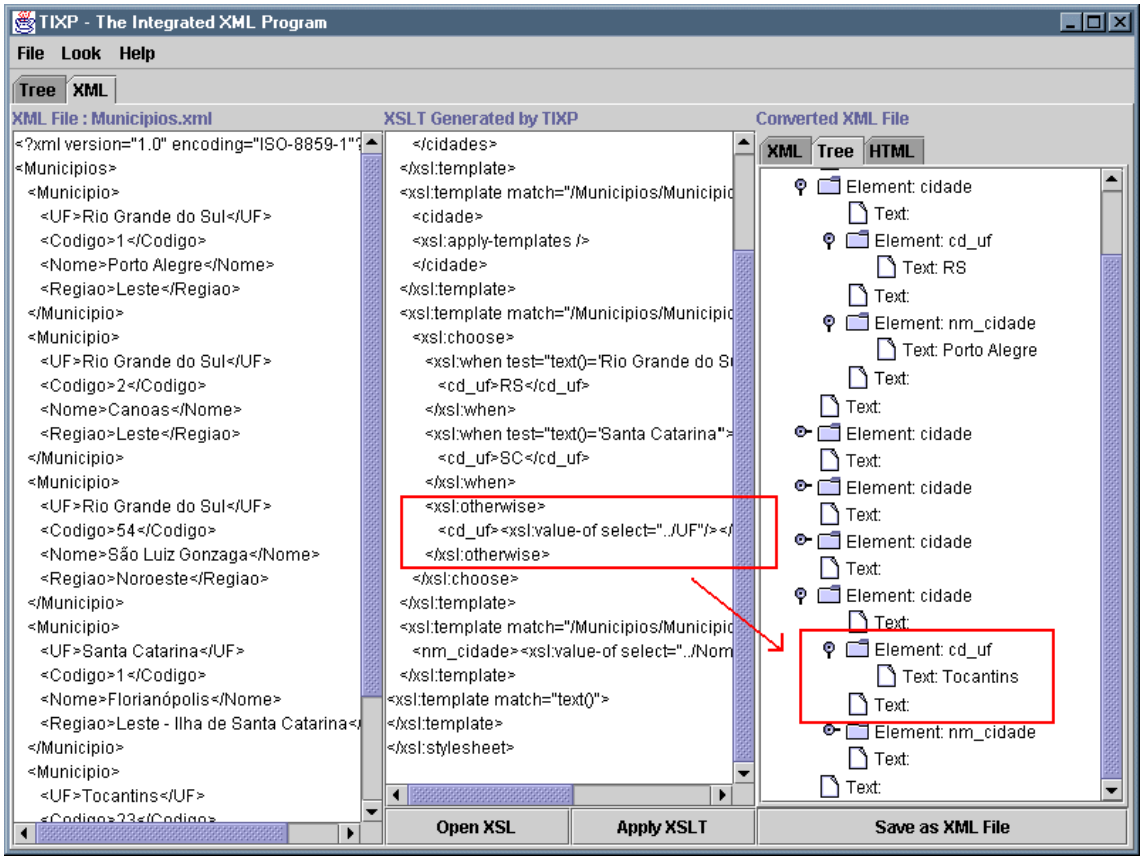

FIGURA 4.31 - Elementos não referenciados não são substituídos

Como foi demonstrado, o TIXP é capaz de gerar *templates* XSL capazes de realizar as duas operações mais necessárias na integração de sistemas heterogêneos: A tradução de estruturas e o mapeamento de valores. Algumas outras características adicionais, necessárias para a maior facilidade de utilização da ferramenta, foram também inseridas, e são apresentadas na seqüência.

Por exemplo, pode-se, durante o desenvolvimento dos *templates*, decidir-se por outro conjunto de arquivos XML, para a geração de um outro XSLT. Para evitar a necessidade de encerrar e reiniciar o programa, foi inserido no menu "File" do TIXP a opção "Restart TIXP", que reinicializa todas as estruturas internas, bem como reestabeece o estado inicial da interface, tornando-a apta ao começo do desenvolvimento de uma nova transformação XSL.

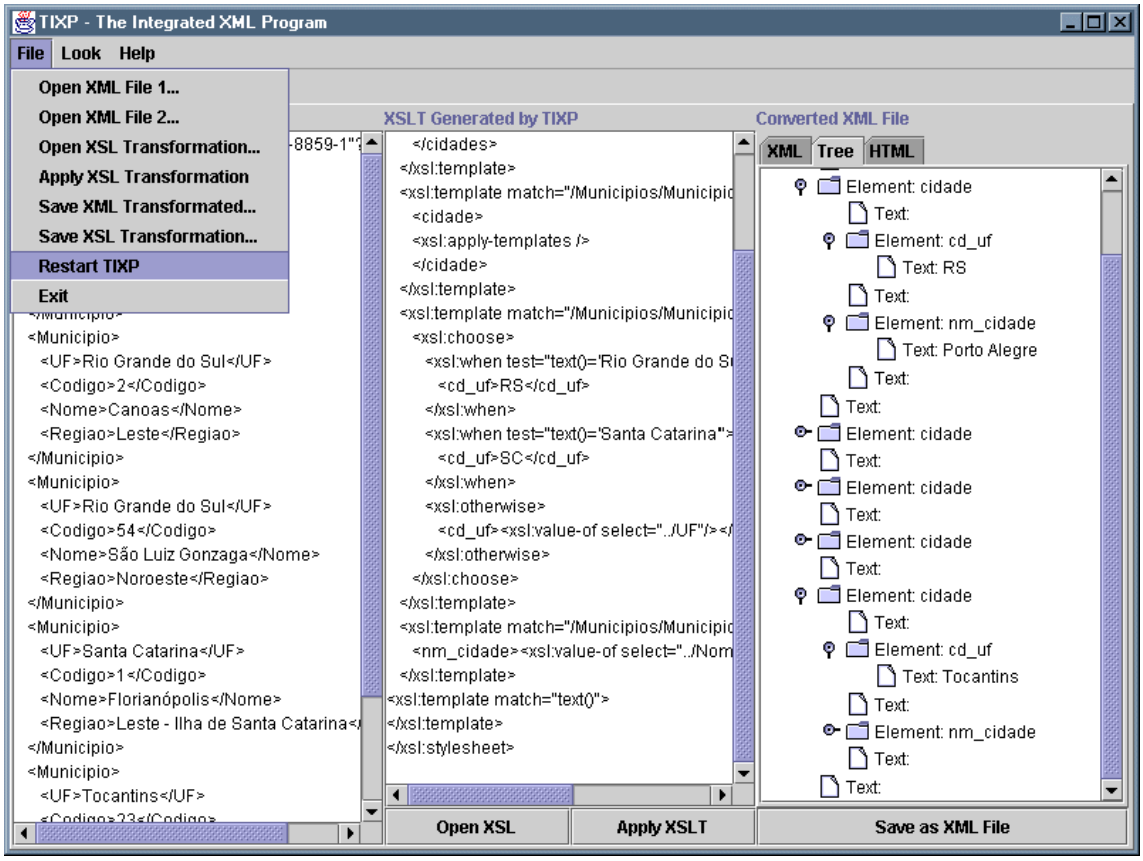

FIGURA 4.32 - A opção "Restart TIXP" do menu "File"

O TIXP volta para seu estado inicial, como demonstrado na figura 4.3.

A última característica funcional inserida no TIXP foi a capacidade de abrir documentos XSL que não tenham sido gerados pelo programa, mas sim em editores de texto ou em softwares de terceiros. Essa capacidade pareceu bastante útil em uma ferramenta que pode ser utilizada não somente para gerar *templates* XSL semi-automaticamente, mas também para a edição direta de documentos XSL. Com essa inclusão, aliada a capacidade de submeter a transformação XSL à documentos XML, o TIXP tornou-se uma ferramenta que pode ser utilizada no aprendizado da especificação XSLT, pois permite testes sobre documentos criados pelo usuário.

Um exemplo disso é demonstrado nas próximas figuras. Um documento XML gerado por uma *stored procedure* desenvolvida para SGBD Sybase pelo autor, representando a estrutura de uma tabela de um banco de dados, é submetido a uma transformação XSL desenvolvida e pré-gravada no próprio TIXP, para gerar um documento HTML contendo um comando DDL de criação da respectiva tabela.

Começa-se o processo pela escolha do documento XML a ser transformado. Para tanto, escolhe-se a opção "Open XML File 1", do menu "File". Note-se que para essa função, só pode ser utilizada a área XML 1 do TIXP.

Para o exemplo, opta-se por um arquivo XML chamado paciente.xml. Seu conteúdo é então demonstrado em forma de árvore hierárquica pelo primeiro *folder* **do** sistema TIXP.

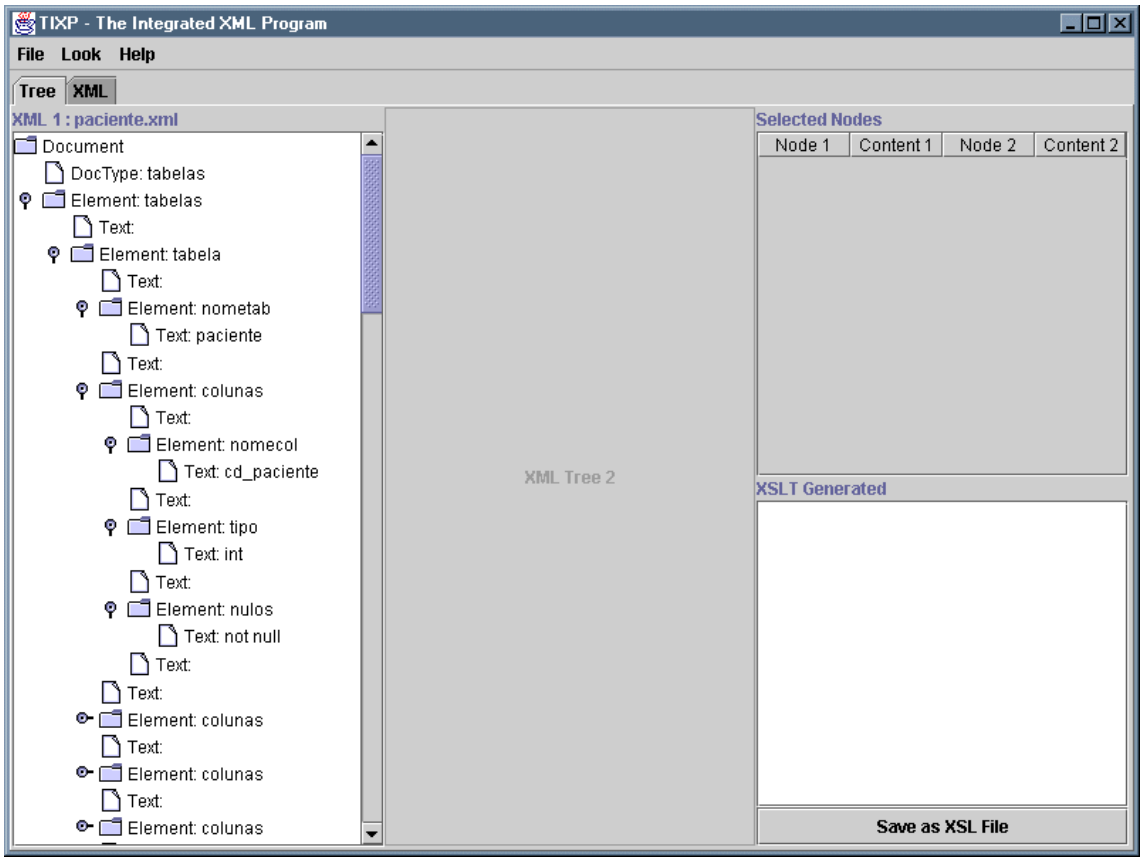

FIGURA 4.33 - A estrutura do documento exemplo paciente.xml

No segundo *folder*, intitulado "XML", o mesmo documento está disponível para visualização no seu formato textual.

O próximo passo é a escolha do documento XSL. Neste exemplo, optou-se por um arquivo capaz de gerar HTML, por dois motivos: O primeiro, porque esta é a forma mais comumente encontrada do XSLT na literatura, o que faz dela bom exemplo didático e prático; segundo, porque serve para demonstrar que o TIXP também é capaz de renderizar documentos HTML gerados por transformação, apresentando-os em sua forma interpretada, em forma textual e na forma de árvore hierárquica.

Escolheremos o arquivo paciente1.xsl para conversão. O sistema TIXP troca automaticamente o *folder* visível para o *folder* XML, pois essa forma de visualização do documento XSL facilita a compreensão de seu conteúdo.

O documento XSL aberto é então demonstrado nos mesmos objetos gráficos utilizados para apresentar os *stylesheets* XSL gerados pelo TIXP. A edição do mesmo é permitida.

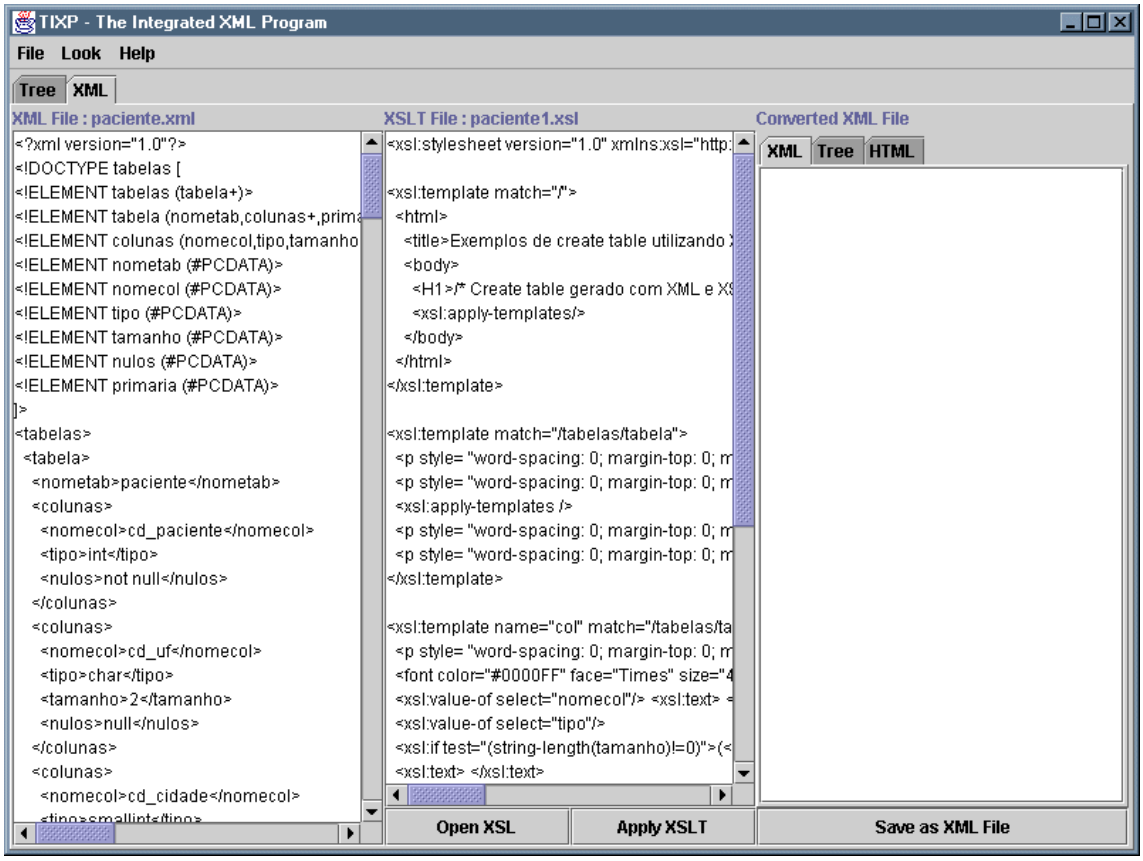

FIGURA 4.34 - Documento XSL aberto pelo TIXP

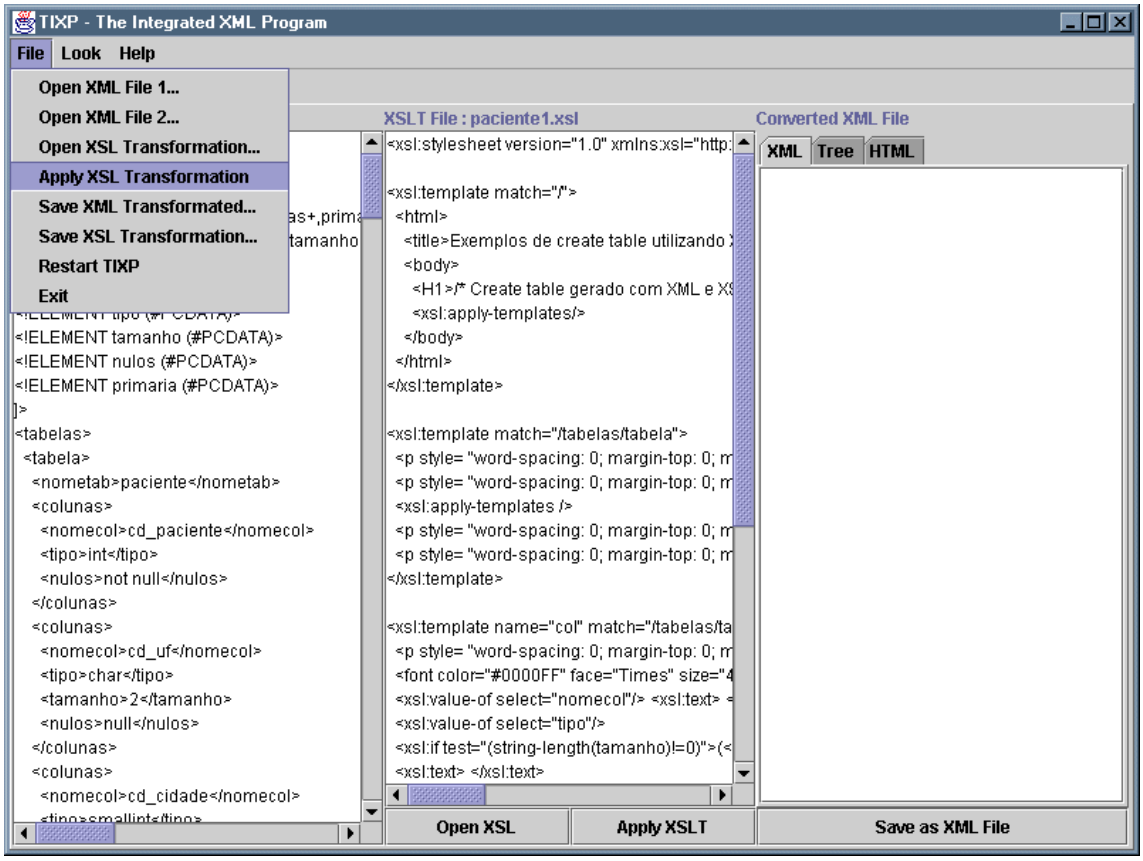

FIGURA 4.35 - A opção "Apply XSL Transformation" do menu "File"

Uma vez aberto o documento XSL, basta submetê-lo ao documento XML, da mesma forma como feito com XSLT's gerados pelo próprio TIXP. Isto pode ser feito através do botão "Apply XSLT", na interface gráfica, ou pelo menu "File", como se observa na próxima figura.

O arquivo convertido é apresentado então em três formas, a textual, a de árvore hierárquica, e a HTML, como pode ser visto nas próximas páginas.

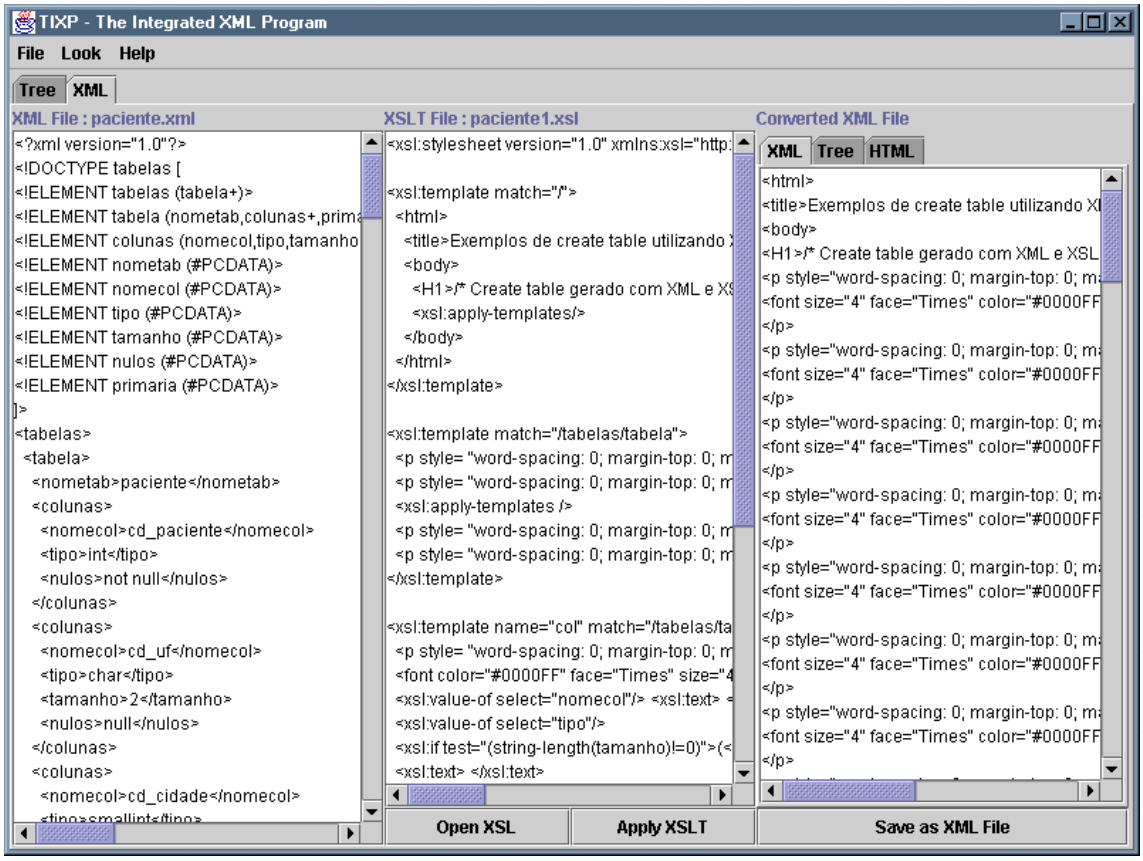

FIGURA 4.36 - O documento transformado em forma textual

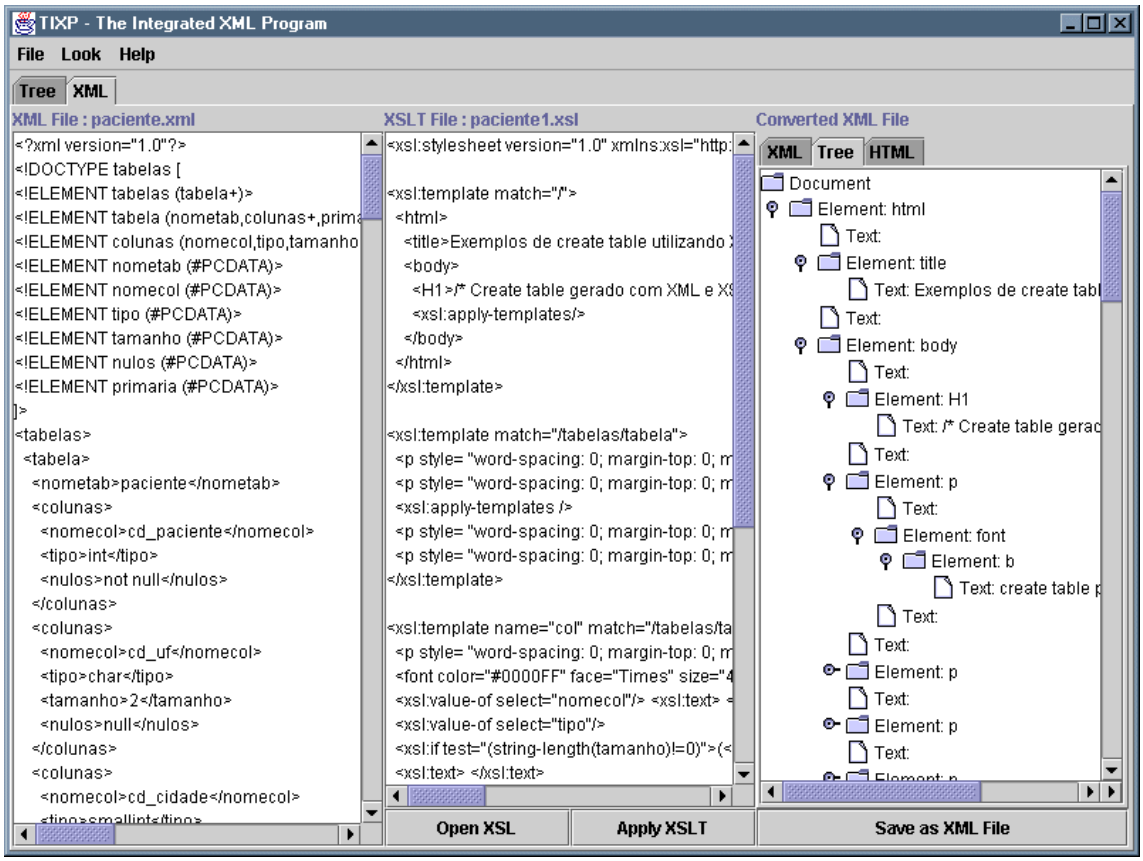

FIGURA 4.37 - O documento transformado em forma de árvore hierárquica

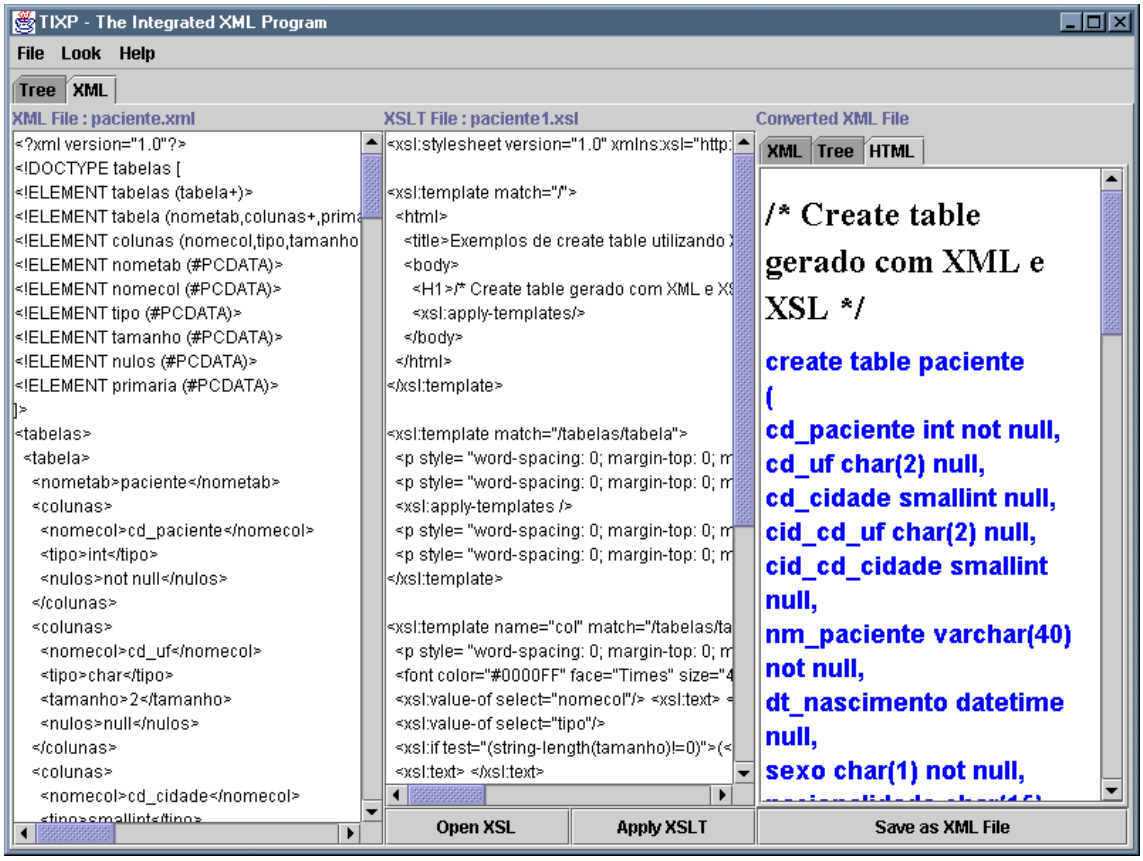

FIGURA 4.38 - A forma HTML interpretada do documento transformado

O TIXP sempre tentará renderizar o documento transformado como HTML. Se o documento resultante da transformação não possuir os *tags* necessários, isso não será possível, e o documento será apresentado sem a devida formatação.

Da mesma forma, só serão representados corretamente como árvores hierárquicas aqueles documentos HTML que tiverem sua estrutura sintática contida na especificação XML. Documentos HTML com construções como, por exemplo, `<META>`, não podem ser demonstrados adequadamente pelo TIXP, uma vez que esse tipo de *tag* só possui o marcador de abertura de seqüência, e não o de fim, o que faz com que o documento seja considerado XML mal-formado.

Uma característica necessária a um programa que se destina a servir de ferramenta de apoio a qualquer tipo de desenvolvimento é a capacidade de rastrear erros durante o próprio desenvolvimento. No TIXP isso foi feito através da implementação de um mecanismo de exposição de erros encontrados durante os processos de *parsing* e de transformação dos documentos XML.

Erros gerados são demonstrados em telas específicamente desenhadas para essa função, para que o usuário interaja com os documentos, corrigindo problemas. Infelizmente, a parte da implementação JAXP responsável pela comunicação de erros durante o processamento de documentos ainda está em fase de implementação, o que faz

com que não seja possível indicar com exatidão o tipo e o local exatos dos erros encontrados, mas apenas aproximações destes.

No documento XSL demonstrado na próxima figura, um erro de sintaxe é permitido de forma deliberada, para que a mensagem de erro seja mostrada.

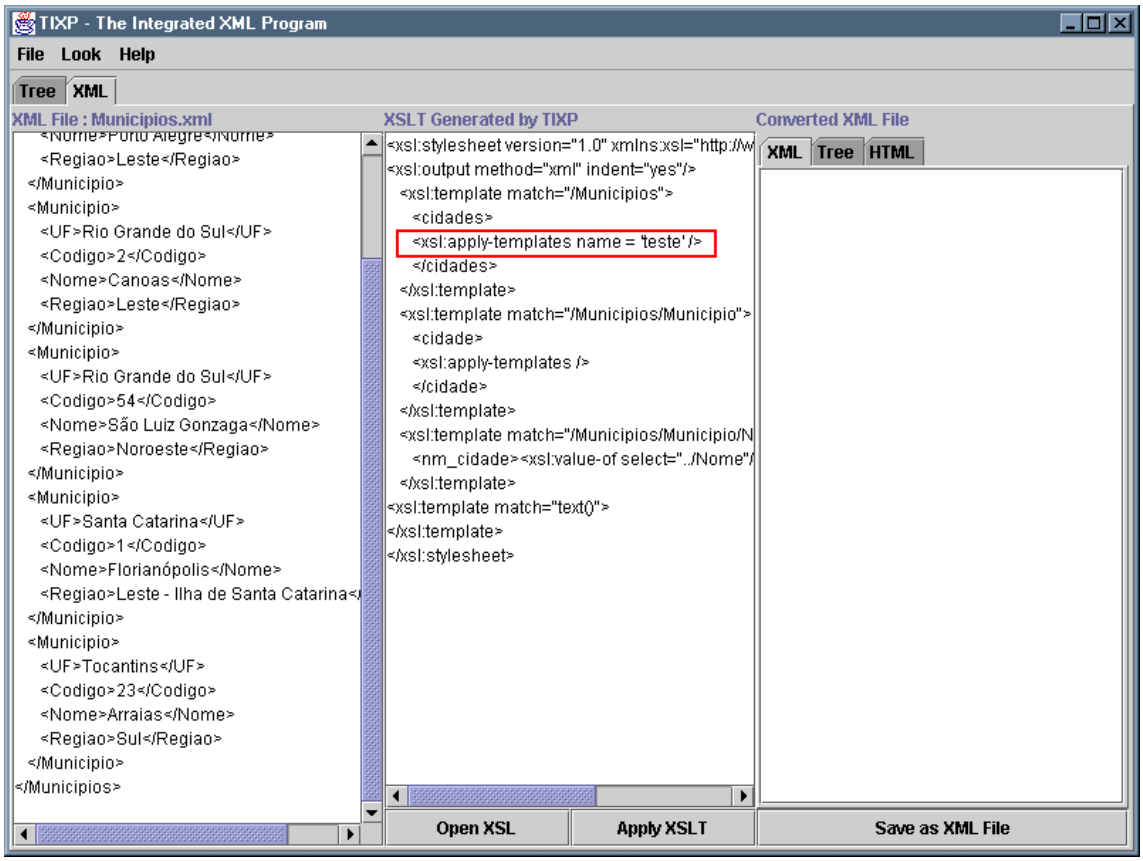

FIGURA 4.39 - *Stylesheet* XSL com erro de sintaxe

Uma tela informa o tipo de erro encontrado, conforme pode ser visto na figura 4.40.

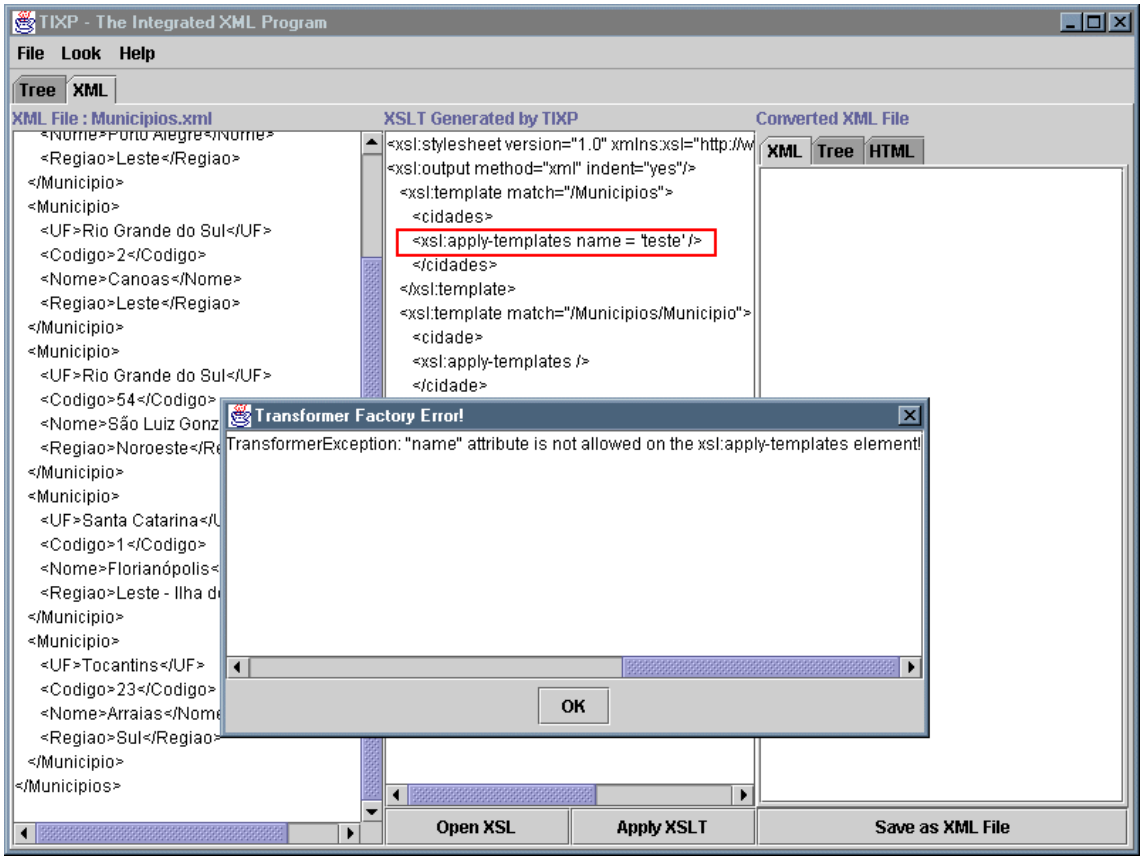

FIGURA 4.40 - Comunicação de erro encontrado no documento XSL

Erros podem ser reportados pelo TIXP em dois momentos: Quando o documento XML é interpretado e tem sua árvore DOM construída, para apresentação como árvore hierárquica, se o documento é mal-formado, ou quando a sintaxe XSLT existente em um documento XSL estiver incorreta.

Como característica adicional, visando a utilização do TIXP em várias plataformas computacionais heterogêneas, assim como os programas que o mesmo pretende integrar, o mesmo foi provido de uma interface capaz de adaptar-se ao sistema operacional onde estiver executando. Isto é visto no menu "Look", onde temos as opções "Metal", que é a forma de apresentação multi-plataforma da biblioteca gráfica Swing, do Java, "CDE/Motif", que é a forma de apresentação de ambientes UNIX com X-Windows, e a opção "Windows", para ambientes com sistemas operacionais da Microsoft.

A forma de apresentação padrão do TIXP é a "Metal", como já apresentado em todas as ilustrações demonstradas até aqui. Através do menu "Look", as outas opções podem ser escolhidas.

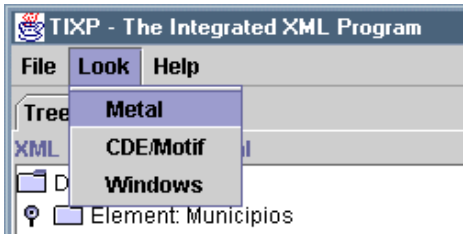

FIGURA 4.41 - O menu "Look"

As formas de apresentação "CDE/Motif" e "Windows" são apresentadas nas figuras 4.42e 4.43, respectivamente.

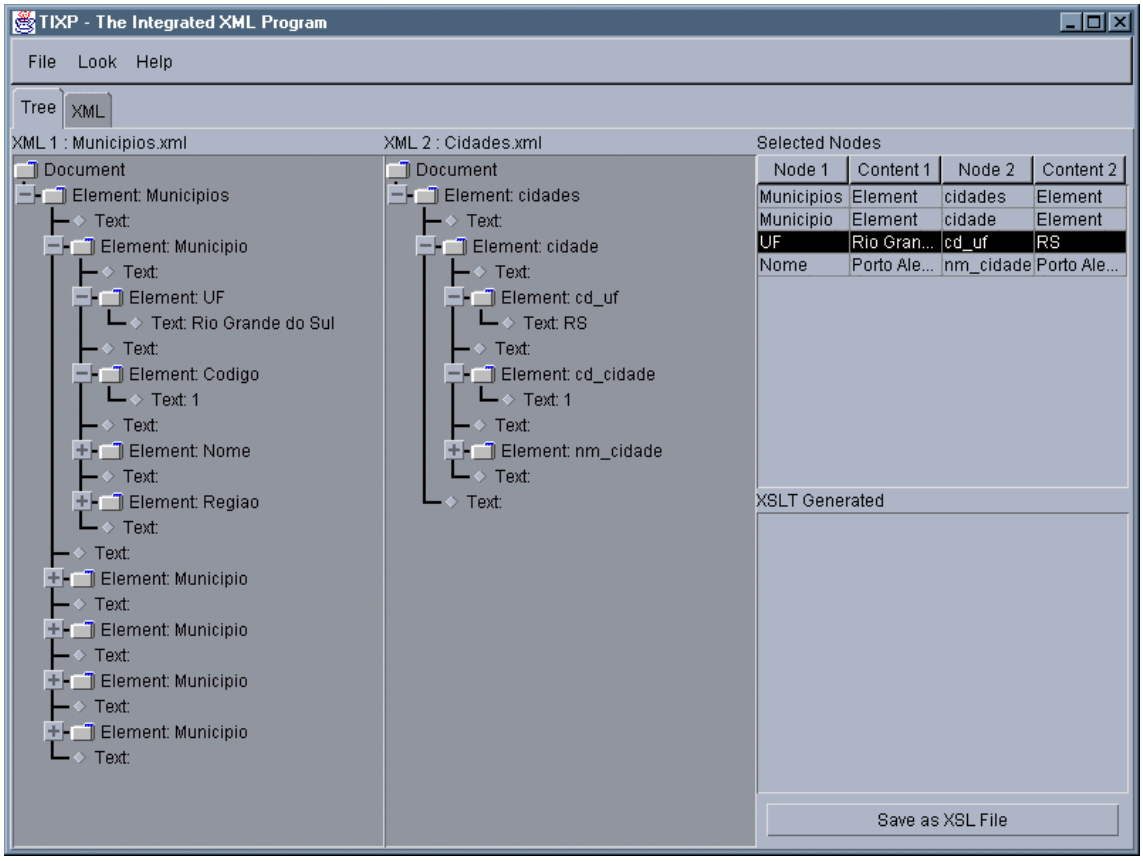

FIGURA 4.42 - A forma de apresentação "CDE/Motif", para UNIX com X-Windows

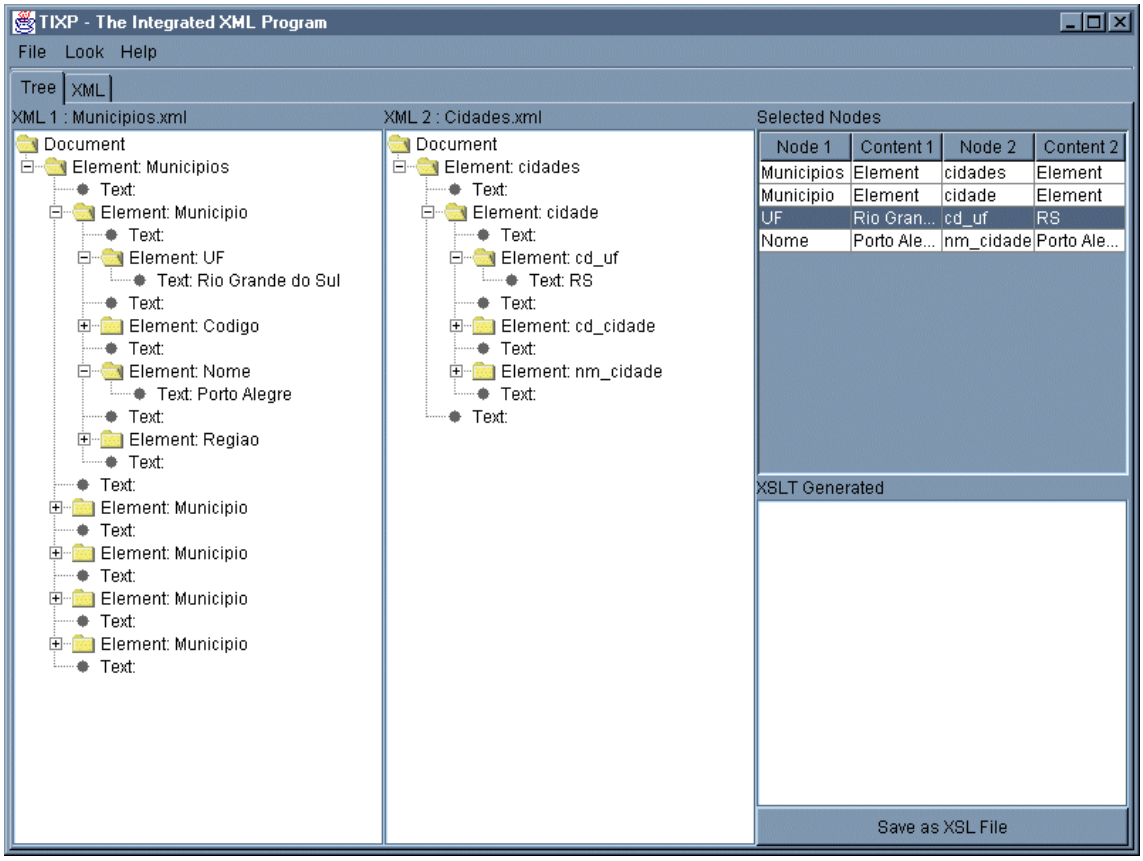

FIGURA 4.43 - A forma de apresentação "Windows" para ambientes Microsoft

Por último, para encerrar o uso do programa, escolhe-se a opção "Exit" do menu "File". Será apresentada uma janela de mensagem solicitando a confirmação da saída do sistema. Pressione o botão "Yes" para sair do programa.

Como foi demonstrado, o sistema TIXP é capaz de gerar *stylesheets* XSL para realizar as transformações necessárias para que documentos XML sejam reconhecidos por aplicações integrantes do *framework* proposto. Assim sendo, TIXP encaixa-se no *framework* como a ferramenta de geração de *stylesheets*, simplificando o processo de integração entre sistemas heterogêneos.

5 Conclusões

Neste trabalho foi realizado um estudo sobre a integração de aplicativos heterogêneos. Foram levantados os principais problemas provenientes da necessidade de interoperação entre softwares, no que diz respeito a grande quantidade de fatores envolvidos, e que podem ser desde diferenças de estilo de modelagem, em ambientes de mesma plataforma, linguagem de programação e SGBD, até diferenças abismais, onde encontram-se sistemas que rodam sobre plataformas diferentes, SGBD's distintos, e paradigmas de modelagem e programação igualmente diversos.

Antes do advento do XML, os dados só poderiam transitar entre diferentes aplicativos através da disponibilização de acessos às bases de dados envolvidas, através de *gateways*, replicadores de dados, sistemas de trocas de mensagens ou conjuntos de exportadores/importadores. Os formatos dos dados trocados teriam que ser combinados entre os representantes das partes envolvidas no processo de integração, formando um protocolo que deveria ser codificado nos programas. Quaisquer alterações, por quaisquer motivos, no formato dos dados trocados, geraria a necessidade de alteração nos softwares.

Nesse contexto, XML, com sua capacidade de adicionar semântica aos dados, através da utilização de *tags* definidos pelo usuário, permite a criação de programas genéricos para importar e exportar dados, utilizando seu padrão, de forma a eliminar a necessidade da definição de protocolos a ser codificados no interior das aplicaçãos.

XML é um padrão universal para troca de informações. Se em um grupo de softwares, todos estes forem capazes de "entender" XML, então possui-se um grupo de softwares onde todos são capazes de interagir com todos. A palavra integração pode ser utilizada no seu mais amplo sentido, com a complexidade algorítmica para solucionar problemas de formatos e diferenças de valores bastante reduzida.

A capacidade de transformar os dados representados em XML, obtida através da utilização de XSL e XSLT, faz de XML um padrão muito flexível e poderoso, que pode ser utilizado para simplificar sistemas a ser integrados, deixando a tarefa da transformação dos dados para um processador XSLT externo aos programas.

Como resultado do estudo realizado, foi proposto aqui um *framework* para a integração de sistemas heterogêneos, utilizando XML, XSL e XSLT como forma de representar os dados e realizar as adequações necessárias aos mesmos, simplificando o processo de implementação de integrações entre softwares, possibilitando assim a redução de tempo de desenvolvimento e custos com pessoal, ideal almejado por todas as instituições contemporâneas.

Para acrescentar ganho a utilização do *framework* de integração, uma ferramenta de apoio a geração de *stylesheets* XSL, para simplificar o processo de elaboração das transformações XSL, através de uma aplicação gráfica capaz de converter mapeamentos realizados em *templates* XSLT, foi desenvolvida, e seu protótipo, já plenamente funcional, foi apresentado também neste trabalho.

Desta forma, as principais contribuições oriundas do desenvolvimento desse projeto foram:

- Um estudo das principais dificuldades encontradas na integração de sistemas, bem como das formas utilizadas para realizar a integração destes.

- O projeto de um *framework* para integração de sistemas, capaz de simplificar o processo de integração de sistemas, reduzindo a necessidade de protocolos rígidos de troca de dados, bem como eliminando a necessidade que esses protocolos sejam programados no interior das aplicações, o que faz com que a necessidade de manutenção dos sistemas, em caso de alteração dos protocolos, não mais exista.

- Uma ferramenta para a geração de *stylesheets* XSL, que serve não somente para gerar os *stylesheets*, mas também para testá-los, servindo como uma base de lançamento, a partir de onde experiências com XSL podem ser tentadas, servindo inclusive como apoio ao aprendizado das tecnologias XML, XSL e XSLT.

5.1 Trabalhos futuros

Como citado anteriormente, esse trabalho foi conduzido de forma a analisar, e endereçar, as dificuldades encontradas na integração de sistemas desenvolvidos para acessar bancos de dados. Existe uma infinitude de outros tipos de sistemas, não apresentados aqui, e que podem se valer das facilidades alcançadas através da utilização do padrão XML, bem como suas tecnologias auxiliares.

Referente à implementação do gerador de *stylesheets*, o mesmo é somente um protótipo, que não está pronto para utilização em escala comercial, podendo receber o acréscimo de funcionalidades que o tornariam uma importante ferramenta, para utilizar em integrações, ou mesmo na produção de documentos XML e XSLT *ad hoc*.

Especificamente quanto à implementação do sistema TIXP, e da utilização do *framework* proposto na construção e/ou interligação softwares , os trabalhos podem evoluir para cobrir alguns dos assuntos a seguir.

A biblioteca JAXP, da Sun, é, como já citado, uma implementação das especificações SAX e DOM, e apresenta todas as características que foram necessárias para a implementação do TIXP, e poderia ser utilizada em escala comercial no desenvolvimento de software aplicativo. Seu único inconveniente é o fato desta ter sido desenvolvida em Java, e portanto só servir para implementações nessa linguagem.

A linguagem Java, por sua vez, apresenta um inconveniente para a produção de aplicativos, que é o elevado nível de exigência de hardware para executar os programas desenvolvidos nela. Suas bibliotecas gráficas ainda são demoradas para carregar e executar, fato este que está sendo trabalhado para a Sun, mas que apesar de melhorias nas últimas versões da linguagem, e da liberação da nova máquina virtual Java (JVM) HotJava, ainda necessitará de tempo para ser resolvido. Enquanto isso não acontece,

programas desenvolvidos em Java só apresentam comportamento satisfatório, no quesito velocidade, em computadores com processadores bastante potentes e com grandes quantidades de memória RAM.

Isto faz com que o desenvolvimento de programas em Java, e portanto utilizando JAXP, não seja uma boa estratégia comercial no momento. Um programa como o TIXP é um software pequeno, desenhado para ser utilizado com uma freqüência baixa, e portanto não apresenta esse problema. Mas para programas utilizados diariamente, Java ainda não parece viável.

Isto leva a necessidade de se pesquisar a implementação de *parsers* e de outros objetos para manipulação de documentos XML compatíveis com a plataforma Windows e as principais linguagens de desenvolvimento de aplicativos para esta, tais como Microsoft Visual Basic, Borland Delphi, Sybase PowerBuilder e C++, para implementar a etapa de leitura e reconhecimento de arquivos XML por parte dos aplicativos.

Dentro do TIXP, implementações futuras poderiam incluir a utilização de JDBC para permitir que o programa comunica-se diretamente à bancos de dados, podendo assim acessar os dados diretamente, exportando-os em XML, utilizando seus catálogos para a geração de XML *Schema* ou mesmo a geração de *stylesheets* diretamente a partir destes. As tabelas de conversão de valores do TIXP poderiam ser armazenadas em bancos de dados, e lidas diretamente por este, o que simplificaria a geração de XSL para documentos com elevado número de mapeamentos de valores.

Bibliografia

[BAL 2000] BALDWIN, Richard. XML for Beginners, Part 1, Structured Documents,Plain Text, and Rendering. **Web Developer Journal at Developer.com**, U.S.A., June 2000. Disponível em: < http://developer.earthweb.com/journal/techworkshop/062600_xml1.html >. Acesso em: 27 jun. 2000.

[BAR 2000] BARROS, Fábio. Touro Indomável**.** In: COMPUTERWORLD Especial Integração de Sistemas. **COMPUTERWORLD**, n. 324, jul. 2000. Disponível em: < http://www.uol.com.br/computerworld/technology/guideline/integ/integ01.htm >. Acesso em: 16 nov. 2001.

[CHU 97] CHUNG, P. Emerald et al. DCOM and CORBA Side by Side, Layer by Layer. New Jersey: Bell Laboratories, 1997. Disponível em: < http://www.cs.wustl.edu/~schmidt/submit/Paper.html >. Acesso em: 12 maio 2000.

[DIC 2001] DICIONÁRIO da Economia Digital. Portugal: **COMPUTERWORLD**, 2001. Disponível em: < http://www.computerworld.iol.pt >. Acesso em: 12 nov. 2001.

[FAY 99] FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. **Building Application Frameworks**: Object-Oriented Foundations of Framework Design. U.S.A.: Wiley Computer Publishing, 1999. p. 4.

[GAM 2000] GAMMA, Erich et al. **Padrões de Projeto**: Soluções Reutilizáveis de Software Orientado a Objetos. Porto Alegre: Bookman, 2000. p. 41.

[GAR 99] GARDNER, Dana. XML promises simple object interoperability. **InfoWorld Electric**, U.S.A., September 27th, 1999. Disponível em: < http://www.infoworld.com/cgi-bin/displayStory.pr?990927.hhxml.html >. Acesso: em 22 maio 2000.

[HAR 99] HAROLD, Elliote Rusty. **The XML Bible.** U.S.A.:Hungry Minds, 1999, chap. 17. Disponível em: < http://metalab.unc.edu/xml/books/bible >. Acesso em: 12 set. 2001.

[IBM 2000] IBM APPLICATION DEVELOPMENT. **XMI - XML Metadata Interchange.** U.S.A., 1998. Disponível em: < http://www-4.ibm.com/softtwares/ad/standards/xmi.html >. Acesso em: 22 maio 2000.

[KIE 99] KIELY, Don. XML: More Than Just a Quick Fix. **Information Week,** U.S.A., February 8th, 1999. Disponível em: < http://www.informationweek.com/720/iuxml >. Acesso em: 13 out 2000.

[LAD 2000] LADDAD, Ramnivas. XML API´s for databases. **JavaWorld**, U.S.A., Jan. 2000. Disponível em: < http://www.javaworld.com/javaworld/jw-01-2000/jw-01-dbxml-p.html >. Acesso em: 20 jun. 2000.

[MEG 2000] MEGGINSON, David. **SAX 2.0**: The Simple API´s for XML. U.S.A., 2000. Disponível em: < http://www.megginson.com/SAX >. Acesso em: 19 jun. 2000.

[SOA 95] SOARES, Luiz Fernando Gomes; LEMOS, Guido; COLCHER, Sérgio. **Redes de Computadores**: Das LANs, MANs e WANs às Redes ATM. Rio de Janeiro: Campus, 1995. p. 139.

[SPE 2000] SPELL, Brett. **Professional Java Programming.** Birmingham: Wrox Press, 2000.

[SUN 2001] SUN MICROSYSTEMS, **Introduction to XSLT.** Palo Alto, 2001. Disponível em: < http://java.sun.com/xml >. Acesso em: 7 ago 2001.

[UEY 2000] UEYAMA, Jó; MADEIRA, Edmundo. A Aplicação de Catálogos XML no Comércio Eletrônico. **Developers' Magazine**, Brasil, p. 20-27, jun. 2000.

[W3C-1 98] W3C, World Wide Web Consortium. **Document Object Model (DOM) Level 1 Specification, Version 1.0**. U.S.A., October 1st, 1998.W3C Recommendation. Disponível em: < http://www.w3.org/TR/REC-DOM-Level-1-19981001.pdf >. Acesso em: 19 jun. 2000.

[W3C-2 98] W3C, World Wide Web Consortium. **Extensible Markup Language (XML 1.0)**. U.S.A., February 10th, 1998. W3C Recommendation. Disponível em: < http://www.w3.org/TR/REC-xml-19980210.html >. Acesso em: 2 jul. 2000.

[W3C-1 99] W3C, World Wide Web Consortium. **XML Path Language (Xpath) Version 1.0.** U.S.A., November 16th, 1999. W3C Recommendation. Disponível em: < http://www.w3.org/TR/REC-xpath-19991116.html >. Acesso em: 21 ago. 2001.

[W3C-2 99**]** W3C, World Wide Web Consortium. **XSL Transformations (XSLT) Version 1.0.** U.S.A., November 16th, 1999. W3C Recommendation. Disponível em: < http://www.w3.org/TR/1999/REC-xslt-19991116.html >. Acesso em: 26 jul. 2000.

[W3C 2000] W3C, World Wide Web Consortium. **Extensible Stylesheet Language (XSL) Version 1.0.** U.S.A., March 27th, 2000. W3C Working Draft. Disponível em: < http://www.w3.org/TR/2000/WD-xsl-20000327.html >. Acesso em: 19 jun. 2000.