Capítulo

1

Aplicando transformações em XML usando XSLT e XSL-FO (Formatting Objects)

Vivian Genaro Motti, Rudinei Goularte e Maria da Graça Campos Pimentel

Abstract

This chapter presents XML concepts and examples of possible transformations for web documents, using XPATH, XSLT and specially XSL-FO. XML is an extensible language to create structured documents. XPath expressions access and select XML elements and attributes which can be exhibited using XSLT or XSL-FO. XSLT transforms XML documents manipulating, sorting, testing its elements, and producing an XML tree of specific nodes. XSL - Formatting Objects are used to structure documents for printing, layout settings, web browsers rendering, audio files creation. With FO, tables, images and graphics can be inserted, web sites can be formated defining its colors, fonts and alignment.

Resumo

Este capítulo apresenta conceitos de XML e exemplifica as transformações possíveis em documentos web, com XPATH, XSLT e principalmente XSL-FO. XML é uma linguagem extensível para criar documentos estruturados. Expressões em XPATH acessam e selecionam elementos e atributos XML e para exibi-los são usadas XSLT ou XSL-FO. XSLT transforma documentos XML manipulando seus elementos, ordenando-os, testando-os, e produzindo, a partir de regras, uma árvore XML com nós específicos.

XSL - Formatting Objects são usados para estruturar documentos para impressão, layout, renderização por navegador web, criação de arquivos em áudio. Com FO pode-se inserir tabelas, imagens e gráficos, formatar páginas definindo suas cores, fontes e tabulação.

1.1. Introdução

Este capítulo tem por objetivo apresentar conceitos fundamentais sobre a linguagem XML *(Extensible Markup Language)* e suas características básicas, como sintaxe, estrutura, suas vantagens e aplicações. Também serão apresentadas as linguagens para transformação de documentos XML, usando XSLT *(Extensible Stylesheet Language Transformations)* e XSL-FO *(Estensible Stylesheet Language – Formatting Objects)*.

Exemplos práticos ilustram como os elementos e atributos das linguagens devem ser utilizados e quais os resultados obtidos com a aplicação deles. Este capítulo está organizado da seguinte maneira: a Seção 1.2 descreve informações sobre a XML; a Seção 1.3 apresenta a definição de XSL; a Seção 1.4 define o que é e quais são os objetivos de se utilizar XPath; a Seção 1.5 trata sobre XSLT; e a Seção 1.6 define e ilustra a utilização de XSL-FO.

1.2. Linguagem XML (*Extensible Markup Language*)

A SGML, ou *Standard Generalized Markup Language*, é uma linguagem que foi publicada em 1986 como padrão internacional (ISO 8879) e permite a geração de diferentes linguagens de marcação para documentos [Goldfarb and Prescod, 1998]. Uma linguagem de marcação se caracteriza por um conjunto de regras que definem e limitam como as marcações devem ser usadas. Ela é composta por símbolos de marcação e visa o compartilhamento de documentos para que estes possam ser lidos por máquinas e tecnologias diferentes em projetos grandes. Dentre suas principais vantagens, pode-se destacar a portabilidade, o intercâmbio de informações, a integridade e o controle dos dados [Spencer, 1999].

Por SGML ser uma linguagem complexa surgiram linguagens derivadas dela, como a HTML, que é uma aplicação e implementação da SGML, e a XML, que é um subconjunto da SGML. Na HTML as regras indicam que informações adicionais sobre os conteúdos devem ser incluídas dentro das *tags*, que são delimitadas por símbolos de menor e maior, respectivamente (por exemplo: `<tag>`). A maioria das *tags* da HTML está associada com a apresentação de conteúdo. Além disso a HTML é uma linguagem simples e poderosa, mas seu conjunto de *tags* é pré-definido, e portanto, ela é limitada [Spencer, 1999].

A XML surgiu na tentativa de simplificar a linguagem SGML para aplicações em geral. XML é uma linguagem extensível, ou seja, suas *tags* não são pré-definidas, ela é considerada uma metalinguagem. Spencer (1999) afirma que XML é um *framework* para criação de linguagens de marcação, que podem, por sua vez, serem específicas para contextos de aplicações, como em tecnologia ou indústrias.

SMIL (*Synchronized Multimedia Integration Language*) é um exemplo de linguagem criada a partir da XML. Ela é utilizada para sincronizar fluxos de mídia da Internet, e com ela é possível definir, por exemplo, que um som deve ser executado um determinado intervalo de tempo após o início da execução de um vídeo [SMIL, 1998].

XML permite definir *tags* próprias e assim atribuir informações aos conteúdos conforme se deseja, o que possibilita o compartilhamento dos dados. Seus documentos possuem *tags* e dados, suas *tags* podem conter atributos e os nomes das *tags* possuem significado sobre o contexto da informação, fornecendo detalhes sobre o conteúdo. As *tags* não têm por objetivo especificar como o conteúdo será apresentado na interface [Spencer, 1999].

1.2.1. Vantagens da XML

Considerando a XML uma metalinguagem exclusiva para marcação de dados, é fundamental destacar que as configurações de estilo necessárias para apresentar a informação adequadamente não estão incluídas no conjunto de *tags* de conteúdo, ou seja, há a separação entre informações de conteúdo e de apresentação. Essa é uma

vantagem importante, principalmente para gerar diferentes *layouts* a partir de um único conteúdo. Embora a XML tenha por objetivo o uso na Internet, usando folhas de estilo[1] e outras tecnologias (como XSLT, XSL-FO ou CSS) é possível gerar formatos de apresentação para diferentes contextos (por exemplo: livros, apresentações, cartazes e vídeos).

Além disso, é possível obter informações de diferentes fontes e padronizá-las usando XML. No contexto de uma empresa com diferentes clientes que fornecem os mesmos tipos de dados, os dados podem ser apresentados usando um formato comum, provido por um documento XML que trate as diferenças nos estilos dos conteúdos, por exemplo [Spencer, 1999].

Alguns benefícios da linguagem estão associados à realização de buscas na Internet. Nas consultas, termos genéricos tendem a retornar muitos resultados, enquanto que consultas muito específicas não retornam nada. Usando XML para definir as *tags*, há informações de contexto associadas ao conteúdo de modo que os resultados das buscas possam ser mais precisos consultando documentos XML se comparados a buscas em arquivos de texto. Em síntese, são quatro os principais benefícios da XML na Internet:

- Páginas mais consistentes e fáceis de manter, devido à separação entre estilo e conteúdo;
- Facilidade para publicação de banco de dados;
- Melhor busca e associação entre páginas e
- Menos processamento no servidor.

1.2.2. Estrutura

Um documento XML é constituído por *tags* intercaladas por conteúdo. A primeira linha do documento consiste na declaração da linguagem e sua respectiva versão, conforme apresentado no seguinte exemplo de código:

```
<? xml version="1.0" ?>
<agendas>
 <agenda>
  <nome>Maria</nome>
  <endereco>Rua Silva, 4</endereco>
  <telefone>123</telefone>
 </agenda>
</agendas>
```

A XML, por ser desenvolvida para leitura humana, é considerada verbosa e seus *parsers*[2] verificam apenas sua sintaxe. As *tags* utilizadas para o conteúdo podem conter atributos modificadores, como no exemplo acima, telefone pode receber o atributo tipo, definindo se ele é residencial, celular ou comercial.

```
<telefone tipo='celular'>123</telefone>
```

---

[1] Folhas de Estilo são usadas para declarar a formatação (cor, fonte, tamanho) de seções específicas em um conteúdo.

[2] Parser é um programa que realiza análise da estrutura sintática de um documento.

1.2.3. Definição

Pode-se definir a XML como uma linguagem subconjunto da SGML, mas com menor complexidade. SGML é uma metalinguagem designada para definir marcação para diversos tipos de documentos [Pitts-Moultis and Kirk, 2000]. Marcação consiste na adição de *tags* aos dados atribuindo a eles significados, que são úteis no processamento do documento por uma aplicação.

Um documento XML é uma combinação entre marcação e dados, sendo que a marcação possui significado específico e é inserida por *tags* ou elementos XML enquanto os dados são o conteúdo.

XML é uma versão menor da SGML, portanto um subconjunto e uma metalinguagem, já a HTML obedece aos requisitos da SGML e é uma implementação dela. Algumas implementações da XML, consequentemente também da SGML, são: MML, CML, SMIL, para aplicações matemáticas, químicas ou multimídia, respectivamente. A Figura 1.2.3.1 ilustra a relação entre implementação e subconjunto

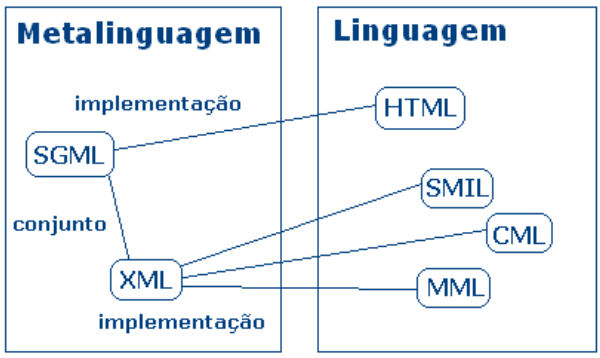

Figura 1.2.3. 1: Diagrama SGML [Spencer, 1999].

das linguagens.

XML pode ser adotada como um padrão robusto pronto para ser usado. No entanto seu principal problema está no DTD *(Document Type Definition),* um documento para descrever a estrutura da sintaxe do XML. Para elaborá-lo há uma sintaxe própria, ou seja, duas sintaxes são necessárias, a das *tags* XML e a do DTD.

1.2.4. Documentos Bem-Formados

XML é uma metalinguagem, portanto o usuário deve definir as *tags* adequadas a sua aplicação. Um par de *tags* define o elemento raiz do documento. Todos os outros pares de *tags* devem estar aninhados ao par raiz e qualquer subelemento deve estar corretamente aninhado com os elementos pais, de modo que se forme uma estrutura de dados hierárquica no documento como uma árvore. Os elementos são formados por pares de *tags* pois a primeira abre a estrutura e a segunda fecha. Há uma construção de *tag* vazia, formada pela inserção da barra (/) após o nome da *tag*, de modo que ao se usar `<nome />` é equivalente ao uso de `<nome></nome>` sem conteúdo inserido entre as *tags*. O espaço em branco antes da barra é opcional. *Tags* sem pares, como a que pode ser utilizada em HTML para definir um parágrafo sem inserir ao final do conteúdo, não são permitidas em XML.

Todos os nomes de *tags* e atributos em XML devem começar por um caractere alfabético ou um *underline* (_) e não devem conter espaço em branco. Caracteres de

pontuação são permitidos, com exceção dos dois pontos (:) reservados a referências de *namespaces* (essa característica auxilia a manter os nomes das *tags* únicos).

Os nomes das *tags* não devem ser iniciados por "xml" (e mesmo combinações de maiúsculas e minúsculas não são permitidas) porque XML é um nome reservado à linguagem. Diferentemente de HTML, os nomes das *tags* em XML são sensíveis à caixa, portanto *tags* como `<nome>`, `<Nome>` e `<NOME>` são diferentes [Spencer, 1999].

Um documento é considerado bem-formado se obedece a certas recomendações da XML. Ele deve possuir um único elemento raiz e todos os outros elementos devem estar corretamente aninhados a ele. Se um documento está bem-formado ele pode ser adequadamente lido por um programa de *parser*. Caso contrário o documento não é considerado XML e os programas que o tratam como tal relatarão erros e não continuarão o processamento, exceto para relatar mais erros [Spencer, 1999].

Para certificar que um documento é bem-formado deve-se verificar se há a declaração de linguagem XML no início, se há um só elemento raiz, se todas as *tags* estão sendo abertas e fechadas, se todas as *tags* têm o mesmo tipo de caixa (alta ou baixa), se todos os elementos estão devidamente aninhados, se os valores dos atributos estão entre aspas e se os caracteres especiais estão sendo tratados com o uso de entidades [XML, 2009].

1.2.5. Tags

A primeira *tag* de um documento XML corresponde à *tag* de abertura do elemento principal, o raiz, que deve ser único em todo o documento. Como todos os elementos, o raiz deve possuir sua *tag* de fechamento, localizada ao final do documento XML. O elemento que sucede ao raiz (ou que é interno a ele) corresponde ao elemento filho (assim como o elemento raiz corresponde ao elemento pai deste). Essas são relações importantes em XML. Todos os elementos, exceto o raiz, devem possuir um único elemento pai e todo elemento pode ter zero ou mais filhos.

Os elementos podem possuir atributos para melhor defini-los, mas é um desafio definir quando devem ser usados atributos ou elementos em um documento. Em geral atributos são mais específicos, detalhando o elemento, e elementos são usados para conteúdos mais abstratos. Outra abordagem também utilizada para se escolher entre atributo e elemento é a necessidade do documento XML ser mais legível por humanos ou por máquinas, ou sendo o uso de mais elementos para ser mais legível por humanos e atributos para ser mais por máquinas. No exemplo a seguir, o atributo moeda especifica o elemento preço:

```
<preco moeda="R$">40,00</preco>
```

Os valores dos atributos sempre devem estar entre aspas (simples ou duplas).

XML possui diferentes tipos de *tags* que podem ser usadas em documentos. Um elemento é simplesmente uma entidade iniciada e terminada por uma *tag*. Um elemento pode ser vazio, possuir conteúdo ou possuir um ou mais filhos. Além dos elementos, um documento XML pode ter outros componentes:

- instruções de processamento para passar instruções específicas de aplicações (por exemplo: `<?xml version="1.0" encoding="UTF-8"?>`);

● seções CDATA que permitem que conteúdo não XML seja incluído nos documentos (por exemplo, para inserir conteúdo com caracteres reservados[3], como o &, deve-se incluí-lo como CDATA: `<nome><![CDATA[C&A]]></nome>`);

● referências a entidades que permitem substituição das referências por outros textos ou informação XML, elas são iniciadas por um símbolo de & e terminadas por ponto-e-vírgula, conforme apresenta a Tabela 1.2.5.1;

● referências a caracteres que permitem incluir muitos caracteres em um documento XML, especialmente para caracteres internacionais ou caracteres reservados também, por exemplo `&#163;` que representa o símbolo £, e

● comentários, como por exemplo: `<!-- comentário -->` sendo que o comentário em si não deve conter dois hífens seguidos para que o *parser* consiga interpretá-lo corretamente (sem confundir os sinais com a *tag* de fechamento).

Tabela 1.2.5.1: Referências a Entidades [Spencer, 1999].

Caractere	Referência
<	&lt;
>	&gt;
&	&amp;
'	&apos;
“”	&quot;

1.2.6. Documentos Válidos

Para que um documento XML seja válido ele deve fazer referência a um DTD (*Document Type Definition*) ou a um XML Schema. Esses documentos têm várias funções, como por exemplo colocar restrições nos conteúdos dos documentos ou definir os nomes das *tags* que serão usadas no documento XML e a ordem na qual elas deverão ocorrer [Goldfarb and Prescod, 1998].

O conceito de documento válido se refere ao documento XML que possui referência a um DTD ou a um XML Schema e está de acordo com as definições contidas nesse.

1.2.7. DTD *(Document Type Definition)*

O DTD inicia com a declaração de que é um documento XML com sua respectiva versão, e segue com a definição do tipo de documento pela referência ao seu elemento raiz. No exemplo abaixo o elemento raiz é *agendas*.

```
<? xml version=”1.0” ?>
<!DOCTYPE agendas [
... declarações ...
]>
```

A sentença DOCTYPE se refere à declaração do tipo de documento (*Document Type Declaration)* e junto com o conteúdo entre colchetes consiste na definição de tipo

[3] '&', '<' e '>' são os caracteres reservados do XML.

de documento *(Document Type Definition)* que define a gramática do documento. Em resumo o documento todo consiste no DTD [Spencer, 1999].

O nome do DOCTYPE sempre deve corresponder ao elemento raiz (`<agendas>` nesse caso). O elemento raiz também deve aparecer na declaração dos elementos, que define quais elementos o documento XML pode conter. No exemplo apenas um elemento está definido, o agendas.

```
<!ELEMENT agendas (agenda)*>
```

O asterisco indica que pode haver um ou mais elementos agenda no documento. O conjunto completo de declaração de todos os elementos se denomina **modelo de conteúdo** do elemento. Para elementos raízes o modelo de conteúdo consiste em todas as declarações de todos os elementos aninhados ao elemento raiz.

```
<!ELEMENT agendas (agenda, nome, telefone+, enderecos, email?)>
```

Esse exemplo mostra que *agendas* é composto pelos elementos: *agenda*, *nome*, *telefone*, *endereço* e *email*, sendo que o sinal de mais (+) indica que pode haver um ou mais telefones, o sinal de interrogação (?) indica que o elemento é opcional e as vírgulas (,) indicam que os elementos devem obedecer estritamente àquela ordem.

Para que o elemento *endereço* seja composto por outros elementos *endereço* aninhados a ele, a seguinte declaração deve ser feita:

```
<!ELEMENT enderecos (endereco+)>
```

E para que o elemento *nome* seja composto apenas por texto, deve ser usado o tipo PCDATA, conforme exemplo abaixo:

```
<!ELEMENT nome (#PCDATA)>
```

PCDATA indica *parser character data* que define que o elemento possui caracteres normais a serem processados pelo *parser*. O símbolo # indica que PCDATA é uma palavra reservada do XML, esse símbolo, portanto, não deve ser usado para nomes de elementos [Spencer, 1999].

O elemento *telefone* também é do tipo PCDATA, mas ele contém o atributo de nome 'tipo', que também contém dados em caracteres (CDATA) e que é necessário como parte do elemento. O uso de CDATA indica que o *parser* não deve interpretar o texto desse campo.

```
<!ELEMENT telefone (#PCDATA)>
<!ATTLIST telefone tipo #CDATA #REQUIRED>
```

Unindo a definição contida no DTD aos elementos declarados no documento XML obtém-se um documento completo, válido e bem-formado.

```
<? xml version="1.0" ?>
<!DOCTYPE agendas [
 <!ELEMENT agendas (agenda)*>
 <!ELEMENT agenda (nome, telefone+, enderecos, email?)>
 <!ELEMENT enderecos (endereco+)>
 <!ELEMENT nome (#PCDATA)>
 <!ELEMENT telefone (#PCDATA)>
 <!ELEMENT endereco (#PCDATA)>
 <!ELEMENT email (#PCDATA)>
 <!ATTLIST telefone tipo CDATA #REQUIRED>
]>
```

```
<agendas>
 <agenda>
  <nome>Maria</nome>
  <endereco>Rua Silva, 4</endereco>
  <telefone tipo="residencial">123</telefone>
  <telefone tipo="celular">971</telefone>
 </agenda>
</agendas>
```

O DTD pode estar no documento XML ou ser apenas referenciado nele. Referências são mais apropriadas quando o DTD é compartilhado entre documentos XML. No entanto é possível combinar DTDs internos e externos, sendo que as definições internas ao documento XML têm prioridade maior caso haja um conflito. Esse recurso é particularmente útil quando um DTD preenche a maioria dos requisitos para um conjunto de documentos XML [Spencer, 1999].

Para que o DTD seja externo ao documento, todo o conteúdo entre colchetes deve ser inserido em um novo arquivo (chamado, por exemplo, de validacao.dtd) armazenado no mesmo diretório do documento XML, que passa a ter a seguinte referência na linha do DOCTYPE:

```
<!DOCTYPE agendas SYSTEM "validacao.dtd">
```

O *parser* de validação processa os documentos do mesmo modo que no documento anterior com o DTD interno.

As principais desvantagens do DTD são: não ter suporte à validação semântica; não permitir muitos tipos de dados; não ter suporte a relacionamentos; não ter suporte a características de objetos, como herança; não permitir usar partes de outros DTDs; ser difícil de escrever; não poder ser estendido; não ter mecanismo de controle de versões e possuir sintaxe única.

1.2.8. Declarações: tipos de elementos e listas de atributos

A declaração de elementos corresponde a definir se o elemento é composto, do tipo texto (PCDATA) ou ambos (elemento misto). Os operadores indicam se os elementos são obrigatórios ou opcionais, quantas vezes e em que ordem eles ocorrem [Spencer, 1999]. A Tabela 1.2.8.1 apresenta o conjunto de todos os operadores e suas respectivas funções.

Tabela 1.2.8.1: Operadores [Spencer, 1999].

Símbolo	Função
,	ordem estrita
?	opcional
+	um ou mais
*	zero ou mais
|	selecione um
()	conjunto

O símbolo de *pipeline* (|) indica que um entre uma série de elementos deve estar presente. Por exemplo, para um elemento *sexo* com duas opções possíveis, a declaração deve ser assim:

```
<!ELEMENT sexo (M|F)>
```

No caso do elemento *telefone*, é possível restringir o conteúdo do atributo da seguinte maneira:

```
<!ATTLIST telefone tipo (residencial|celular|comercial) #REQUIRED>
```

O uso de #REQUIRED força a ocorrência do atributo sempre que o telefone for inserido, mas também é possível inserir um valor padrão:

```
<!ATTLIST telefone tipo (residencial|celular|comercial) "residencial">
```

Caso a presença do elemento não seja obrigatória e não haja um padrão para o atributo, pode ser usado #IMPLIED, e a lista de atributos pode ser substituída por CDATA para permitir inserção de qualquer valor:

```
<!ATTLIST telefone tipo CDATA #IMPLIED>
```

Outra alternativa possível é fixar o valor, usando #FIXED, nesse caso se o elemento telefone for usado sem atributo, o valor celular será assumido e não poderá ser alterado:

```
<!ATTLIST telefone tipo CDATA #FIXED "celular">
```

1.2.9. Namespaces

Elementos filhos que possuem o mesmo nome e significados diferentes são um problema para validação por DTDs. *Namespaces* tratam isso indicando que elementos devem ser interpretados de modos diferentes. Os *namespaces* podem ser identificados por URIs[4], que sejam únicas, como um endereço na Web. As recomendações de *namespaces* da W3C definem atalhos para associar prefixos curtos com a URI. XMLNS: declara o *namespace* como um atributo do elemento raiz do documento [Namespaces, 2006]. No exemplo abaixo define-se *ag* como um prefixo (ou apelido) para o elemento raiz agendas:

```
<ag:agendas xmlns:ag="http://www.xml.com/namespaces/agendas/">
```

Referências similares podem ser declaradas para outros contextos. Para o caso de outro documento XML que também possua o elemento *nome*, mas que seja referente a dados médicos em uma clínica, por exemplo, pode ter criado o *namespace* dele com prefixo *me* e utilizar o seguinte modelo para acessar o elemento nome nos dois casos (*agendas* e *médicos*):

```
<ag:nome>João</ag:nome>
<me:nome>José</me:nome>
```

1.2.10. Schemas

XML *Schema, ou esquemas XML,* é uma maneira de se descrever as características dos dados, assim como o DTD, mas aliando funcionalidade, facilidade de uso e compatibilidade. Um XML *Schema* define os blocos de construção do documento, indicando quais elementos e atributos que podem aparecer. Ele define também quem são os elementos filhos, quantos são e em qual ordem devem ocorrer, além de definir se os elementos são vazios ou contém texto. É possível ainda definir quais são os tipos de

---

[4] URI é uma cadeia de caracteres que identifica ou denomina um recurso na Internet.

dados permitidos para elementos e atributos, e quais são seus valores padrão ou fixos [Schema, 2001].

As vantagens de se usar esquemas XML ao invés de DTD são: (i) a possibilidade de extensão com adições, (ii) eles serem escritos em XML, (iii) eles serem mais ricos e úteis e (iv) eles suportarem tipos de dados e *namespaces*. O suporte a tipos de dados é vantajoso porque facilita a validação dos dados, o uso de banco de dados, a definição de restrições e formatos aos dados e a conversão dos dados para outros tipos. E o uso da sintaxe XML permite o uso do mesmo editor e *parser* do documento XML, além da manipulação do esquema com DOM e o uso de XSLT [Schema, 2009].

A especificação do formato de dados permite a comunicação segura de dados, como, por exemplo, quando é utilizado o tipo *date* sabe-se que o formato indica AAAA-MM-DD (ano-mês-dia).

```
<data type=”date”>2008-04-12</date>
```

XML *Schema* se tornou uma recomendação oficial da W3C em Maio de 2001 [Schema, 2001]. Seguindo o exemplo de uma agenda, o esquema XML do documento pode ser representado como:

```
<? xml version=”1.0” ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
      <xs:element name="agenda”>
      <xs:complexType>
       <xs:sequence>
         <xs:element name=" nome " type="xs:string"/>
         <xs:element name=" telefone " type="xs:string"/>
         <xs:element name=" enderecos " type="xs:string"/>
         <xs:element name=" email " type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
  </xs:element>
  </xs:schema>
```

Neste exemplo, agenda é um tipo complexo por conter outros elementos; os elementos restantes são denominados simples. O elemento *schema* é o raiz e pode conter atributos, que indicam de qual *namespace* vem os tipos de dados usados no esquema, e como o *namespace* deve ser referenciado (por exemplo por xs:). O último atributo indica que todo elemento usado em um documento XML declarado pelo esquema deve ser qualificado pelo *namespace* (elementFormDefault="qualified"). Nos documentos XML a referência ao esquema se dá após a declaração do XML:

```
<agenda xmlns="http://www.w3schools.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3schools.com agenda.xsd">
```

Os tipos de dados comuns a serem usados no esquema são: *string*, *decimal*, *integer*, *boolean*, *date* e *time*, sendo que todos devem ser precedidos pelo prefixo do

namespace (xs:). Elementos simples podem ter um valor padrão ou fixo, e em ambos os casos os valores dados são automaticamente atribuídos aos elementos, mas valores fixos não podem ser alterados.

```
<xs:element name="sexo" type="xs:string" default="M"/>
```

```
<xs:element name="sexo" type="xs:string" fixed="M"/>
```

Elementos com atributos sempre têm uma definição do tipo complexo e podem ser fixados como obrigatórios (use=”required”) ou opcionais (use=”optional”). Atributos obrigatórios são definidos assim:

```
<xs:attribute name="moeda" type="xs:string" use=”required”/>
```

Além disso, é possível definir vários tipos de restrições ou facetas, para elementos e atributos usando um esquema. Podem-se restringir valores, conjunto de valores possíveis, caracteres vazios, comprimento de strings e tipos de dados. Esse conteúdo não será abordado, mas pode ser consultado no material de referência.

1.2.11. *Parsers* XML: DOM *(Document Object Model)* e SAX

Um *parser* XML decompõe a estrutura de um documento em uma estrutura de árvore, permitindo que ela seja manipulada por meio de programas ou *scripts*. Dois padrões para *parser* bastante utilizados são DOM e SAX.

DOM (*Document Object Model*) permite acessar a estrutura de um documento XML, ler e modificar seu conteúdo. Toda manipulação em XML usa DOM, diretamente ou por meio de um processador XSL, por exemplo [DOM, 2009].

SAX (API Simples para XML) é uma API baseada em eventos para parser de XML escrita em linguagem orientada a objeto. SAX permite que programadores troquem o *parser* sem alterar a aplicação em uso [SAX, 2000].

1.2.12. Editores XML

XML é uma linguagem de marcação baseada em texto, portanto, seus documentos podem ser criados e editados usando um editor de texto convencional, como o Bloco de Notas. Entretanto, é mais fácil utilizar um editor específico para XML com suporte ao desenvolvimento, evitando erros e validando o documento com DTD ou esquema. Para evitar erros os editores adicionam *tags* de fechamento automaticamente, validam o documento e destacam a sintaxe do XML com cores.

O XMLSpy[5] é um editor da Altova que completa as *tags* automaticamente, checa se o documento é bem-formado, colore a sintaxe, faz validação com DTDs ou esquemas, suporta importação e exportação para banco de dados, além de permitir a criação, edição e verificação de XQuery e XSLT. Outra ferramenta que também auxilia na criação de XML é o Eclipse, que além de uma plataforma para desenvolvimento de diversas linguagens também auxilia na escrita de XML.

---

[5] Disponível em: http://www.altova.com/products/xmlspy/xml_editor.html

1.3. XSL (*Extensible Stylesheet Language)*

XSL transforma e adiciona folhas de estilo ao XML usando uma linguagem de transformação e outra de formatação. A linguagem de transformação gera a partir de um documento XML bem-formado um formato XML alternativo [Spencer, 1999].

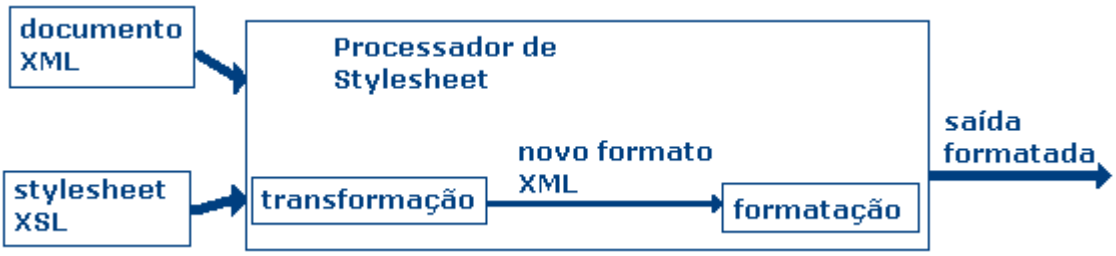

Figura 1.3.1: Um documento XML é apresentado com XSL passando por 2 fases.

A transformação reorganiza o conteúdo do documento adicionando, movendo ou excluindo elementos. XSL é um mecanismo de folhas de estilo personalizado que converte documentos XML em HTML, RTF ou mesmo outros vocabulários XML. O XSL é uma linguagem de formatação.

O XSL associa regras de formatação a elementos do XML e instrui o navegador sobre como formatar e exibir o conteúdo do elemento. É possível criar várias folhas de estilo gerando exibições diferentes do mesmo documento. As folhas de estilo XSL devem ser legíveis, claras e fáceis de serem criadas.

Alguns termos são fundamentais para entendimento da XSL [Pitts-Moultis and Kirk, 2000]:

- Regras de construção contêm instruções de formatação para qualquer elemento de documento dado;
- Padrão é a parte da regra que identifica o elemento a ser formatado;
- Ação é a parte da regra que corresponde à formatação do elemento;
- Objeto de Fluxo é a estrutura que descreve como formatar o conteúdo;
- Regra-raiz indica como um elemento deve ser formatado e
- Regra-padrão descreve como formatar conteúdo sem regras específicas.

Em XSL regras mais específicas têm precedência em caso de conflitos.

O elemento XSL declara que o documento é XML e configura o restante do documento para descrever regras de estilo para elementos contidos no documento ao qual a folha de estilos está associada. A instrução de processamento XML-STYLESHEET faz referência a uma folha de estilos XSL em um documento XML, assim:

```
<?XML_STYLESHEET HREF="/styles/main-style.xsl" TYPE="TEXT/XSL"?>
```

A XSL é uma linguagem para expressar folhas de estilo que são arquivos para descrever como um documento XML deve ser apresentado e é constituída por três partes: as transformações ou XSLT, a linguagem PATH ou XPATH e os objetos de

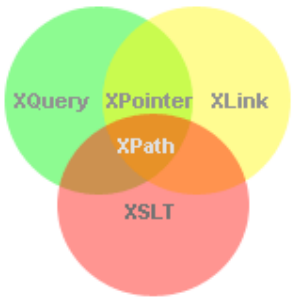

Figura 1.3.2: XPath e suas relações com outras linguagens.
Fonte: http://www.w3schools.com/XPath/xpath1.gif

formatação ou XSL-FO [XSL W3C, 2009]. XQuery e XPointer foram implementadas com expressões XPath. A Figura 1.3.2 ilustra a relação entre as linguagens.

1.4. XPATH

XPath é uma linguagem para acessar partes de documentos XML, ela usa XSLT e XPointer. Possui funcionalidades básicas para manipulação de *string*, números e valores *booleanos*. Ela opera na estrutura lógica de um documento XML, navegando pela estrutura hierárquica dele [XPATH, 2009].

1.4.1 Expressões em XPATH

O documento XML é visto pela XPath como uma árvore de nós, e existem três tipos de nós possíveis: de elementos, de atributos e de texto. XPath suporta XML *namespaces*. Uma expressão XPath é avaliada para produzir um objeto, que pode ser de quatro tipos básicos [XPATH, 2009]:

- Nó-conjunto (uma coleção não ordenada de nós, sem duplicações);
- Booleano (verdadeiro ou falso);
- Numérico (um valor em ponto flutuante) e
- String (uma sequência de caracteres).

XPath é uma sintaxe para definir partes de um documento XML e que usa expressões para navegar nos elementos e atributos dos documentos. XPath contém uma biblioteca com mais de 100 funções padrão e é uma recomendação da W3C.

Existem cinco tipos de nós possíveis de serem interpretados pela XPath: elementos, atributos, *namespaces*, instruções de processamento e nó raiz do documento. Valores atômicos constituem em nós sem filhos ou pais, como por exemplo, um valor de um atributo.

Os nós são classificados em pais, filhos, irmão, ancestrais ou descendentes. Todo elemento ou atributo possui nó **pai**, cada elemento pode ter nenhum, um ou mais nós **filhos**. Nós **irmãos** são os que possuem o mesmo pai, **ancestrais** correspondem a nós pais, ou pais de nós pais e **descendentes** são os nós filhos, ou filhos de nós filhos.

Existem algumas expressões para selecionar nós por meio de caminhos ou expressões. A Tabela 1.4.1.1 ilustra seis exemplos deles.

Tabela 1.4.1.1: Expressões XPath [XPATH, 2009].

Expressão	Descrição

nome do nó	seleciona todos os nós filhos do nó correspondente
/	seleciona do nó raiz
//	seleciona nós do documento a partir do nó atual
.	seleciona o nó atual
..	seleciona o pai do nó atual
@	seleciona os atributos do nó

Por exemplo, ao se utilizar a expressão "*agenda"* todos os nós filhos deste elemento serão selecionados. Já a expressão "*/agenda"* seleciona o elemento raiz *agenda*. "*agenda/nome"* retorna todos os elementos nome que são filhos de *agenda*. "*//telefone"* seleciona todos os elementos cujo nome é *telefone* em qualquer ponto do documento. "*agenda//endereco"* seleciona todos os elementos *endereco* que forem descendentes de *agenda* independentemente do local onde eles se encontram. E "*//@tipo"* seleciona todos os atributos cujos nomes sejam *tipo*.

Predicados são usados para encontrar nós específicos ou nós que possuam valores específicos, e sempre estão entre colchetes. A Tabela 1.4.1.2 exemplifica resultados que retornam de expressões com predicados no formato XPath.

Tabela 1.4.1.2: Expressões de XPath [XPATH, 2009].

Expressão PATH	**Resultados**
/agenda/nome[1]	seleciona o primeiro elemento nome que é filho de agenda
/agenda/nome[last()]	seleciona o último elemento nome que é filho de agenda
/agenda/nome[last() - 1]	seleciona o penúltimo elemento nome que é filho de agenda
/agenda/nome[position() < 3]	seleciona os dois primeiros elemento nome que é filho de agenda
//telefone[@tipo]	seleciona todos os elementos telefone que tenham um atributo tipo
//telefone[@tipo='celular']	seleciona todos os elementos telefone que tenham um atributo tipo de valor celular
/agenda/nome[1]	seleciona o primeiro elemento nome que é filho de agenda

Há três expressões genéricas para seleção de nós em XPath. O sinal de * seleciona qualquer nó elemento. Já @* seleciona qualquer atributo de elemento e a expressão node() seleciona qualquer nó, de qualquer tipo.

Para selecionar diversos caminhos pode-se usar o sinal |. Por exemplo, ao usar `//agenda/nome | //agenda/telefone` são selecionados todos os nomes e os telefones de todos os elementos agenda.

1.4.2 Eixos XPATH

Eixos em XPath definem um conjunto de nós relativo ao nó atual, a Tabela 1.4.2.1 apresenta 13 eixos que podem ser usados para seleção de nós.

Tabela 1.4.2.1: Eixos Path e seus resultados [XPATH, 2009].

Eixo PATH	**Resultados**
ancestor	seleciona todos os ancestrais (pais, avós, etc.) do nó atual
ancestor-or-self	seleciona todos os ancestrais (pais, avós, etc.) do nó atual e ele próprio
attribute	seleciona todos os atributos do nó atual
child	seleciona todos os filhos do nó atual
descendant	seleciona todos os descendentes (pais, avós, etc.) do nó atual
descendant-or-self	seleciona todos os descendentes (pais, avós, etc.) do nó atual e ele próprio
following	seleciona tudo no documento que vier na sequência do nó atual
following-sibling	seleciona todos os irmão após o nó atual
namespace	seleciona todos os nós de namespaces do nó atual
parent	seleciona o pai do nó atual
preceding	seleciona tudo no documento que está antes do nó atual
preceding-sibling	seleciona todos os irmão antes do nó atual
self	seleciona o nó atual

Um caminho para localização de nós pode ser absoluto ou relativo, caminhos para locais absolutos iniciam com barra (/) e relativos não, mas ambos consistem em um ou mais passos separados por barra. Cada passo é comparado ao conjunto de nós com relação ao nó atual. Um passo consiste em um eixo, um teste de nó ou zero ou mais predicados, sendo que um eixo define as três relações possíveis entre os nós selecionados e o atual, um teste de nó identifica um nó dentro de um eixo, e os predicados refinam o conjunto de nós selecionados. A sintaxe para um passo de localização é definida assim: `nomedoeixo::testedenó[predicado]`. A Tabela 1.4.2.2 ilustra como podem ser usados comandos para localização de nós.

Tabela 1.4.2.2: Exemplos de uso [XPATH, 2009].

Exemplo PATH	**Resultados**
child::nome	seleciona todos os nós nome que são filhos do nó atual

attribute::tipo	seleciona o atributo tipo do nó atual
child::*	seleciona todos os filhos do nó atual
attribute::*	seleciona todos os atributos do nó atual
child::text()	seleciona todos filhos do tipo texto do nó atual
child::node()	seleciona todos filhos do nó atual
descendant::telefone	seleciona todos descendentes telefone do nó atual
ancestor::endereco	seleciona todos ancestrais endereco nó atual
ancestor-or-self::nome	seleciona todos ancestrais nome do nó atual – é ele próprio se ele for nome também
child::*/child::nome	seleciona todos os netos nome do nó atual

1.4.3 Operadores em XPATH

Existe uma lista de operadores que podem ser usados para comandos em XPath. Uma expressão em XPath retorna um conjunto de nós, uma string, um booleano ou um número. A Tabela 1.4.3.1 ilustra os operadores, suas funções, exemplos de como podem ser usados e seu valor de retorno.

Tabela 1.4.3.1: Operadores relacionais e suas funções [XPATH, 2009].

Operador	**Descrição**	**Exemplo**	**Valor de retorno**
|	processa dois conjuntos de nós	// nome | //telefone	um conjunto de nós com todos os elementos nome e telefone
+	adição	2 + 1	3
-	subtração	3 - 2	1
*	multiplicação	4 * 2	8
div	divisão	8 div 4	2
=	igual	valor = 1	verdadeiro se o valor for 1, falso caso contrário
!=	diferente	valor != 2	verdadeiro se o valor não for 2, falso caso contrário
<	menor que	valor < 3	verdadeiro para valores menores que 3, falso caso contrário
<=	menor ou igual a	valor <= 4	verdadeiro para valores menores ou iguais a 4, falso caso contrário
>	maior que	valor > 3	verdadeiro para valores maiores que 3, falso caso contrário
>=	maior ou igual a	valor >= 3	verdadeiro para valores maiores ou

			iguais a 3, falso caso contrário
or	ou	valor = 3 or valor = 1	verdadeiro para valor 1 ou 3, falso caso contrário
and	e	valor > 2 and valor < 5	verdadeiro para valores 3 ou 4, falso caso contrário
mod	módulo (resto da divisão)	5 mod 2	1

Os métodos usados para seleção de nós são diferentes conforme o tipo de navegador utilizado. Para o Internet Explorer, usando o XMLDOM para carregar o documento e o método *selectNodes()*, os seguinte comandos devem ser usados:

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("books.xml");
xmlDoc.selectNodes(xpath);
```

Para outros navegadores, usando o método *implementation()* para carregar o documento e o método *evaluate()* para selecionar os nós, os seguinte comandos devem ser usados:

```
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async=false;
xmlDoc.load("books.xml");
xmlDoc.evaluate(xpath, xmlDoc, null, XPathResult.ANY_TYPE,null);
```

Para selecionar todos os nomes de todas as agendas o seguinte comando deve ser usado:

```
agendas/agenda/nome
```

Substituindo `agenda` por `agenda[1]` seleciona-se o primeiro nó agenda. No entanto, dependendo do navegador em uso os resultados são diferentes. Para o IE o índice [0] retorna o primeiro nó e segundo o padrão da W3C isto deveria ocorrer para o índice [1] [XPATH, 2009]. Então para resolver esse problema deve-se configurar a linguagem para XPath, com o seguinte código:

```
xml.setProperty("SelectionLanguage","XPath");
xml.selectNodes("/agendas/agenda[1]/nome");
```

1.5. XSLT

XSL (*EXtensible Stylesheet Language*) foi desenvolvida pela W3C (*World Wide Web Consortium*) porque havia a necessidade de uma linguagem para folhas de estilo baseadas em XML. Uma folha de estilo XSL consiste em um ou mais conjunto de regras chamadas *templates*. Um *template* é composto por regras que são aplicadas quando um determinado nó é encontrado.

1.5.1 Funções da XSLT

XSLT se refere a transformações do XSL e possibilita transformar documentos XML em outros formatos, como HTML ou XHTML. XSLT é a parte mais importante de XSL, usa XPath [XPATH, 2009] para navegar pelos elementos e atributos dos documentos XML, e é uma recomendação da W3C [XSLT, 2009].

Com XSLT é possível remover ou adicionar elementos e atributos tanto do documento original ou o resultante. Também é possível rearranjar ou ordenar elementos, executar testes e tomar decisões, por exemplo, sobre quais elementos devem ou não ser exibidos. Pode-se dizer então que XSLT transforma uma árvore-fonte XML em uma árvore resultante [XSLT, 2009].

No processo de transformação XSLT usa o XPath [XPATH, 2009] para definir quais partes do documento original devem ser tratadas por *templates* já definidos, quando o conjunto de nós associado ao estilo é encontrado, ele é transformado gerando o documento resultante.

1.5.2 XSLT e os Navegadores Web

Os navegadores Internet Explorer 6, Mozilla Firefox 3, Google Chrome 1 e Opera 9 suportam XML, XSLT e XPath [XSLT, 2009].

1.5.3 Estrutura XSLT

O elemento raiz que declara que o documento é uma folha de estilo XSL é `<xsl:stylesheet>` ou `<xsl:transform>` as duas declarações são equivalentes, portanto, ambas podem ser usadas [XSLT, 2009]. Conforme ilustram os dois exemplos abaixo respectivamente:

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

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

Para acessar os elementos, atributos e funções XSLT deve-se declarar seu namespace correspondente incluindo o atributo de versão=”1.0”. O exemplo de folha de estilo abaixo (*agendas.xsl*) aplica uma transformação no documento original.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>Meus contatos telefônicos</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Nome</th>
      <th align="left">Telefone</th>
    </tr>
    <xsl:for-each select="agendas/agenda">
    <tr>
      <td><xsl:value-of select="endereco"/></td>
      <td><xsl:value-of select="email"/></td>
    </tr>
```

```
      </xsl:for-each>
      </table>
    </body>
    </html>
  </xsl:template>
  </xsl:stylesheet>
```

O elemento *template* é usado para construir *templates* e o atributo *match* associa um *template* com um elemento XML (ou com um documento XML inteiro). O valor do atributo *match* é uma expressão XPath (por exemplo: / que define que o *template* se aplica ao documento todo).

Como uma folha de estilos é um documento XML, ela deve começar com a declaração da linguagem, o elemento seguinte (`<xsl:stylesheet>`) define que o documento é uma folha de estilo XSLT, seguido pelo número da versão, e pelos atributos de *namespace* do XSLT. O elemento (`<xsl:template>`) define um template onde o atributo match="/" associa o *template* com o elemento raiz do documento. O conteúdo interno ao elemento `<xsl:template>` define código HTML para gerar e exibir o documento de saída. As duas últimas linhas definem o final do *template* e da folha de estilos.

Então se deve adicionar a referência à folha de estilos ao documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="agendas.xsl"?>
<agendas>
<agenda>
<nome>Maria </nome>
<endereco>Rua Silva, 4</endereco>
<telefone>123</telefone>
</agenda>
</agendas>
```

1.5.4 Elemento Value-of

Assim, com um navegador que interprete XSLT pode-se visualizar o resultado da transformação de XML em XHTML. O elemento `<xsl:value-of>` pode ser usado para extrair o valor de um elemento XML e adicioná-lo à transformação de saída [XSLT, 2009]. Observe o exemplo abaixo:

```
      <?xml version="1.0" encoding="ISO-8859-1"?>
      <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
   <h2>Meus contatos telefônicos</h2>
   <table border="1">
     <tr bgcolor="#9acd32">
       <th>Nome</th>
       <th>Telefone</th>
     </tr>
     <tr>
```

```
      <td><xsl:value-of select="agenda/nome"/></td>
      <td><xsl:value-of select="agenda/telefone"/></td>
     </tr>
   </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

O valor do atributo select é uma expressão XPath [XPATH, 2009] que funciona como um sistema de navegação por arquivos no qual a barra (/) seleciona um subdiretório. O exemplo acima retorna apenas o primeiro elemento do documento XML. Para retornar mais elementos deve-se usar o elemento: `<xsl:for-each>` conforme ilustra o exemplo abaixo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
   <h2>Meus contatos telefônicos</h2>
   <table border="1">
     <tr bgcolor="#9acd32">
       <th>Nome</th>
       <th>Telefone</th>
     </tr>
    <xsl:for-each select="agendas/agenda">
     <tr>
      <td><xsl:value-of select="nome"/></td>
      <td><xsl:value-of select="telefone"/></td>
     </tr>
    </xsl:for-each>
   </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

O resultado da aplicação dessa transformação resulta em uma tabela com todos os registros existentes no documento XML original. Também é possível adicionar um critério de seleção ao atributo no elemento `<xsl:for-each>`. Por exemplo, para selecionar contatos cujo nome seja Maria deve-se usar o seguinte comando: `<xsl:for-each select=”agendas/agenda[nome='Maria']”>`. Há quatro operadores de filtro que podem ser usados: = (igual), != (diferente), < (menor que) e > (maior que). E o resultado dessa transformação consiste em uma tabela com registros cujo nome seja igual a Maria.

1.5.5 Ordenação e Condições em XSLT

Para ordenar as informações pode-se usar o elemento `<xsl:sort>` após o uso do `<xsl:for-each>` especificando qual elemento deve ser ordenado. Por exemplo, para ordenar pelo elemento nome, deve-se usar: `<xsl:sort select=”nome”>`.

Para incluir um teste condicional deve-se usar o elemento `<xsl:if>` que deve ser colocado após o comando `<xsl:for-each>`. O exemplo abaixo ilustra como incluir um teste para seleção de nós cujos nomes sejam diferente de Maria.

```
<xsl:for-each select="agendas/agenda">
 <xsl:if test="nome != 'Maria'">
  <tr>
   <td><xsl:value-of select="nome"/></td>
   <td><xsl:value-of select="telefone"/></td>
  </tr>
 </xsl:if>
</xsl:for-each>
```

O valor atributo *test* possui a expressão a ser avaliada. Para incluir testes com condições múltiplas deve-se combinar três comandos: `</xsl:choose>`, `</xsl:when>` e `</xsl:otherwise>`. O exemplo abaixo ilustra como eles podem ser usados:

```
<xsl:choose>
 <xsl:when test="nome != Maria">
  <td bgcolor="#ff00ff">
  <xsl:value-of select="nome"/></td>
 </xsl:when>
 <xsl:otherwise>
  <td><xsl:value-of select="nome"/></td>
 </xsl:otherwise>
</xsl:choose>
```

Este exemplo coloca como fundo da tabela a cor rosa para registros cujos nomes sejam diferentes de Maria. O elemento `<xsl:choose>` serve para expressar um teste com múltiplas condições.

O elemento `<xsl:apply-templates>` aplica um template ao elemento atual ou aos seus filhos. Ao adicionar um atributo de seleção ao elemento `<xsl:apply-templates>` ele processará somente o filho que corresponda ao valor do atributo. Pode-se usar o atributo select para especificar a ordem na qual os nós filhos são processados.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
 <body>
  <h2>Meus contatos telefônicos</h2>
   <xsl:apply-templates/>
 </body>
</html>
</xsl:template>
```

```
 <xsl:template match="agendas">
  <p>
   <xsl:apply-templates select="agenda/nome"/>
   <xsl:apply-templates select="agenda/telefone"/>
  </p>
 </xsl:template>
 <xsl:template match="agenda/nome">
   Nome: <span style="color:#ff0000">
   <xsl:value-of select="."/></span>
   <br />
 </xsl:template>
 <xsl:template match="agenda/telefone">
   Telefone: <span style="color:#00ff00">
   <xsl:value-of select="."/></span>
   <br />
  </xsl:template>
</xsl:stylesheet>
```

O resultado desta transformação consiste em uma lista com os nomes e telefones do documento XML sendo que os nomes são exibidos em cor vermelha e os telefones em cor verde.

1.5.6 Características da XSLT

Os objetivos do XSLT consistem em transformar documentos em formatos diversos e suportar navegadores e necessidades de usuários diferentes. Como nem sempre é desejável incluir uma referência explícita à folha de estilos em um documento XML, uma solução mais versátil seria utilizar JavaScript para fazer a transformação. Com JavaScript pode-se testar qual navegador está sendo usado e usar folhas de estilo adequadas conforme o navegador e as necessidades do usuário. Com navegadores mais específicos no mercado, como em Braille, e para dispositivos móveis, XSLT tende a ser altamente necessário para apresentar documentos web [XSLT, 2009].

Para informações sobre como usar JavaScript (ou até mesmo ASP) na transformação de documentos XML usando folha de estilo, consulte material do tutorial da W3C. Disponível em: http://www.w3schools.com/xsl/xsl_client.asp. Com ASP pode-se transformar o documento XML original em HTML no servidor, e enviar o resultado para o navegador, de modo que não é necessário adequar o código conforme a especificação usada [XSLT, 2009].

1.6. XSL-FO

XSL-FO (*Extensible Stylesheet Language – Formatting Objects*) é uma linguagem de marcação baseada em XML que descreve a formatação de dados XML para saída em papel, tela ou outras mídias. Documentos XSL-FO são documentos XML com informações sobre o *layout* e o conteúdo de saída. *Formatting Objects* é uma linguagem para editoração, ou seja, para estruturar a apresentação de um documento XML em diferentes tipos de arquivos de saída (PDF, PS, SVG, HTML, TXT). XSL-FO é uma recomendação da W3C. XSL-FO hoje é formalmente denominada XSL [FO, 2009].

1.6.1 Definição XSL-FO

XSL-FO é um conjunto de ferramentas para descrever a apresentação do XML, pode ser usado para produzir material impresso de alta qualidade. A marcação do XSL-FO é um tanto quanto complexa e verbosa: há 56 elementos e mais de 200 atributos. A maneira prática de produzir documentos XSL-FO é usando XSLT para produzir um documento fonte, para que seja então renderizado em um meio de saída. Como não há muitas ferramentas que fazem isto XSL-FO não se popularizou tão rapidamente quanto XSLT.

O CSS também pode ser usado na formatação de documentos para a web, no entanto, XSL-FO é mais completo, pois permite formatar como livro, página web, inclui todos os estilos CSS2 para áudio, direciona textos, configura margens, rodapés e cabeçalhos, controla a separação de sílabas, e sua aplicação não está restrita somente à web.

1.6.2 Processamento XSL-FO

Dois passos são necessários para a transformação de um documento usando XSL-FO, eles estão ilustrados na Figura 1.6.2.1 Inicialmente, o processador XSLT recebe o documento XML e sua folha de estilos apropriada, desenvolvida especialmente para produzir outro documento XML, que usa o namespace XSL-FO, e é esperada pelo formatador XSL-FO. Gerado o documento resultante, ele é processado pelo formatador XSL-FO que produz então o produto final: um documento impresso, formatado para apresentação visual.

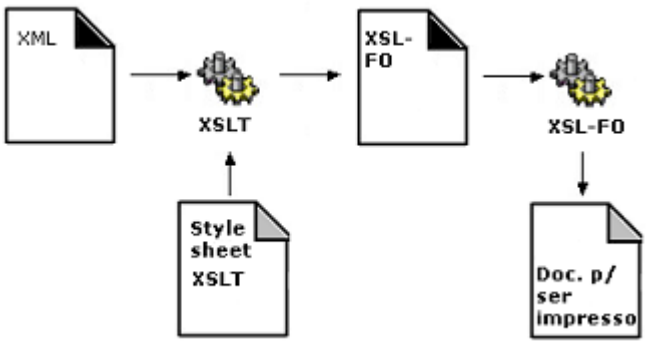

Figura 1.6.2.1. Processamento XSL-FO [FO, 2002].

Para processamento do documento XSL-FO os seguintes aplicativos podem ser usados: RenderX [RenderX, 2005], Antenna House [AH, 1996], Passive Tex [PT, 1999] e FOP [FOP, 2009], estes dois últimos são gratuitos, enquanto os dois primeiros são versões comerciais, mas possuem versão para teste. FOP (*Formatting Objects Processor*) é uma aplicação Java de código aberto para leitura e renderização de documentos FO, conforme ilustra a Figura 1.6.2.2, que provê vários formatos de arquivos de saída, como PDF, PostScript, SVG, XML, PNG, RTF, AWT, MIF e TXT [FOP, 2009].

Os requisitos para que o documento resultante do FO seja gerado e visualizado consistem em possuir Java 1.1x instalado, parser XML com suporte a SAX ou DOM, parser XSLT, biblioteca SVG (do arquivo w3c.jar do FOP). O comando a ser usado é *Fop livro.fo livro.pdf* que produz o arquivo em PDF que pode ser visualizado em uma aplicação do tipo Acrobat Reader [FOP, 2009].

A vantagem do XSL-FO é que seu arquivo de entrada é um XML, que pode ser gerado de fontes diferentes.

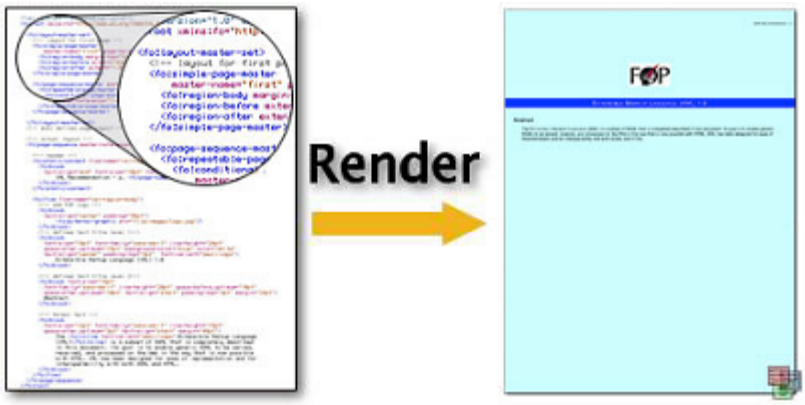

Figura 1.6.2.2. Processo de renderização do FOP[6].

1.6.3 Terminologia e Estrutura em XSL-FO

O vocabulário FO descreve como será a apresentação para o leitor. Documentos XSL-FO são armazenados em arquivos com extensão .fo ou .fob. Também é comum encontrar esses documentos com a extensão .xml, pois isso os torna mais acessíveis a editores XML. Dos 56 elementos FO, 99% deles são iniciados pelo prefixo FO:. A estrutura de um documento XSL-FO é iniciada pela declaração do XML e sua versão e codificação (como todo documento baseado em XML).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
  <fo:simple-page-master master-name="A4">
    <!– Espaço para o template da página -->
  </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="A4">
  <!–Espaço para o conteúdo da página -->
</fo:page-sequence>
</fo:root>
```

Após a declaração do XML tem-se o elemento `<fo:root>` que é o elemento raiz do documento, no qual está contida a declaração do namespace para o documento. O elemento `<fo:layout-master-set>` possui um ou mais templates de páginas e especifica o tamanho da página. Cada elemento, por sua vez, contém um template individual de página. `<fo:simple-page-master>`. Cada template deve ter um nome único no documento que é definido pelo seguinte elemento `<fo: simple-page-master`

---

6 Fonte: http://xmlgraphics.apache.org/fop/

master-name="Nome">. Um ou mais elementos <fo:page-sequence> descrevem o conteúdo da página, sendo que o atributo master-reference se refere ao template simple-page-master com o mesmo nome. O atributo master-reference pode receber um nome qualquer mas é recomendado que ele descreva um formato de página pré-definido, que faça referência à estrutura da apresentação do conteúdo, como A4 ou contra-capa, por exemplo.

1.6.4 Elemento Área

Elementos de área (fo:area) definem áreas retangulares, ou caixas de texto, onde serão apresentados os conteúdos. Todos os resultados de saída, como imagens, gráficos ou textos, devem ser formatados nessas caixas nas quais eles serão impressos ou exibidos. Existem cinco tipos específicos de área que serão detalhados:

- Pages,
- Regions,
- Block areas,
- Line areas e
- Inline areas.

Há quatro elementos que são considerados principais em um documento FO: regiões, blocos, linhas e entre-linhas. Sendo que as regiões ocupam o mais alto nível na hierarquia do documento e contêm cabeçalho, texto e rodapé; blocos representam um bloco de texto como um parágrafo, fo:block e fo:list-block; linhas representam linhas de texto em um parágrafo; e entre-linhas são partes de linhas que podem ser compostas por caracteres especiais, referências a rodapés, ou equações matemáticas, como, por exemplo, fo:external-graphics e fo:inline.

1.6.5 Elemento Página

Elementos de páginas (fo:pages) servem para formatação do conteúdo em FO. Saídas impressas são automaticamente geradas em páginas separadas enquanto documentos apresentados em navegadores são exibidos em uma única página contínua. As páginas contêm elementos de região (fo:region) que são classificados em cinco tipos:

- region-body (referente ao corpo da página)
- region-before (referente ao cabeçalho da página)
- region-after (referente ao rodapé da página)
- region-start (referente à lateral esquerda da página)
- region-end (referente à lateral direita da página)

As regiões das páginas contêm os elementos áreas de blocos (fo:block-areas) que definem pequenos elementos de blocos que normalmente começam com uma nova linha como parágrafos, tabelas ou listas. Essas áreas podem conter outras áreas de blocos, mas em geral elas contêm áreas de linha. Os elementos áreas de linhas (fo:line-areas) definem linhas de texto dentro das áreas de blocos e podem conter

elementos entre-linhas (`fo:inline-areas`) que definem, por sua vez, textos internos às linhas, como marcadores, caracteres especiais, e gráficos.

1.6.6 Elemento Bloco

Blocos de conteúdos fluem nas páginas e também nas mídias de saída. Uma saída XSL-FO é normalmente aninhada dentro de elementos de blocos (`fo:block`) que por sua vez, são aninhados dentro de elementos de fluxo (`fo:flow`), usados para distribuir conteúdos nas páginas, que por sua vez, são aninhados dentro de elementos de sequência de página (`fo:page-sequence`), estes elementos são usados para suporte à paginação complexa. Conforme ilustra o exemplo abaixo:

```
<fo:page-sequence>
  <fo:flow flow-name="xsl-region-body">
    <fo:block>
      <!—Espaço para a saída -->
    </fo:block>
  </fo:flow>
</fo:page-sequence>
```

Para exibir o conteúdo de texto ‘Texto de exemplo’ na página, o seguinte código pode ser utilizado:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
<fo:layout-master-set>
  <fo:simple-page-master master-name="A4">
    <fo:region-body />
  </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="A4">
  <fo:flow flow-name="xsl-region-body">
    <fo:block>Texto de exemplo.</fo:block>
  </fo:flow>
</fo:page-sequence>
</fo:root>
```

1.6.7 Elemento Sequência

Os elementos de sequência de páginas em XSL-FO (`<fo:page-sequence>`) definem as páginas de saída que por sua vez se referem à página mestre onde está definido o layout. Cada página de saída possui um elemento de fluxo `<fo:flow>` que possui todos os elementos a serem impressos na página. Quando uma página está completa, a mesma estrutura da página mestre será continuamente utilizada até que todo o texto ou conteúdo tenha sido organizado.

1.6.8 Elemento Fluxo

O elemento de fluxo `<fo:flow>` possui um atributo de nome (`flow-name`) que define onde o conteúdo do elemento deve ser apresentado. Os seguintes valores são válidos para esse atributo:

- region-body
- region-before
- region-after
- region-start
- region-end

A Figura 1.6.8.1 exibe quais são os locais correspondentes a esses atributos:

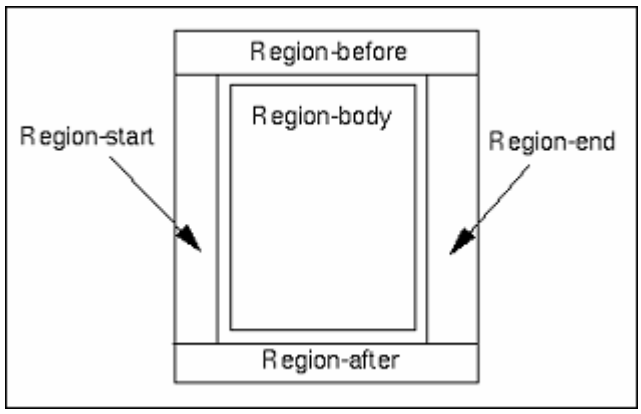

Figura 1.6.8.1. Áreas de conteúdo nas páginas.

1.6.9 Templates

O XSL-FO usa *templates* de páginas chamados de página mestre para definir o *layout* das páginas e cada *template* deve possuir um nome único no documento. No exemplo abaixo três elementos de página mestre definem três *templates* diferentes, que têm nomes diferentes. O primeiro *template* se denomina capa, e pode ser usado para elabora primeira página de um documento, e o segundo e o terceiro *templates* são chamados de esquerda e direita e correspondem respectivamente às páginas pares e ímpares, por exemplo.

```
<fo:simple-page-master master-name="capa">
  <fo:region-body margin="5in" />
</fo:simple-page-master>
<fo:simple-page-master master-name="esquerda">
  <fo:region-body margin-left="2in" margin-right="3in" />
</fo:simple-page-master>
<fo:simple-page-master master-name="direita">
  <fo:region-body margin-left="3in" margin-right="2in" />
</fo:simple-page-master>
```

1.6.10 Tamanho de Páginas

Para definir o tamanho de uma página XSL-FO usa os seguintes atributos:

- page-width que define a largura de uma página e
- page-height que define a altura de uma página.

  E para definir as margens de uma página cinco atributos devem ser usados:

- margin-top que define a margem superior
- margin-bottom que define a margem inferior
- margin-left que define a margem da lateral esquerda
- margin-right que define a margem da lateral direita
- margin que define as quatro margens

1.6.11 Regiões de Texto

A região do corpo do texto (`region body`) identifica em qual região da página o texto será exibido, ela é composta por quatro regiões: `region-before`, `region-start`, `region-after` e `region-end`. E para evitar que o texto na região do corpo da página se sobreponha nestas regiões, a região do corpo de texto deve possuir margens com pelo menos o mesmo tamanho dessas regiões [XSL-FO, 2001]. A Figura 1.6.11.1 ilustra um exemplo de sobreposição causada por margens menores do que as configuradas pelo elemento de região do corpo de texto [XSL-FO, 2001].

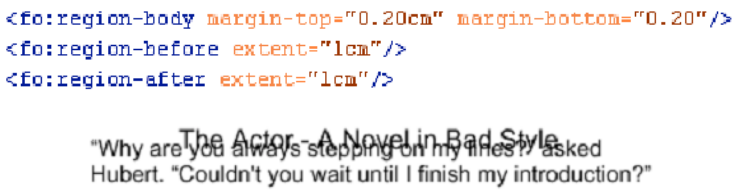

Figura 1.6.11.1. Exemplo de sobreposição de texto [XSL-FO, 2001].

Para se definir um template simples de página mestre, cujo nome seja A4, o seguinte excerto de código pode ser utilizado:

```
<fo:simple-page-master master-name="A4"
 page-width="297mm" page-height="210mm"
 margin-top="1cm"   margin-bottom="1cm"
 margin-left="1cm"  margin-right="1cm">
  <fo:region-body   margin="3cm"/>
  <fo:region-before extent="2cm"/>
  <fo:region-after  extent="2cm"/>
  <fo:region-start  extent="2cm"/>
  <fo:region-end    extent="2cm"/>
</fo:simple-page-master>
```

A largura da página está definida em milímetros pelo atributo `page-width` e a altura, também em milímetros, pelo atributo `page-height`. As margens superior, inferior, esquerda e direita foram fixadas em 1 centímetro. O corpo da página tem uma

margem de 3 centímetros em todas as laterais. E as regiões de cabeçalho, rodapé, esquerda e direita da página possuem todas elas 2 centímetros [XSL-FO, 2001].

Há seis tipos diferentes de unidades que podem ser usadas para definir larguras: px, pt, mm, cm, em e in [XSL-FO, 2001].

A largura do corpo da página nesse exemplo pode ser calculada subtraindo-se a margem esquerda e direita e as margens da região do corpo de texto (`region-body`) do total da largura de página. As regiões correspondentes às laterais esquerda e direita da página (`region-start` e `region-end` respectivamente) não fazem parte do cálculo porque elas são partes do corpo de texto [XSL-FO, 2001].

A Figura 1.6.11.2 [FO W3C, 2000] ilustra a estrutura dos elementos de página que são utilizados em um documento FO conforme a hierarquia, observa-se que a partir do nós raiz há três elementos a serem declarados com seus respectivos elementos filhos.

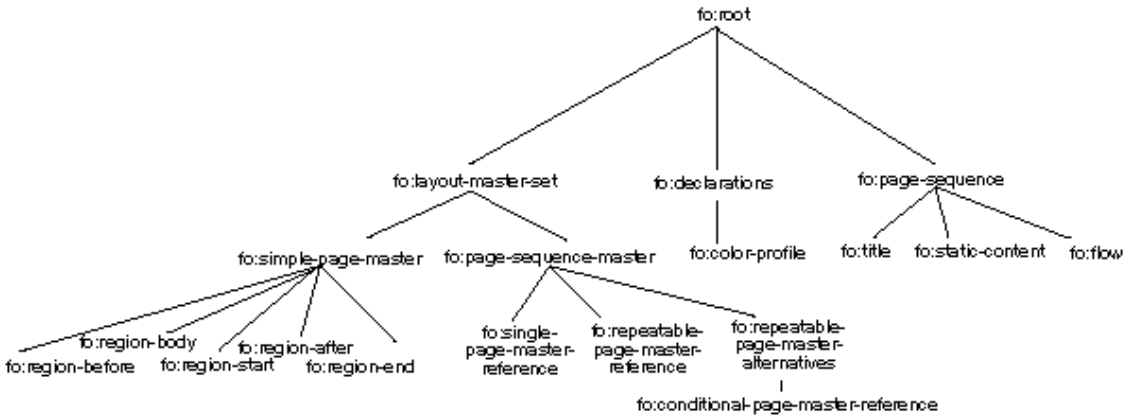

Figura 1.6.11.2. Representação dos FO para estrutura de paginação[7].

1.6.12 Blocos de Texto

As saídas da XSL-FO são estruturadas em blocos. Blocos distribuem o conteúdo pelas páginas de saída. A saída da XSL-FO está normalmente aninhada em elementos de bloco aninhados em elementos de fluxo, que por sua vez estão aninhados em elementos de sequência de página. Blocos são sequências de saídas em caixas de texto retangulares, conforme ilustrado no exemplo seguinte:

```
<fo:block border-width="1mm">
A borda ao redor deste bloco de saída possui 1 milímetro.
</fo:block>
```

Como áreas de blocos são caixas retangulares, elas têm muitas propriedades de área comuns:

- space before
- space after

---

[7] Fonte: www.w3.org/TR/2001/REC-xsl-20011015/PageTree.gif

- margin
- border
- padding

O `space-before` e o `space-after` são espaços em branco que separam o bloco de outros blocos. A margem (`margin`) é uma área vazia externa ao bloco. A borda (`border`) é o retângulo desenhado ao redor do limite externo da área. Ela pode ter diferentes larguras em cada um dos quatro cantos. Ela também pode ser preenchida como diferentes cores e imagens de fundo. O `padding` é a área entre a borda e a área de conteúdo. A área de conteúdo (`content`) possui o conteúdo real, como textos, figuras, gráficos, entre outros. A Figura 1.6.12.1 ilustra a relação entre as margens de página conforme localização.

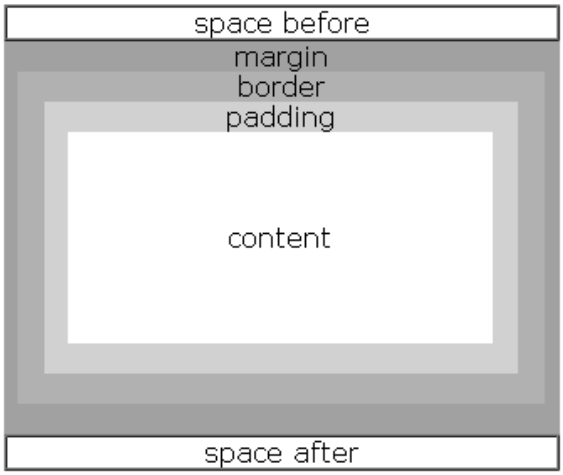

Figura 1.6.12.1. Margens de página [FO, 2009].

Há cinco tipos de margens de bloco [FO W3C, 2000]:

- margin
- margin-top
- margin-bottom
- margin-left
- margin-right

E os atributos de bordas de bloco são classificados em três tipos: estilo, cor e largura, que podem ser definidos em oito critérios além deles próprios (`before`, `after`, `start`, `end`, `top`, `bottom`, `left`, `right`). A Tabela 1.6.12.1 lista os atributos de borda.

Tabela 1.6.12.1. Atributos de estilo de borda

Border style attributes	**Border color attributes**	**Border width attributes**
border-style	border-color	border-width

border-before-style	border-before-color	border-before-width
border-after–style	border-after–color	border-after–width
border-start–style	border-start–color	border-start–width
border-end–style	border-end–color	border-end–width
border-top–style	border-top–color	border-top–width
border-bottom-style	border-bottom-color	border-bottom-width
border-left–style	border-left–color	border-left–width
border-right-style	border-right-color	border-right-width

Para o elemento `block-padding` há nove atributos que podem ser configurados: i) padding; ii) `padding-before`; iii) `padding-after`; iv) `padding-start`; v) `padding-end`; vi) `padding-top`; vii) `padding-bottom`; viii) `padding-left` e ix) `padding-right`.

Para o elemento `block-background` há quatro atributos que podem ser configurados: i) `background-color`; ii) `background-image`; iii) `background-repeat` e iv) `background-attachment` (`scroll` ou `fixed`).

Blocos são sequências de saídas que podem ter seu estilo configurado individualmente [FO, 2001]. O código abaixo ilustra um exemplo:

```
<fo:block font-size="12pt" font-family="sans-serif">
Este bloco de saída será escrito em fonte sem serifa com tamanho 12.
</fo:block>
```

Há cinco atributos de fontes que podem ser configurados: i) `font-family`; ii) `font-weight`; iii) `font-style`; iv) `font-size` e v) `font-variant`. E nove atributos de texto que podem ser configurados: i) `text-align`; ii) `text-align-last`; iii) `text-indent`; iv) `start-indent`; v) `end-indent`; vi) `wrap-option` (define se haverá rolagem); vii) `break-before` (define quebra de página); viii) `break-after` (define quebra de página) e ix) `reference-orientation` (define rotação de texto em incrementos de 90º).

```
<fo:block font-size="14pt" font-family="verdana" color="red"
          space-before="5mm" space-after="5mm">
   Título
</fo:block>
<fo:block text-indent="5mm" font-family="verdana" font-size="12pt"
    space-before="5mm" space-after="5mm">
    Conteúdo.
</fo:block>
```

Neste exemplo é possível notar que muito código é necessário para gerar um documento com muitos cabeçalhos e parágrafos. Normalmente o documento em XSL-FO não combina informação de formatação e conteúdo como no exemplo acima [FO,

2002]. Utilizando XSLT pode-se colocar informação de formatação em *templates* e escrever um código mais claro.

Para o *layout* das páginas FO considera que uma página é composta por 2 elementos:

- Block: parágrafos que começam em uma nova linha
- Inline: negrito, itálico

Sendo que o atributo `Block progress direction` define a ordem na qual os parágrafos se posicionam na página, e permite que conteúdos em diferentes linguagens sejam apresentados.

1.6.13 Listas de Itens

XSL-FO usa blocos de lista para definir listas. Existem quatro objetos em XSL-FO que são usados para criar listas:

- fo:list-block, que contém a lista completa;
- fo:list-item, que contém cada item da lista;
- fo:list-item-label, que contém o rótulo para o item da lista, tipicamente um `<fo:block>` contendo um número, ou caractere e
- fo:list-item-body, que contém o conteúdo ou corpo do item da lista tipicamente composto por um ou mais objetos `<fo:block>`.

1.6.14 Exemplos em XSL-FO

Para inserir numeração de páginas em um elemento bloco deve-se usar o elemento `fo:page number`, sendo que o elemento `fo:leader` insere uma linha sobre o número de página, seus atributos `leader-pattern` e `leader-length` definem respectivamente o tipo e a espessura do traço. A Figura 1.6.14.1 exemplifica o uso desses elementos:

```
<fo:page-sequence master-reference="simple">
  <fo:static-content flow-name="xsl-region-after">
    <fo:block><fo:leader leader-pattern="rule" leader-length="16cm" />
    </fo:block>
    <fo:block>Página: <fo:page-number/></fo:block>
  </fo:static-content>
```

Figura 1.6.14.1 Exemplo de paginação.

Para inserir imagens em blocos de conteúdo deve-se usar o comando *external graphics* que possui o atributo *src* onde deve ser colocada a URI da imagem. A Figura 1.6.14.2 ilustra o código com a respectiva imagem resultante dele.

```
<fo:block text-align="center" space-after.optimum="10pt">
  <fo:external-graphic src="teste.png" width="417px" height="219px"/>
</fo:block>
```

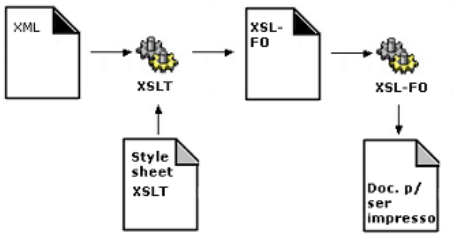

Figura 1.6.14.2. Exemplo de inserção de imagens em FO.

Para inserção de texto na página pode ser usado o próprio elemento `<fo:block>` que possui os atributos `font-size` e `font-family` com o tamanho e a fonte, `text-align` com o alinhamento do texto e `line-heigth` e `space-after.optimum`. A Figura 1.6.14.3 ilustra o código e o arquivo resultante da inserção de texto.

```
<fo:block font-size="12pt" font-family="sans-serif" line-height="15pt"
          space-after.optimum="3pt" text-align="justify">
  A figura 1 ilustra os processos pelos quais deve passar o documento fonte XML
  e o documento XSLT para que resulte em um documento XSL-FO.
  O documento FO por sua vez ao ser processado pode resultar em um documento PDF.
</fo:block>
```

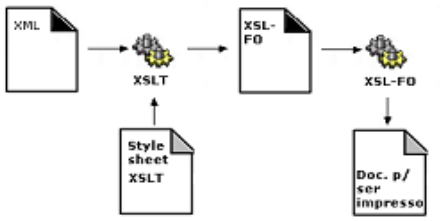

Figura 1. Diagrama sobre o processamento do XSL-FO.

A figura 1 ilustra os processos pelos quais deve passar o documento fonte XML documento XSLT para que resulte em um documento XSL-FO. O documento FO sua vez ao ser processado pode resultar em um documento PDF.

Figura 1.6.14.3. Exemplo de inserção de texto em bloco.

Para inserir lista de itens deve-se definir seu marcador e seu conteúdo, usando para tal os elementos, `<fo:list-item-label>` e `<fo:list-item-body>`. A lista de itens possui ao todo quatro elementos:

- fo:list-block (contém a lista toda)
- fo:list-item (contém cada item da lista)
- fo:list-item-label (contém o rótulo para o item da lista – em geral um `<fo:block>` com número ou caractere)
- fo:list-item-body (com o conteúdo do item da lista – em geral um ou mais objetos `<fo:block>`)

Os elementos obedecem à hierarquia ilustrada pela árvore da Figura 1.6.14.4.

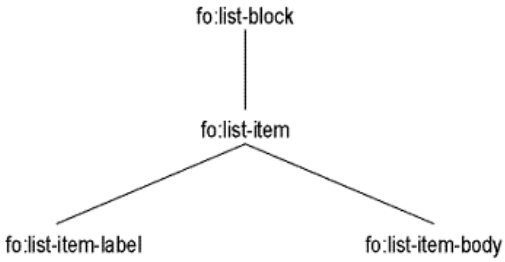

Figura 1.6.14.4 Hierarquia dos elementos em uma lista de itens [FO W3C, 2000].

A Figura 1.6.14.5 ilustra o código relativo ao processo de criação de uma lista com dois itens e a página resultante.

```
<fo:list-block >
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()" >
      <fo:block>1)</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block >Primeiro processa o XML + XSLT</fo:block>
    </fo:list-item-body>
  </fo:list-item>
  <fo:list-item>
    <fo:list-item-label end-indent="label-end()" >
      <fo:block >2)</fo:block>
    </fo:list-item-label>
    <fo:list-item-body start-indent="body-start()">
      <fo:block>Por último processa o XSL-FO</fo:block>
    </fo:list-item-body>
  </fo:list-item>
</fo:list-block>
```

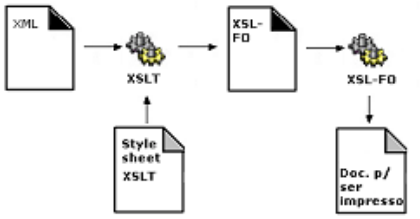

Figura 1. Diagrama sobre o processamento do XSL-FO.

A figura 1 ilustra os processos pelos quais deve passar o documento fonte XML e o documento XSLT para que resulte em um documento XSL-FO. O documento FO por sua vez ao ser processado pode resultar em um documento PDF.

1) Primeiro processa o XML + XSLT
2) Por último processa o XSL-FO

Página: 2

Figura 1.6.14.5. Exemplo de inserção de lista de itens.

Em FO é possível inserir e configurar bordas em um bloco de conteúdo. A Figura 1.6.14.6 ilustra o código e as bordas resultantes dele em um bloco de texto.

```
<fo:block font-size="12pt" font-family="sans-serif"
  line-height="1.25em" space-after="12pt" text-align="start"
  background-color="rgb(220,220,220)" start-indent="3mm" end-indent="3mm"
  padding-left="3mm" padding-right="3mm" padding-top="4mm" padding-bottom="4mm"
  border-style="dashed" border-left-width="1pt" border-right-width="1pt"
  border-top-width="2pt" border-bottom-width="2pt"
  border-top-color="rgb(140,140,140)" border-bottom-color="rgb(200,200,200)">
  Para fazer o processamento é necessário instalar programas específicos.
  FOP é uma das opções gratuitas de processadores.
  Ele faz a conversão para FO e também para PDF.
</fo:block>
```

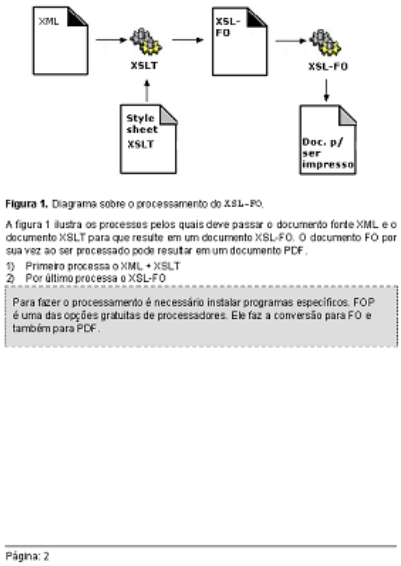

Figura 1.6.14.6. Configuração de borda em bloco de texto.

Também é possível adicionar cor de fundo e links em blocos de texto. Na Figura 1.6.14.7 há um exemplo de código para gerar esses elementos.

```
<fo:block background-color="cyan" space-after.optimum="10pt">
  Para fazer download entrem em:
  <fo:basic-link
    external-destination="http://xmlgraphics.apache.org/fop/">
    FOP
  </fo:basic-link>
</fo:block>
```

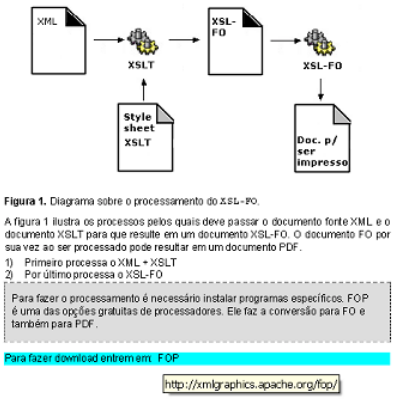

Figura 1.6.14.7. Exemplo de inserção de link e cor de fundo em bloco de texto.

Para a criação de tabelas em FO há diversos elementos e atributos que podem ser utilizados. Na Figura 1.6.14.8 observa-se um exemplo de inserção de tabela.

```
<fo:table table-layout="fixed" text-align="left" start-indent="5pt">
    <fo:table-column column-width="42mm" space-after="3pt" space-before="3pt"/>
    <fo:table-header>
      <fo:table-row>
        <fo:table-cell border-color="black" border-width="0.7mm" border-style="solid"
          height="18pt" display-align="center">
          <fo:block font-weight="bold">Objetos</fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-header>
    <fo:table-body>
      <fo:table-row>
        <fo:table-cell border-color="black" border-width="0.7mm" border-style="solid"
          height="17pt" display-align="center">
          <fo:block>fo:table-and-caption</fo:block>
        </fo:table-cell>
      </fo:table-row>
```

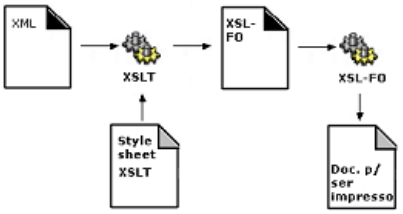

Figura 1. Diagrama sobre o processamento do XSL-FO.

A figura 1 ilustra os processos pelos quais deve passar o documento fonte XML e o documento XSLT para que resulte em um documento XSL-FO. O documento FO por sua vez ao ser processado pode resultar em um documento PDF.

1) Primeiro processa o XML + XSLT
2) Por último processa o XSL-FO

Para fazer o processamento é necessário instalar programas específicos. FOP é uma das opções gratuitas de processadores. Ele faz a conversão para FO e também para PDF.

Para fazer download entrem em: FOP

Objetos
fo:table-and-caption
fo:table-caption
fo:table-column
fo:table-header
fo:table-footer
fo:table-body
fo:table-row
fo:table-cell
fo:table

Página: 2

Figura 1.6.14.8. Exemplo de criação de tabela em FO.

Nove elementos do XSL-FO são usados para criar tabelas, sendo que quatro destes, marcados com * são opcionais:

- fo:table-and-caption
- fo:table
- fo:table-caption
- fo:table-column *
- fo:table-header *
- fo:table-footer *
- fo:table-body *
- fo:table-row

- fo:table-cell

A inserção dos elementos de tabela em XSL-FO obedece à hierarquia proposta na definição da linguagem [FO W3C, 2000]. A Figura 1.6.14.9 ilustra a representação dos objetos de formatação para tabelas.

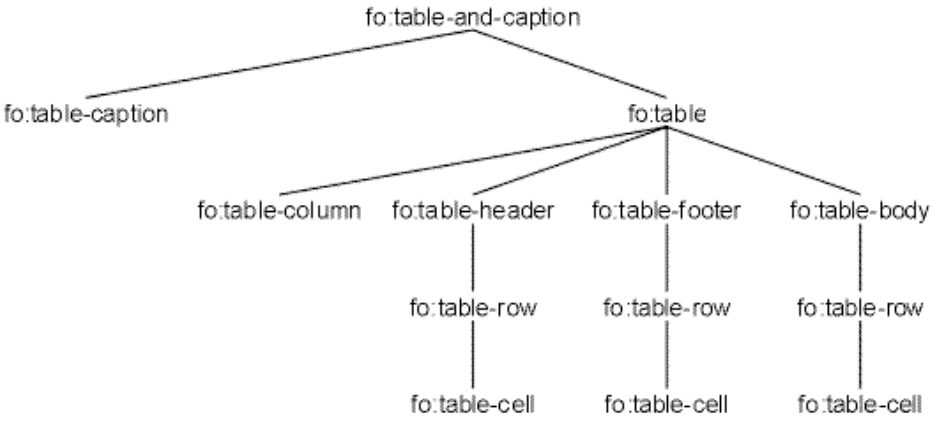

Figura 1.6.14.9: Estrutura de FO para tabelas [FO W3C, 2000].

Para inserção de nota de rodapé deve ser usado o elemento `<fo:footnote>`. Na Figura 1.6.14.10 vê-se um exemplo para inserção de nota de rodapé.

```
<fo:block>
    Conteúdo da página.
    Palavra com nota associada<fo:footnote >
        <fo:inline baseline-shift="super"  font-size="8pt"
            color="red">(1)</fo:inline>
        <fo:footnote-body >
            <fo:block font="8pt Verdana">
                1) Aqui você insere a nota de rodapé.
            </fo:block>
        </fo:footnote-body>
    </fo:footnote> E então você continua o conteúdo.
```

Figura 1.6.14.10. Exemplo de inserção de nota de rodapé em FO.

Há também a opção de quebrar o texto em colunas, usando para isto o atributo `column-count`. A Figura 1.6.14.11 exemplifica como criar um texto em 2 colunas.

```
<fo:layout-master-set>
    <fo:simple-page-master master-name="all-pages">
        <fo:region-body region-name="Content"
            margin="0.7in" column-count="2"  />
    </fo:simple-page-master>
</fo:layout-master-set>
```

Figura 1.6.14.11. Inserindo textos em 2 colunas.

1.6.15 Elementos em XSL-FO

Como a linguagem possui 56 elementos e mais de 200 atributos apenas os elementos e atributos mais relevantes foram detalhados nesta seção. A lista completa com os elementos está apresentada na Tabela 1.6.15.1 abaixo.

Tabela 1.6.15.1. Elementos FO.

fo:basic-link	fo:list-block	fo:repeatable-page-master-alternatives
fo:bidi-override	fo:list-item	fo:repeatable-page-master-reference
fo:block	fo:list-item-body	fo:retrieve-marker
fo:block-container	fo:list-item-label	fo:root
fo:character	fo:marker	fo:simple-page-master
fo:color-profile	fo:multi-case	fo:single-page-master-reference
fo:conditional-page-master-reference	fo:multi-properties	fo:static-content
fo:declarations	fo:multi-property-set	fo:table
fo:external-graphic	fo:multi-switch	fo:table-and-caption
fo:float	fo:multi-toggle	fo:table-body

fo:flow	fo:page-number	fo:table-caption
fo:footnote	fo:page-number-citation	fo:table-cell
fo:footnote-body	fo:page-sequence	fo:table-column
fo:initial-property-set	fo:page-sequence-master	fo:table-footer
fo:inline	fo:region-after	fo:table-header
fo:inline-container	fo:region-before	fo:table-row
fo:instream-foreign-object	fo:region-body	fo:title
fo:layout-master-set	fo:region-end	fo:wrapper
fo:leader	fo:region-start	

1.7. Considerações Finais

Dentre as principais vantagens de se usar a XML se destaca o fato da linguagem ser extensível e permitir intercâmbio de informações entre diferentes plataformas, além de incluir informações semânticas sobre o conteúdo. Ela atende seus objetivos iniciais, de ter sido criada para permitir uso por longos períodos de tempo e para aplicações diversas, e se caracteriza por sua estrutura hierárquica em árvore.

Para utilização do conteúdo armazenado em documentos XML foi criada a X-Path que possibilita manipular os documentos XML por meio de comandos e expressões. Com o acesso aos conteúdos dos nós tornou-se possível executar operações e utilizar os conteúdos armazenados nos documentos XML em aplicações práticas.

As linguagens de transformação foram criadas para suprir a necessidade de manipular e apresentar apropriadamente conteúdos de documentos XML. Com elas é possível gerar diferentes formatos de arquivos e consequentemente atender a diferentes necessidades. XSL-FO, por exemplo, permite gerar arquivos de áudio, apresentações, vídeos, livros e cartazes a partir de um mesmo documento fonte em XML.

A tecnologia que a XML provê atende a necessidades atuais de estruturação e manipulação de documentos web, como com a atribuição de informação semântica aos dados, independência de plataforma e possibilidade de armazenamento de grandes quantidades de informações de maneira estruturada. Embora tenha uma aplicação vasta, ainda há diversos aspectos que estão em desenvolvimento para aprimorar e estender sua utilização, buscando acompanhar o avanço das tecnologias, principalmente no relacionado a ambientes web.

Referências

[AH, 1996] Antenna House. Disponível em: http://www.antennahouse.com/ Acesso em: 03/2009.

[DOM, 2005] Especificação W3C do XML DOM. Disponível em: http://www.w3.org/DOM/ Acesso em: 03/2009.

[DOM, 2009] Tutorial sobre XML DOM. Disponível em: http://www.w3schools.com/dom/default.asp Acesso em: 03/2009.

[FO, 2001] Disponível em: http://www.xml.com/pub/a/2001/01/24/xsl-fo/index.html ?page=1 Acesso em: 03/2009.

[FO, 2002] Printing from XML: An Introduction to XSL-FO. Disponível em: http://www.xml.com/pub/a/2002/10/09/xslfo.html Acesso em: 03/2009.

[FO, 2009] Tutorial da W3C sobre XSL-FO. Disponível em http://www.w3schools.com/xslfo/ Acesso em: 03/2009.

[FOP, 2009] Processador de FO. Disponível em: http://xmlgraphics.apache.org/fop/

[FO W3C, 2000] Formatting Objects W3C Candidate Recommendation. Disponível em: http://www.w3.org/TR/2000/CR-xsl-20001121/slice6.html#fo-section Acesso em: 03/2009.

[Goldfarb and Prescod, 1998] Goldfarb, C. and Prescod P. The XML handbook. Prentice Hall. 1998.

[Namespaces, 2006] Namespaces in XML 1.0. Disponível em: http://www.w3.org/TR/xml-names/ Acesso em: 03/2009.

[PT, 1999] Passive Tex. Disponível em: http://www.ibiblio.org/bosak/conf/xmldev99/rahtz/passivetex.html Acesso em: 03/2009.

[Pitts-Moultis and Kirk, 2000] Pitts-Moultis, N. and Kirk, C. XML Black Book. Makron Books. 2000.

[RenderX, 2005] RenderX. Disponível em: http://www.renderx.com/ Acesso em: 03/2009.

[SAX, 2000] Projeto SAX. Disponível em: http://www.saxproject.org/ Acesso em: 03/2009.

[Schema, 2001] XML Schema. Disponível em: http://www.w3.org/XML/Schema Acesso em: 03/2009.

[Schema, 2001] Tutorial sobre XML Schema. Disponível em:http://www.w3schools.com/schema/ Acesso em: 03/2009.

[SMIL, 1998] SMIL (Standard Multimedia Integration Language), disponível em: http://www.w3.org/TR/1998/TR/1998/PR-smil-19980409/ Acesso em: 03/2009.

[Spencer, 1999] Spencer, P. XML Design and Implementation. Wrox Press Ltda. 1999.

[XML, 2009] Tutorial sobre XML da W3C. Disponível em: http://www.w3schools.com/xml/ Acesso em: 03/2009.

[XPath, 1999] Definição de XPath disponível em: http://www.w3.org/TR/xpath Acesso em: 03/2009.

[XPath, 2009] Tutorial sobre XPath disponível em: http://www.w3schools.com/xpath/ Acesso em: 03/2009.

[XSL, 2009] Tutorial sobre XSL da W3C. Disponível em:http://www.w3schools.com/Xsl/xsl_languages.asp Acesso em: 03/2009.

[XSL-FO, 2001] Using XSL Formatting Objects. Disponível em: http://www.xml.com/pub/a/2001/01/17/xsl-fo/index.html Acesso em: 03/2009.

[XSL W3C, 2009] Definição de XSL disponível em: http://www.w3.org/Style/XSL/

[XSLT, 2009] Tutorial da W3C sobre XSLT. Disponível em: http://www.w3schools.com/xsl/ Acesso em: 03/2009.