***Extensible Markup Language* (XML)**

Júnio César de Lima *Cedric Luiz de Carvalho*

Technical Report - RT-INF_002-05 - Relatório Técnico
June - 2005 - Junho

Instituto de Informática
Universidade Federal de Goiás
www.inf.ufg.br

Extensible Markup Language (XML)

Júnio César de Lima * junio@inf.ufg.br

Cedric Luiz de Carvalho † cedric@inf.ufg.br

Abstract. *This text introduces the XML language. It is a goal-marking language that, like HTML, includes data among tags. The XML tags are not predefined, in other words, the creator of a XML document can create their own tags. To define which tags are valid, it can be used a DTD or a XML Schema. The presentation of XML document data is possible through the use of stylesheets. XML is one of the Semantic Web technologies. The main characteristics of XML will be discussed in this text.*

Keywords: XML, XML Schema, *Namespaces*, XSL, DOM, SAX and Semantic Web.

Resumo. *Este texto apresenta uma introdução à linguagem XML, uma linguagem de meta-marcação que, como a HTML, inclui dados entre marcadores (*tags*). Os marcadores da XML não são pré-definidos, ou seja, o criador de um documento XML pode criar seus próprios marcadores. Para definir quais marcadores são válidos, pode-se usar um DTD ou um Esquema XML. A XML não foi projetada para a apresentação de dados como a HTML. Entretanto, a apresentação de dados de documentos XML é possível através do uso de folhas de estilos, que permitem transformar dados de forma que possam ser visualizados em navegadores ou impressos em mídias. Além disso, XML é uma tecnologia que faz parte da Web Semântica, a qual visa facilitar o processamento automatizado de informações disponibilizadas na Internet. Todos estes aspectos de XML, seu uso com a linguagem Java e alguns outros detalhes são discutidos neste texto.*

Palavras-Chave: XML, Esquema XML, Espaço de Nomes, XSL, DOM, SAX e Web Semântica.

1 Introdução

Documentos escritos em HTML (*Hipertext Markup Language*) [20] são legíveis para seres humanos, mas não são de fácil manipulação por computadores. A XML (*Extensible Markup Language*) é uma linguagem que permite a construção de documentos legíveis para seres humanos e que podem ser facilmente tratados por máquinas, ou seja, a XML possui um conjunto de marcadores que é mais expressivo e flexível do que a HTML, uma vez que estes marcadores podem ser definidos de acordo com as necessidades do usuário.

*Mestrando em Ciência da Computação - GEApIS/INF/UFG

†Orientador - GEApIS/INF/UFG

XML é uma meta-linguagem, ou seja, ela oferece recursos para a definição de gramáticas que caracterizam linguagens para classes de documentos específicos, com conjunto de elementos, atributos e regras de composição bem determinados. Existem duas linguagens que permitem impor regras para a estrutura dos documentos XML: DTD e Esquema XML, as quais são discutidas na Seção 5.

Duas características importantes de XML são a independência de dados e separação entre conteúdo e apresentação. Com isso, um documento XML que descreve dados pode ser perfeitamente processado por uma aplicação. Este processamento é facilitado pela ausência de instruções de apresentação.

Uma outra vantagem da linguagem é que ela é baseada em texto, logo qualquer um pode criar um documento XML com uma ferramenta de texto simples. Além disso, ela não é limitada para descrever somente dados de texto, mas também pode descrever imagens, gráficos, animações ou qualquer outro tipo de dado.

Como XML realiza somente a marcação de dados, pode-se utilizar a XSL [1] e a *XLink* [8] para realizar as apresentações e as ligações, respecticamente. Com XSL (discutida na Seção 6.2) um mesmo documento XML pode ser publicado em diferentes formatos. A *XLink* (discutida na Seção 8.1) permite fazer uma ligação entre diferentes tipos de recursos. Um *link* pode ser feito, por exemplo, entre uma letra de uma canção e um trecho de um vídeo. Além disso, *links* podem conter informações sobre os recursos associados à eles, ou seja, desempenham o papel de metadados, facilitando a busca de informações na *Web*. O RDF [13] (discutido na Seção 8.4) adiciona à XML a capacidade para se trabalhar com metadados, definindo categorias completas de dados.

XML está sendo cada vez mais utilizada, seja para a construção de arquivos de configuração, seja para o intercâmbio de dados entre aplicações na *Web*, ou estruturação e armazenamento de dados. Além disso, existem várias outras aplicações para linguagem XML. A MathML [21], por exemplo, é uma linguagem de marcação matemática utilizada para descrever expressões e fórmulas matemáticas em páginas na *Web*. Um outro exemplo é a *MusicXML*, através da qual diferentes aplicativos para tratamento de músicas podem intercambiar partituras musicas [17].

Todas estas vantagens fazem com que XML seja um arcabouço ideal para troca de dados. Reconhecendo este fato, os desenvolvedores de *software* estão integrando XML dentro de suas aplicações para ganhar em funcionalidade e interoperabilidade na *Web*. Além disso, ela está sendo cada vez mais usada na construção de base de dados. Acredita-se que no futuro, com a contínua expansão da *Web*, e o advento da Web Semântica [3], XML se tornará a linguagem universal para representação de dados. Logo, todas as aplicações serão capazes de se comunicar, uma vez que elas poderão entender os vocabulários e/ou marcações de outros documentos produzidos por outras aplicações.

Este texto tem por objetivo discutir as principais características da linguagem XML. Ele está organizado como se segue: na Seção 2, são definidos alguns conceitos básicos sobre XML. Na Seção 3, é apresentada sua sintaxe básica. Na Seção 4, se discute os analisadores sintáticos XML. Na Seção 5, são apresentados os esquemas que definem uma gramática XML (DTD e Esquema XML). Na Seção 6, se discute a apresentação de um documento XML através do uso de folhas de estilo. As duas principais linguagens para especificação de folhas de estilo, *Cascading Style Sheets* (CSS) e a *Extensible Stylesheet Language* (XSL), também são apresentadas. Já na Seção 7, é tratado sobre os Espaços de Nomes (*namespaces*) XML. Eles são usados para prevenir colisões de nomes. Na Seção 8, são mostradas algumas linguagens baseadas em XML que estão sendo estudadas e recomendadas pelo W3C. Dentre estas tecnologias, quatro são discutidas: *XLinks*, *XPointers*, *XQuery* e RDF. Na Seção 9, se discute o uso de XML com a linguagem Java. Para exemplificar o emprego de XML com Java é discutida a API JAXP.

Ela é utilizada para leitura, criação, manipulação e transformação de dados XML, através de analisadores sintáticos (*parsers*) baseados em DOM ou SAX. Exemplos são mostrados para o melhor entendimento. Finalmente, na Seção 10, são apresentadas as considerações finais.

2 Conceito de XML

Segundo Bray et al. [6], XML foi desenvolvida pelo *XML Working Group* (originalmente conhecido como *SGML Editorial Review Board*) formado sobre o patrocínio do *World Wide Web Consortium* (W3C) [24] em 1996. Este grupo foi presidido por Jon Bosak, da *Sun Microsystems*, com participação ativa do *XML Special Interest Group*, também organizado pelo W3C. Em fevereiro de 1998 tornou-se uma tecnologia recomendada pelo W3C.

XML (*eXtensible Markup Language*) é um conjunto de regras para a definição de marcadores semânticos, que dividem um documento em partes identificáveis. É uma meta-linguagem que define uma sintaxe para ser utilizada na criação de outras linguagens de marcação para um domínio específico, com estrutura e semântica próprias [11].

Além disso, XML pode também ser definida como uma linguagem de meta-marcação que, como a HTML, inclui dados entre marcadores. Entretanto, elas possuem propósitos diferentes: HTML é uma linguagem para apresentação ao passo que XML é uma linguagem para descrição de dados estruturados. Em outras palavras, os marcadores XML estão relacionados ao significado do texto delimitado por eles ao passo que os marcadores HTML especificam como os dados serão apresentados.

Devido aos marcadores XML indicarem o conteúdo e a estrutura dos dados, é possível armazenar e recuperar automaticamente dados em arquivos. Por outro lado, seres humanos também conseguem facilmente compeender um documento XML. Além disso, o conteúdo XML e os marcadores podem conter caracteres *Unicode*, permitindo que informações de várias linguagens possam ser representadas.

Como XML não fornece instruções de como o conteúdo deve ser apresentado, pode-se usar outras linguagens para tal, como a XSL (*XML Stylesheets Language*)[1]. Vale ressaltar que documentos XML são, geralmente, armazenados em arquivos texto com a extensão `.xml`, embora isto não seja um requisito. Além disso, qualquer editor de texto pode ser usado para criar um documento XML. Na Seção 3, se discute a sintaxe básica, ou seja, como definir um documento XML.

3 Sintaxe básica

Um criador de um documento XML pode definir novas marcações para indicar a estrutura que melhor representa as informações que serão disponibilizadas. Por exemplo, a estrutura **<livro> ... </livro>** pode ser usada para descrever os dados referentes a um livro. A descrição destas informações é feita de maneira estruturada e hierárquica. A hierarquia de um documento XML é semelhante à uma árvore, ou seja, os documentos possuem um único elemento raiz que contém todos os outros elementos, chamados de elementos filhos.

XML permite que os usuários definam a sua própria linguagem de marcação adaptada aos seus requisitos. O Código 1 mostra um exemplo de um documento XML.

[1] Discutida na Seção 6.2.

Código 1 Código XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE Agenda SYSTEM "C:\Mestrado\Biblioteca.dtd">
3 <bibliografia>
4   <livro>
5     <titulo>Verdes Campinas</titulo>
6     é um livro muito bom, escrito por
7     <autor webid="http://exemplo.com/#OP5">Olimpio Pereira</autor> .
8   </livro>
9 </bibliografia>
```

A linha 1 do Código 1 é obrigatória em todos os documentos. Ela especifica a versão da XML (Versão 1.0, neste caso) que está sendo usada e também a codificação dos caracteres (UTF-8, neste caso) [6]. A declaração de uma gramática[2] (discutida em detalhes na Seção 5), feita na linha 2, é usada para validação de um documento. Ela deve aparecer antes do primeiro elemento ou elemento raiz (`bibliografia`).

Os **elementos** são os componentes básicos da XML[3]. Eles são criados pelo autor de um documento XML. O nome do elemento é delimitado pelas marcações: $<$ e $>$. As marcações, juntamente com o texto entre elas formam o elemento. O conteúdo de um elemento é delimitado pelo marcador de início (**<nome>**) e o marcador de fim (**</nome>**). Para cada marcador de início deve existir umo marcador de fim com o mesmo nome.

É importante observar a diferença entre marcação de texto e caracteres de dados. Marcação de texto é o texto entre $<$ e $>$. Por exemplo, `titulo` é uma marcação de texto, no Código 1. `Caracteres de dados` é o texto entre o marcador de início e o marcador de fim. Por exemplo, `Verdes Campinas`, no Código 1, representa caracteres de dados, pois está entre o marcador de início **<titulo>** e o marcador de fim **</titulo>**. Elementos filhos são considerados marcação e não caracteres de dados.

O conteúdo de um elemento pode ser um texto simples e/ou outros elementos. Por exemplo, o conteúdo do elemento `bibliografia` é o elemento `livro`. O conteúdo do elemento `titulo` é um texto simples, "Verdes Campinas". Já o conteúdo do elemento `livro` é formado pelos elementos `titulo` e `autor` e pelo texto simples "é um livro muito bom escrito por".

Em alguns casos um elemento pode não possuir conteúdo. Um elemento sem conteúdo pode ser representado por **<autor> </autor>** ou por **<autor/>**.

Os elementos `bibliografia`, `livro`, `titulo` e `autor` refletem uma estrutura particular associada a este contexto. Os marcadores permitem que qualquer pessoa interprete o seu conteúdo. Por exemplo, o elemento **<titulo>** Verdes Campinas **</titulo>** consiste do marcador de início **<titulo>**, o conteúdo do elemento e o marcador final **</titulo>**. Com isso, é possível interpretar claramente que o conteúdo do marcador é o título de alguma coisa. Como os elementos `titulo` e `autor` estão aninhados dentro do elemento `livro`, se percebe que um livro está sendo descrito, logo "Verdes Campinas" é o título de um livro cujo autor é "Olimpio Pereira".

Em alguns casos, um marcador de início pode ter informações adicionais sobre um elemento, chamadas de **atributos**. Um elemento pode ter um ou vários atributos. Um atributo descreve características ou propriedades dos elementos e consiste de um nome, um sinal de igual e um valor entre aspas. Os atributos devem ser descritos no marcador de iní-

[2] O uso de uma gramática é opcional.

[3] Neste exemplo, os elementos são: `bibliografia`, `livro`, `titulo` e `autor`.

cio. Por exemplo, o marcador de início do elemento `autor` (linha 7) contém o atributo `webid="http://exemplo.com/#OP5"`.

No processamento de um documento é freqüentemente útil a identificação da linguagem cujo conteúdo de um elemento ou atributo é escrito. Um atributo especial chamado `xml:lang` pode ser inserido em documentos para especificar a linguagem usada. Os valores do atributo, que são identificadores da linguagem, são definidos por Bray et al. [6]. Na Subseção 3.1, são apresentadas algumas regras que devem ser seguidas na criação de um documento XML válido.

3.1 Regras

Todo documento XML deve ser bem formado. Isto significa que ele deve ser sintaticamente correto, ou seja, deve seguir certas regras. Algumas destas regras são discutidas a seguir:

- Os nomes dos elementos devem estar de acordo com as seguintes regras:
  1. iniciar com uma letra ou o caractere sublinha "_";
  2. o resto do nome deve consistir de:
     - letras, dígitos, o caractere sublinha "_", ponto final "." ou um hífen "−";
  3. podem ter qualquer tamanho;
  4. a seqüência de caracteres `xml` é uma palavra reservada e não pode ser usada para dar nome aos elementos;
- Os nomes dos atributos seguem a mesma regra para nomes de elementos;
- O valor do atributo deve estar entre aspas;
- Todo marcador de início deve ter umo marcador de fim com o mesmo nome. Por exemplo:
  1. **<casa> ... </casa>** é bem formado;
  2. **<casa> ... </aula>** não é bem formado;
- Os elementos devem estar aninhados dentro de outro elemento e não sobrepostos. Por exemplo:
  1. **<bibliografia> <livro> </livro> </bibliografia>** é bem formada;
  2. **<bibliografia> <livro> </bibliografia> </livro>** não é bem formada;
- Os documentos XML podem ser visualizados como uma árvore. Assim, um documento XML deve possui um elemento raiz que contenha o restante dos elementos aninhados. Por exemplo, o elemento raiz no Código 1 é `bibliografia`.
- XML diferencia letras maiúsculas e minúsculas. Por exemplo, o elemento **<livro>** é diferente do elemento **<Livro>**;
- Qualquer caractere pode ser usado em um documento XML, entretanto, os caracteres `&`, <, >, ' e " são caracteres reservados e não podem ser usados como caracteres de dados;
- Espaço em branco, tabulações e retorno de carro são chamados de caracteres de espaço em branco. Uma aplicação pode considerar caracteres de espaço em branco como sendo significantes (preservados pela aplicação) ou insignificantes (não preservados pela aplicação) [7]. Dependendo da aplicação, caracteres de espaço em branco insignificantes podem ser removidos ou substituídos por um único espaço em branco;

- Comentários podem aparecer em qualquer lugar de um documento fora das marcações [6] e também podem aparecer dentro de um DTD ou Esquema XML. Como um comentário não faz parte de um documento, um analisador sintático pode ignorar os comentários. Contudo, dependendo do analisador sintático, pode ser possível que uma aplicação possa recuperar um texto comentado. Um comentário inicia com o sinal $<!--$ e termina com o sinal $-->$. Um exemplo de um comentário é mostrado a seguir:

  $<!--$ `Exemplo de um comentário` $-->$

3.2 Seções CDATA

As seções CDATA podem conter texto, caracteres reservados e caracteres de espaço em branco. Caracteres de dados em uma seção CDATA não são processados por um analisador sintático XML. O Código 2 mostra um exemplo de uma seção CDATA.

Código 2 Um seção CDATA.

```
1  <questao id="339">
2    <introducao>O código abaixo produz um erro de compilação?
3      <java><![CDATA[
4         class Final {
5            public int i = 0;
6            public void M() {
7              final Final f = new Final();
8              f.i = 2;
9            }
10        }
11     ]]></java>
12   </introducao>
13   <resposta valor="f"/>
14 </questao>
```

O marcador **<java><![CDATA[** do Código 2 indica o uso de uma seção CDATA contendo um bloco de texto (um techo de código Java). Um analisador sintático XML não deve tratar este bloco como caracteres de dados ou marcação, ou seja, o analisador sintático deve ignorar este bloco de texto. A seção CDATA é muito utilizada por elementos destinados a conterem *scripts* de linguagens de programação que fazem uso de caracteres reservados da XML.

Uma vez que a seção CDATA não é analisada, ela pode conter qualquer texto, incluindo caracteres de dados normalmente reservados para XML, como $<$, $>$ e `&`. Entretanto, seções CDATA não podem conter o texto `]]>`, pois ele é usado para sinalizar o término de uma seção CDATA. A Subseção 3.3, a seguir, discute as entidades, que são unidades de texto muito utilizadas na criação de documentos XML.

3.3 Entidades

As entidades são blocos de construção de documentos XML. Uma entidade é uma unidade de texto que pode ser tão simples quanto um único caractere ou tão complexa quanto um documento inteiro. Elas são incluídas em um documento XML através de uma **referência de entidade**. A criação de uma referência de entidade é feita em um DTD. Com uma referência de

entidade, um analisador sintático pode pesquisar a entidade referenciada e colocar seu conteúdo no local da referência.

Um uso simples e comum de uma referência de entidade é para escrever caracteres reservados em um documento XML. Por exemplo, o caractere < é reservado em XML, uma vez que ele faz com que um analisador sintático analise os caracteres seguintes como marcação. A referência de entidade para este caractere é `"<"`. Então, quando um analisador sintático encontrar esta referência de entidade, ele substitui pelo caractere <.

As entidades também podem ser usadas para incluir arquivos inteiros de texto XML ou agir como taquigrafia [4] útil para trechos usados com freqüência.

As entidades podem ser declaradas em um arquivo DTD separado ou dentro do próprio documento XML. Em documentos que seguem a linguagem RDF/XML, as entidades são, geralmente, declaradas dentro do próprio documento, uma vez que RDF/XML não necessita ser validada [13]. O Código 3 exemplifica a declaração de entidades.

Código 3 Declaração de entidades.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE rdf:RDF [
3    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
5    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
6  ]>
7
8  <rdf:RDF
9    xmlns:rdf = "&rdf;"
10   xmlns:rdfs = "&rdfs;"
11   xmlns:xsd = "&xsd;">
12
13   ... declarações ...
14
15 </rdf:RDF>
```

O Código 3 mostra, nas linhas de 2 a 6, a declaração de um DTD que cria três entidades: `rdf`, `rdfs` e `xsd`. Quando um analisador sintático for analisar este documento e encontrar, por exemplo, a entidade `&rdf;`, ele substituirá esta entidade por `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Este é um exemplo de uma taquigrafia. Na Seção 4, são discutidos os analisadores sintáticos XML.

4 Analisadores Sintáticos (*Parsers*) XML

Um *software* chamado de analisador sintático XML (ou processador XML) é requerido para processar um documento XML. Um analisador sintático é, usualmente, um utilitário que permite a obtenção de elementos pela relação de pais e filhos [16]. Um analisador sintático XML lê um documento XML, verifica sua sintaxe, relata possíveis erros e permite acessar o conteúdo do documento [7]. A Figura 1 mostra como funciona um analisador sintático XML.

Pela Figura 1 se percebe que um sistema baseado em XML envolve pelo menos três componentes: um documento XML (com as marcações), um processador XML (que divide o

[4] Escrita abreviada e simplificada.

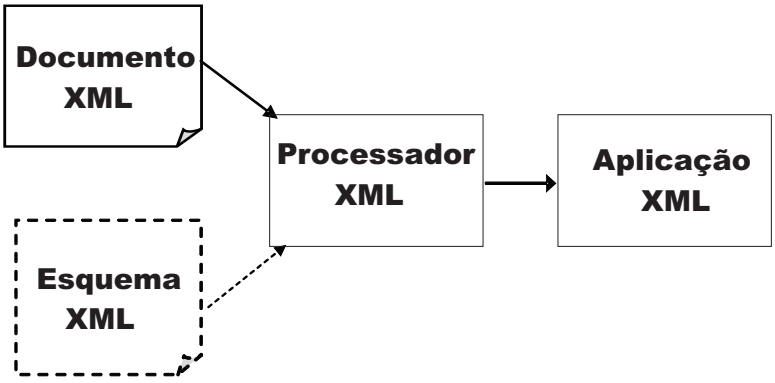

Figura 1: O funcionamento de um analisador sintático XML.

documento em "pedaços" de marcação e de dados de caracteres) e uma aplicação XML (que utiliza as informações enviadas pelo analisador sintático).

Um analisador sintático XML, como dito anteriormente, é usado para ler um documento XML e permitir acesso ao seu conteúdo e sua estrutura. É assumido que um analisador sintático XML está fazendo seu trabalho como representante de uma aplicação XML [6].

Um esquema XML (discutido na Seção 5) que defina o vocabulário do documento XML é opcional. Um documento criado de acordo com uma sintaxe (DTD ou XML *Schema*) é um documento XML válido.

Os analisadores sintáticos podem utilizar o *Document Object Model* (DOM) e/ou o *Simple API for XML* (SAX) para acessar o conteúdo de um documento XML usando uma linguagem de programação, tal como Java, *Python* ou C.

DOM apresenta um documento XML como uma hierarquia de elementos que são acessados como uma árvore. SAX oferece várias possibilidades para usar um documento XML, uma vez que cada elemento é analisado e enviado para a aplicação como resposta a um evento. Logo, pode-se dizer que DOM vincula a análise de um documento a uma visão de uma árvore e SAX pode ser usada para criar sua própria visão [16]. Detalhes de DOM e SAX são discutidos na Seção 9.1.

A maioria dos analisadores sintáticos XML podem ser conseguidos sem nenhum custo. Algumas aplicações, tais como *Microsoft Internet Explorer* 5 (IE5) e versões superiores, possuem um analisador sintático XML embutido. Na Seção 5, é discutido sobre a criação de gramáticas para validar documentos XML.

5 Definição da Estrutura

Um documento XML pode, opcionalmente, incluir um outro documento, também chamado de esquema, que define uma gramática, a qual define restrições na estrutura de um documento e diz quais marcadores são válidos juntamente com seus atributos.

O documento esquema também é chamado de validador. Um documento XML é dito válido se ele segue uma gramática ou esquema. O uso de um documento esquema é opcional. Existem dois tipos de esquemas XML: DTD e Esquema XML, os quais são apresentados na duas próximas Subseções.

5.1 DTD - *Document Type Definition*

Uma declaração de tipo de documento XML contém ou aponta para declarações de marcação que provê uma gramática para uma classe de documentos. Uma declaração de marcação é a declaração de um tipo de elemento, uma lista de atributos e uma entidade [6]. Esta gramática é conhecida como definição de tipo de documento ou DTD (*Document Type Definition*). O DTD é um esquema que faz parte da especificação do XML 1.0 [15].

As declarações em um DTD especificam quais elementos e atributos podem aparecer em um documento XML correspondente a este DTD. O relacionamento entre estes elementos e atributos (qual elemento pode ser aninhado dentro de um outro elemento ou quais atributos podem aparecer em determinados elementos) e quais elementos e atributos são opcionais ou não.

Segundo Manola e Miller [13], um documento XML pode apontar para um DTD localizado em outro documento (chamado de subconjunto externo, que pode ser usado para permitir que uma gramática possa ser compartilhada entre múltiplos documentos), o DTD pode estar no próprio documento (chamada de subconjunto interno) ou o documento XML pode ter ambos, ou seja, subconjunto externo e interno. O Código 4 mostra a declaração de um DTD.

Código 4 Declaração de um DTD.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!ELEMENT Agenda (Itens)+>
3 <!ELEMENT Itens(#PCDATA)>
4 <!ATTLIST Itens
5     Dia CDATA #REQUIRED
6     Hora CDATA #REQUIRED
7     Minutos CDATA #REQUIRED
8     NomeCompromisso CDATA #REQUIRED>
```

O DTD, mostrado no Código 4, define dois elementos: `Agenda` e `Itens`. O elemento raiz é o elemento `Agenda`, linha 2. O conteúdo do elemento `Agenda` é outro elemento (o elemento `Itens`) que deve ser aninhado dentro do elemento `Agenda`. O símbolo **+**, associado ao elemento `Itens` na linha 2, indica que podem ocorrer um ou mais elementos `Itens` aninhados no elemento `Agenda`.

O conteúdo do elemento `Itens`, linha 3, é do tipo `#PCDATA`, ou seja, dados de caracteres. O elemento `Itens` contém quatro atributos: `Dia`, `Hora`, `Minutos` e `NomeCompromisso`. O valor dos atributos é `CDATA`, ou seja, pode conter qualquer caractere de texto, exceto <, >, &, ' e ". Além disso, estes atributos possuem a propriedade `#REQUIRED`, que especifica que os atributos são obrigatórios. O Código 5 mostra um documento XML que é válido segundo o DTD apresentado no Código 4.

Código 5 Um documento XML que é validado por um DTD.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE Agenda SYSTEM "C:\Mestrado\Agenda.dtd">
3
4 <Agenda>
5   <Itens Dia="14" Hora="10" Minutos="15" NomeCompromisso="Reunião"/>
6   <Itens Dia="19" Hora="19" Minutos="45" NomeCompromisso="Cinema"/>
7 </Agenda>
```

Na declaração da linha 2, do Código 5, a palavra-chave `DOCTYPE` indica a declaração de um DTD. Ela é seguida pelo nome do elemento raiz (`Agenda`), pela palavra-chave `SYSTEM`[5] e pelo nome do arquivo que contém o DTD (`"C:\Mestrado\Agenda.dtd"`).

DTDs externos são especificados usando-se as palavras-chave `SYSTEM` ou `PUBLIC`. No exemplo foi usado `SYSTEM`. A palavra-chave `PUBLIC` é usada para indicar que o DTD é amplamente usado. Geralmente, este tipo de DTD fica acessível em localizações bastante conhecidas. O Código 6 mostra um documento XML que contém um DTD interno.

Código 6 Documento XML com um DTD interno.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE Agenda [
3     <!ELEMENT Agenda (Itens)+>
4     <!ELEMENT Itens(#PCDATA)>
5     <!ATTLIST Itens
6       Dia CDATA #REQUIRED
7       Hora CDATA #REQUIRED
8       Minutos CDATA #REQUIRED
9       NomeCompromisso CDATA #REQUIRED>
10 ]
11
12 <Agenda>
13   <Itens Dia="14" Hora="10" Minutos="15" NomeCompromisso="Reunião"/>
14   <Itens Dia="19" Hora="19" Minutos="45" NomeCompromisso="Cinema"/>
15 </Agenda>
```

No Código 6, tudo que está declarado entre os colchetes (“[” e “]”) constitui a declaração do DTD interno. Pode-se perceber que os documentos dos Códigos 5 e 6 seguem a especificação de um DTD, logo, estes documentos XML são ditos serem válidos.

Se um analisador sintático recebe um documento XML que segue um DTD, ele pode processar o documento de acordo com as regras especificadas no DTD. Por exemplo, dado o DTD `Agenda.dtd`, um analisador sintático poderá conhecer a estrutura e o tipo do conteúdo de qualquer documento baseado neste DTD. Se o analisador sintático for de validação ele poderá saber que um documento é ou não válido segundo o DTD. Na Subseção 5.2, é apresentado o Esquema XML, que é uma linguagem significativamente mais poderosa do que o DTD.

5.2 Esquema XML

Um DTD não pode ser manipulado da mesma maneira de um documento XML, uma vez que ele não é um documento XML. Além disso, um DTD descreve a estrutura de um documento XML, não o contéudo dos seus elementos. Existe uma outra linguagem de esquema recomendada pelo W3C que é significativamente mais poderosa do que o DTD, a linguagem Esquema XML.

Esquemas XML são uma alternativa para validação de documentos XML. Um Esquema XML é também um documento XML. A melhor forma de se discutir sobre os Esquemas XML é através de exemplos. O Código 7 mostra um documento XML que está associado a um Esquema.

[5]Que indica que o DTD é localizado em um arquivo externo, ao invés de ser localizado no próprio documento.

Código 7 Um documento XML que é validado por um Esquema XML.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <venda id="159798756"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:noNamespaceSchemaLocation="C:\XML\EsquemaUFG.xsd">
6    <empresa>UFG</empresa>
7    <contato>João de Deus</contato>
8    <produto>
9       <nome>MSSQL Server 2000</nome>
10      <idproduto>MS20005489</idproduto>
11      <quantidade>25</quantidade>
12   </produto>
13 </venda>
```

O documento do Código 7 é um documento XML comum. Entretanto, o segundo atributo do elemento `venda` ainda não foi discutido. O atributo da linha 4, `xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance`, declara um Espaço de Nomes[6], com o prefixo `xsi`, para o Esquema XML. O atributo da linha 5, `xsi:noNamespaceSchemaLocation`, recebe como valor o endereço do Esquema XML, `C:\XML\EsquemaUFG.xsd`. Quando um analisador sintático for analisar este documento e encontrar o atributo `xsi:noNamespaceSchemaLocation`, ele saberá que o valor deste atributo contém o endereço de um Esquema XML que deve ser usado para validar o documento XML.

Um Esquema XML pode utilizar dois atributos para definir os esquemas associados ao documento: localização de um esquema sem um Espaço de Nomes alvo e localização de um esquema com um Espaço de Nomes alvo. O atributo usado para localização de um esquema sem um Espaço de Nomes alvo é o atributo `xsi:noNamespaceSchemaLocation`, usado no Código 7 (linha 5). Para referenciar esquemas com um Espaço de Nomes alvo é usado o atributo `xsi:schemaLocation`. Detalhes sobre estes atributos estão em [18].

A seguir se discute como construir um Esquema XML para validar o documento apresentado no Código 7. A definição do esquema deve iniciar com o seguinte código:

```
<?xml version="1.0"encoding="uft-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

A declaração se inicia com a definição de um Espaço de Nomes para definição da Esquema XML. O esquema para o documento é iniciado com o elemento `venda`. O seu conteúdo é constituído de elementos aninhados (filhos), logo, ele é definido como um Tipo Complexo (`complexType`). Os elementos que possuem conteúdo com texto simples são definidos como Tipos Simples (`simpleType`). A lista de filhos do elemento `venda` é descrita pelo elemento `sequence`.

```
 <xsd:element name="venda">
  <xsd:complexType>
   <xsd:sequence>
```

Os elementos `empresa` e `contato` são tipos simples porque eles não têm atributos e nem elementos filhos. O tipo `xsd:string` possui o prefixo `xsd`, do Espaço de Nomes associado com o Esquema XML, indicando um tipo de dado predefinido pelo Esquema XML.

[6] Discutido na Subseção 7

```
<xsd:element name="empresa"type="xsd:string"/>
<xsd:element name="contato"type="xsd:string"/>
```

O elemento `produto` é um tipo complexo e é definido como se segue:

```
<xsd:element name="produto"minOccurs="0"maxOccurs="unbounded">
 <xsd:complexType>
  <xsd:sequence>
```

A ocorrência (ou cardinalidade) do elemento `produto` foi definida. O atributo `minOccurs="0"` (cardinalidade mínima) diz que o elemento pode ocorrer, no mínimo, zero vezes, ou seja, é opcional. O atributo `maxOccurs="unbounded"` (cardinalidade máxima) diz que o elemento pode aparecer, no máximo, ilimitadas vezes. A lista dos filhos do elemento `produto` é definida a seguir:

```
 <xsd:element name="nome"type="xsd:string"/>
 <xsd:element name="idproduto"type="xsd:string"
              minOccurs="0"maxOccurs="unbounded"/>
 <xsd:element name="quantidade"type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

A seqüência dos filhos do elemento `venda` está completa.

```
 </xsd:sequence>
```

Por fim, são declarados os atributos de um elemento; no caso deste exemplo, o atributo `id` do elemento `venda`.

```
 <xsd:attribute name="id"type=xsd:string">
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Segundo Rambhia [16], por recomendação do W3C, a declaração de atributos de um elemento deve ser colocada no final da declaração do elemento, ou seja, depois de `</xsd:sequence>`. O esquema completo é mostrado no Código 8.

Código 8 Um documento Esquema XML completo.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="venda">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="empresa" type="xsd:string"/>
    <xsd:element name="contato" type="xsd:string"/>
    <xsd:element name="produto" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
       <xsd:sequence>
         <xsd:element name="nome" type="xsd:string"/>
         <xsd:element name="idproduto" type="xsd:string"
                      minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="quantidade" type="xsd:string"/>
       </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string">
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Pode-se perceber que, diferentemente de um DTD, um Esquema XML é um documento XML. Logo, ele pode ser manipulado como um documento XML comum. Além das declarações discutidas anteriormente, podem existir várias outras declarações no Esquema XML. Por exemplo, o Código 9 mostra uma restrição sobre o elemento `mes`. A definição do elemento `mes` mostra uma forma conveniente para definir `simpleType` no Esquema XML. O marcador `restriction` foi aplicado ao elemento `mes` para dizer que ele deve ser um tipo *string* e ter um tamanho máximo `12`.

Na Seção 6, discute-se a apresentação de um documento XML através do uso de folhas de estilo. As duas principais linguagens de folhas de estilo, *Cascading Style Sheets* (CSS) e a *Extensible Stylesheet Language* (XSL), também são apresentadas.

Código 9 Trecho de um documento Esquema XML definindo uma restrição.

```
<xsd:simpleType name="mes">
  <xsd:restriction base="xsd:string">
     <xsd:maxLength value="12"/>
  <xsd:restriction/>
<xsd:simpleType/>
```

6 Estilo de Apresentação

As marcações XML especificam somente o conteúdo do documento, diferentemente de HTML, que não diz nada sobre os dados, mas somente como eles devem ser apresentados.

As informações sobre como um documento XML será apresentado quando for impresso ou visualizado em um navegador ficam armazenadas em uma folha de estilo (*style sheet*).

Diferentes folhas de estilos podem ser usadas para um mesmo documento. Pode-se, por exemplo, usar uma folha de estilo que especifica uma fonte média para impressão, outra com fonte pequena para apresentação na tela de um celular e outra com fonte grande para apresentação em uma tela grande.

O princípio das linguagens de folha de estilo está em definir uma sintaxe para identificar partes específicas do conteúdo de documentos e um conjunto de estilos ou ações que devem ser realizadas sobre as partes identificadas. As duas linguagens de folhas de estilo mais usadas atualmente são: *Cascading Style Sheets* (CSS) e a *Extensible Stylesheet Language* (XSL), as quais são discutidas nas duas próximas Subseções.

6.1 *Cascading Style Sheets* - CSS

A CSS é uma linguagem de folha de estilo simples, originalmente projetada para ser usada com HTML, mas que se aplica tanto a documentos XML quanto a documentos HTML. Ela permite especificar o estilo de uma página de elementos, como espaçamento, margem e fonte, separadamente da estrutura do documento. Esta separação entre estrutura e apresentação facilita a manutenção do dados e dos estilos de apresentação.

As folhas de estilo CSS são ditas em cascatas porque permitem que estilos sejam definidos por usuários, autores e agentes (navegadores) [7]. Os estilos definidos por autores têm maior precedência sobre os estilos definidos pelos usuários e os estilos definidos pelos usuários têm maior precedência sobre os estilos definidos pelos agentes.

A associação de uma folha de estilo a um documento XML é realizada através da inserção, no conteúdo de um documento XML, de uma instrução de processamento que referencia a folha de estilo a ser aplicada. Por exemplo, assumindo-se que a folha de estilo esteja armazenada no arquivo `exemplo.css`, no mesmo diretório em que encontra-se o conteúdo do documento, a seguinte instrução de processamento deve ser acrescentada ao documento:

```
<?xml-stylesheet type="text/css"href="exemplo.css"?>.
```

Maiores detalhes sobre CSS estão em [19]. Na Subseção 6.2, é apresentada a XSL, que é uma linguagem de folha de estilo mais poderosa do que CSS.

6.2 *Extensible Stylesheet Language* - XSL

XSL é uma família linguagens recomendadas para transformação e apresentação de documentos XML. Segundo Berglund [1], XSL consiste de três linguagens descendentes de XML descritas em três recomendações feitas pelo W3C:

- XSLT (XSL *Transformations*): linguagem para descrever como transformar um documento XML em outro documento.
- *XPath* (XML *Path Language*): linguagem usada pelo XSLT para acessar ou referenciar partes específicas de um documento XML (*XPath* também é usada pela especificação da XML *Linking*[7]).
- XSL-FO (XSL *Formatting Objects*): um vocabulário XML para especificação da formatação da semântica.

[7] Discutida na Subseção 8.1.

Uma folha de estilo XSL especifica a apresentação de uma classe de documentos XML descrevendo como uma instância desta classe é transformada em um documento XML que usa o vocabulário de formatação [1], ou seja, dada uma classe de documentos XML ou arquivos de dados, um projetista usa uma folha de estilo XSL para expressar suas intenções sobre como este conteúdo estruturado deve ser apresentado.

Um processador de folha de estilo XSL aceita um documento ou dados em XML e uma folha de estilo XSL e produz a apresentação destes dados de acordo com que foi projetado pelo projetista da folha de estilo. Segundo Berglund [1], há dois aspetos deste processo de apresentação: 1) construção de uma árvore de resultado em relação a uma árvore XML fonte e 2) interpretação da árvore de resultado para produzir a apresentação satisfatória para uma tela de um computador, impressora ou qualquer outro dispositivo. O primeiro aspecto é chamado de **árvore de transformação** e o segundo é chamado de **formatação**. O processo de formatação é executado pelo formatador, que pode ser um software dentro de um navegador.

Formatação inclui semântica de formatação na árvore de resultado, a qual é expressa em termos de um catálogo de classes de objetos de formatação (os nós da árvore de resultado).

A árvore de transformação constrói a árvore de resultado. Em XSL, esta árvore é chamada de **árvore elemento e atributo** [1]. Nela, um objeto de formatação é representado como um elemento XML, com as propriedades representadas por um conjunto XML de pares atributo-valor. O conteúdo de um objeto de formatação é o conteúdo de um elemento XML. Árvore de transformação é discutida na Seção 6.2.2.

A folha de estilo XSL é usada na árvore de transformação. Uma folha de estilo contém um conjunto de regras para construção da árvore. Segundo [1], as regras da árvore de transformação possuem duas partes: um padrão que é comparado com os elementos da árvore fonte e um molde (*template*) que constrói uma porção da árvore de resultado. Isto permite que uma folha de estilo seja aplicável para uma ampla classe de documentos que possuem árvores fontes similares.

A formatação interpreta uma árvore de resultado na forma da árvore de objeto de formatação para produzir a apresentação projetada pelo projetista da folha de estilo. XSL-FO (XSL *Formatting Objects*) é discutido na Seção 6.2.3.

A XSL foi projetada para dar aos projetistas controle sobre as características necessárias quando documentos são paginados [8], como também para fornecer um quadro ("*frame*") equivalente baseado na estrutura para navegação na *Web*. Para executar este controle, XSL estende o conjunto de objetos de formatação e propriedades de formatação [1]. Além disso, a seleção de componentes fontes XML que podem ser nomeados são baseados em XSLT (discutido na Subseção 6.2.2) e *XPath* (discutido na Subseção 6.2.1). *XPath* oferece um mecanismo poderoso para seleção de componentes XML (atributos e elementos, por exemplo).

6.2.1 XML *Path Language - XPath*

XML permite a descrição de dados de uma forma flexível e eficiente com a utilização de marcadores descritivos. Entretanto, ela não oferece meios para localizar pedaços de dados dentro de um documento. Com o uso *XPath* pode-se localizar partes específicas de um documento XML de maneira eficiente [7].

XPath é uma linguagem de expressão que permite o processamento de valores de acordo com um modelo de dados [2]. Este modelo contém uma árvore de representação de um documentos XML como também de valores atômicos tais como inteiros, *strings* e *booleanos*. O resultado de uma expressão *XPath* pode ser uma seleção de nós de um documento ou um valor atômico.

[8] Paginar: arranjar graficamente as páginas de um livro, registro etc..

Em *XPath*, um documento XML é visualizado, conceitualmente, como uma árvore, onde cada parte do documento é representada como um nó. Existem sete tipos de nós: raiz, elemento, atributo, texto, comentário, instrução de processamento e Espaço de Nomes.

Segundo Harold [11], uma árvore *XPath* possui um nó raiz que contém todos os outros nós da árvore. O nó raiz e os nós elementos contém uma lista ordenada de todos os nós filhos. Cada nó, exceto o nó raiz, tem um nó pai e todos os nós pai possuem um nó filho ou descendente. Os nós que podem ter nós filhos são os seguintes: elemento, comentário, texto e instrução de processamento.

Para localizar uma parte de um documento na estrutura de árvore definida em *XPath* é usado um **caminho de localização**. Um caminho de localização é uma expressão que especifica como navegar em uma árvore *XPath* de um nó para outro. Um caminho de localização é composto de passos de localização, sendo que cada um é composto de um eixo (*axis*), um nó teste e um nó predicado, opcional [7]. Detalhes e exemplos de *XPath* estão em Berglund et al. [2].

6.2.2 XSL *Transformations* - XSLT

XSLT é a parte mais importante da família XSL. Ela é usada para transformar um documento XML em outro documento XML ou outro tipo de documento que é reconhecido por um *browser*, como o HTML e o XHTML.

Uma transformação expressa em XSLT descreve regras para transformação de zero ou mais árvores fontes em zero ou mais árvores resultado [12]. A transformação é executada por um conjunto de modelos de regras. Um modelo de regra associa um padrão que emparelha nós em um documento fonte com uma seqüência.

Segundo Grosso et al. [10], a XSLT pode também adicionar novos elementos na saída de um arquivo ou remover elementos. Além disso, ela também pode reestruturar e classificar elementos, testar e tomar decisões sobre quais elementos serão visualizados, por exemplo.

Uma forma para descrever o processo de transformação é dizer que XSLT transforma uma árvore XML fonte em uma árvore XML resultado. Segundo Berglund [1], as regras de construção da árvore possuem duas partes: um padrão que é comparado com os elementos da árvore fonte e um molde que constrói uma porção da árvore de resultado.

No processo de transformação, XSLT usa *XPath* para definir partes de um documento fonte que se combina com um ou mais moldes pré-definidos. Quando uma combinação é encontrada, XSLT transforma a parte da combinação do documento fonte no documento resultado.

A XSL usa XSLT e *XPath* para construir uma árvore e um padrão de seleção, permitindo assim um alto grau de controle sobre como os conteúdos fontes são apresentados e quais propriedades são associadas com os conteúdos [1]. Por exemplo, o padrão de *XPath* permite a seleção de uma cadeia de caracteres ou o k-ésimo nó texto em um parágrafo. Isto permite que os usuários tenham uma regra que faz com que todo o terceiro parágrafo apareça em negrito, por exemplo. Além disso, propriedades podem ser associadas com um conteúdo baseado em um valor numérico de um conteúdo ou atributo. Isto permite ter uma regra de estilo que faz com que um valor negativo apareça em `"vermelho"` e um valor positivo em `"verde"`, por exemplo.

O Código 10 mostra um documento XML, (`exemplo.xml`), que será transformado em XHTML [9]:

[9] Detalhes sobre XHTML estão em [20].

Código 10 Um documento XML com conteúdo sobre livros.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <?xml-stylesheet type="text/xsl" href="exemplo.xsl"?>
3
4  <biblioteca>
5    <livro>
6      <titulo>Verdes Campinas</titulo>
7      <autor>Olimpio Pereira</autor>
8      <editora>Campo Formoso</editora>
9      <preco>30.00</preco>
10   </livro>
11 </biblioteca>
```

O arquivo deve ter uma referência XSL para ligar o arquivo XML ao arquivo XSL. A linha 2, `<?xml-stylesheet type="text/xsl"href="exemplo.xsl"?>`, faz esta referência.

O Código 11 mostra uma folha de estilo XSL (`exemplo.xsl`), com um modelo de transformação, que transforma o documento `exemplo.xml` em um documento XHTML que pode ser visualizado em um navegador:

Código 11 Transformando dados XML em XHTML.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xsl:stylesheet version="1.0"
3      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5  <xsl:template match="/">
6  <html>
7    <body>
8     <h1>Promoção</h1>
9     <table border="1">
10    <tr bgcolor="#9acd32">
11      <th align="left">Titulo</th>
12      <th align="left">Autor</th>
13    </tr>
14    <xsl:for-each select="biblioteca/livro">
15    <tr>
16      <td><xsl:value-of select="titulo"/></td>
17      <td><xsl:value-of select="autor"/></td>
18    </tr>
19    </xsl:for-each>
20    </table>
21   </body>
22 </html>
23 </xsl:template>
24
25 </xsl:stylesheet>
```

O documento produzido pelo Código 11 é chamado de **árvore resultado**, ou seja, ele é produzido a partir de uma árvore fonte (`exemplo.xml`). Um documento XSLT é um docu-

mento XML que possui o elemento raiz `stylesheet`. O Espaço de Nomes para um documento XSLT é `http://www.w3.org/1999/XSL/Transform`.

A linha `<xsl:template match = "/">` mostra um elemento `template`. Este elemento compara (*matches*) nós específicos de um documento XML usando espressões *XPath* no atributo da combinação. Neste caso, se faz a comparação com o nó raiz, simbolizado por "/".

Quando o elemento raiz combina com o elemento na árvore fonte, o conteúdo do elemento `template` é colocado na árvore de resultado. Pelo uso do elemento `value-of` e da expressão *XPath* no atributo `select`, o conteúdo do texto do conjunto de nós retornado pela expressão *XPath* são colocados na árvore de resultado. Mais detalhes sobre XSLT estão em Kay [12]. Na Subseção 6.2.3, é discutido sobre XSL-FO que geralmente é usada quando o resultado da transformação é para uma mídia impressa.

6.2.3 XSL *Formatting Objects* - XSL-FO

Na Seção 6.2.2, foi discutido sobre a XSLT, que transforma documentos XML em XHTML. A XSL-FO é tipicamente usada quando o resultado da transformação é para uma mídia impressa, como livros e revistas. Um documento XML é transformado em um documento XSL que marca os dados usando objetos de formatação. Este documento XSL pode ser então transformado em outros formatos, por exemplo, PDF ou TEX.

A XSL-FO é uma aplicação XML que descreve como páginas serão exibidas aos leitores, ou seja, como os usuários vão visualizar as informações. Uma folha de estilo usa a linguagem XSLT para transformar um documento XML, com um vocabulário semântico, em um novo documento XML que usa o vocabulário de apresentação XSL-FO.

Documentos XSL-FO são arquivos XML com informações de saída. Eles contêm informações sobre a disposição e o contéudo da saída. Os documentos XSL-FO são armazenados em arquivos com extensão `.fo` ou `.fob`. Todavia, é normal eles serem armazenados com a extensão `.xml`.

O vocabulário de objetos de formatação da XSL representa o conjunto de abstrações tipográficas disponíveis para o projetista. Existem 56 elementos de objetos de formatação XSL. O Espaço de Nomes padrão para estes objetos é `http://www.w3.org/1999/XSL/Format`. O prefixo padrão para este Espaço de Nomes é `fo` - *formatting objects* (objetos de formatação).

Formatação inclui semântica de formatação sobre árvore de resultado. Semântica de formatação é expressa em termos de um catálogo de classes de objetos de formatação. Os nós da árvore de resultado são os objetos de formatação.

As classes de objetos de formatação denotam abstrações tipográficas tais como página, parágrafo e assim por diante. Um bom controle sobre a apresentação destas abstrações é fornecida por um conjunto de propriedades de formatação, tais como alinhamento e espaço entre palavras e letras, por exemplo. Na XSL, as classes de objetos de formatação e propriedades de formatação fornecem o vocabulário para expressar a apresentação desejada.

Formatação consiste na geração de uma árvore de áreas geométricas, chamadas de **árvore área**. As áreas geométricas são posicionadas em uma seqüência de uma ou mais páginas. Cada área geométrica possui uma posição na página, uma especificação do que será visualizado naquela área e um plano de fundo e borda opcional.

O modelo de formatação XSL define as áreas para exibir a saída. Todas as saídas (texto, figuras ou qualquer outra coisa) são formatadas nestas áreas e são exibidas ou impressas em uma mídia alvo. Segundo [11], as áreas geométricas definidas na árvore área são:

- Páginas: a saída da XSL-FO é formatada em páginas, compostas de regiões. A impressão, normalmente, ocorre em muitas páginas separadas. A saída em um navegador freqüentemente é em uma página longa.
- Regiões: cada página XSL-FO contém várias regiões, como mostra a Figura 2:
  1. `region-body`: o corpo da página.
  2. `region-before`: o cabeçalho da página.
  3. `region-after`: o rodapé da página.
  4. `region-start`: lado esquedo da página.
  5. `region-end`: o lado direito da página.

  As regiões XSL-FO contém áreas de blocos.
- Áreas de blocos: definem pequenos blocos de elementos (normalmente iniciam com uma nova linha) como parágrafos, tabelas e listas. Áreas de blocos podem conter outras áreas de blocos, mas elas contém, geralmente, áreas de linhas.
- Áreas de linhas: definem linhas de texto dentro das áreas de blocos. Áreas de linhas contém áreas *inline*.
- Áreas *inline*: definem o texto no interior das linhas (caracteres simples, gráficos etc.).

  A estrutura de um documento XSL-FO é apresentada no Código 12.

Código 12 Estrutura de um documento XSL-FO.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
3    <fo:layout-master-set>
4      <fo:simple-page-master master-name="A4" page-height="4in">
5        ...
6        <!-- Todas as páginas mestre vão aqui -->
7        ...
8      </fo:simple-page-master>
9    </fo:layout-master-set>
10   <fo:page-sequence master-reference="A4">
11       ...
12       <!-- O conteúdo da página vai aqui -->
13       ...
14   </fo:page-sequence>
15 </fo:root>
```

Como os documentos XSL-FO também são documentos XML, eles sempre devem iniciar com a declaração mostrada na linha 1, do Código 12. O elemento `<fo:root>`, na linha 2, declara um documento XSL-FO. Na linha 2 também é declarado o Espaço de Nomes para XSL-FO.

Na editoração [10], a página mestre (`page masters`) define o *layout* da página, ou seja, as margens, cabeçalho, rodapé etc.. A página mestre permite ao autor do documento a flexibilidade para realizar mudanças no formato do documento página por página. O elemento

[10] Segundo Ferreira [9], editoração significa: reunir, organizar e anotar para publicação.

`<fo:layout-master-set>` contém uma ou mais páginas mestre, também chamadas de páginas *templates*. Para criar uma página mestre o elemento `fo:simple-page-master` é usado:

```
<fo:simple-page-master master-name="A4"page-height="4in">
      ⋮
   <!-- Formatação da página mestre vai aqui -->
      ⋮
</fo:simple-page-master>
```

Cada elemento `<fo:simple-page-master>` contém uma página mestre simples. Os atributos `master-name` e `page-height` especificam o nome da página e a sua altura, respectivamente. Os objetos de formatação XSL também fornecem um atributo para largura da página. Um documento pode conter qualquer número de páginas mestre.

Em uma página mestre `simple-page-master`, o documento é dividido em cinco regiões ou áreas, como mostra a Figura 2. O cabeçalho (*header*), corpo (*body*), rodapé (*footer*), lado esquerdo (*start*) e o lado direito (*end*) são representados pelos elementos de formatação XSL: `fo:region-before`, `fo:region-body`, `fo:region-after`, `fo:region-start` e `fo:region-end`, respectivamente.

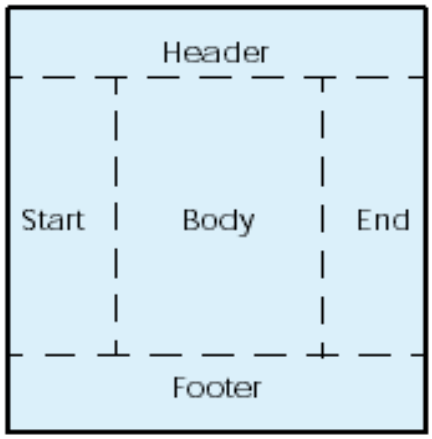

Figura 2: Regiões de um documento `simple-page-master` [11].

Um ou mais elementos `<fo:page-sequence>` descrevem o conteúdo da página. O atributo `master-reference` refere-se à página mestre `simple-page-master` com o mesmo nome:

```
<fo:page-sequence master-reference="A4">
      ⋮
  <!-- O conteúdo da página vai aqui -->
      ⋮
</fo:page-sequence>
```

Mais detalhes sobre XSL-FO podem ser vistos em Berglund [1]. Na Seção 7, discute-se o uso dos Espaços de Nomes, uma característica de XML que permite prevenir colisões de marcadores.

7 Espaços de Nomes (*Namespaces*) XML

Uma vez que XML permite que os autores de documentos criem suas próprios marcadores, colisões de nomes podem ocorrer. Uma colisão de nomes ocorre quando dois elementos diferentes possuem o mesmo nome. Por exemplo, um autor cria o elemento `data` para marcar a data de criação de uma página *Web*. Um outro autor pode também usar o elemento `data` para marcar a data da última atualização de uma página *Web* ou ainda a data de nascimento do autor. Se ambos elementos fossem usados em um mesmo documento haveria uma colisão de nomes e seria difícil determinar o tipo de dado que cada elemento contém.

Para prevenir colisões de nomes, é necessário identificar unicamente os elementos. Isto é feito usando-se Espaços de Nomes XML. Um Espaço de Nomes é uma forma para identificar uma parte da *Web* (espaço) que atua como um qualificador para um conjunto específico de nomes [13]. Por exemplo, considere-se o elemento `operacao` para marcar um trecho de uma informação:

```
<operacao>joelho< /operacao>
```

e

```
<operacao>Compra de dólares< /operacao>
```

No primeiro caso, `operacao` refere-se a medicina, ou seja, refere-se ao ato de realizar uma cirurgia no joelho, por exemplo. Já o segundo caso, `operacao` refere-se a movimentações financeiras, ou seja, à compra de dólares. Entretanto, os nomes dos elementos são iguais, logo, não é possível diferenciar o conteúdo dos elementos. Para poder diferenciar estes dois elementos é necessário o uso de Espaços de Nomes. Por exemplo:

```
<medicina:operacao>joelho< /medicina:operacao>
```

e

```
<financeira:operacao>Compra de dólares< /financeira:operacao>
```

Ambos `medicina` e `financeira` são prefixos de Espaços de Nomes. Prefixos de Espaços de Nomes são usados anexando-os aos nomes de elementos e atributos para especificar o Espaço de Nomes em que o elemento ou atributo pode ser encontrado. Cada prefixo está vinculado a um Identificador Único de Recursos (*Uniform Resource Identifier*) - URI[11], que identifica unicamente um Espaço de Nomes.

Autores de documentos podem criar seus próprios prefixos de *namespaces*. Qualquer nome pode ser usado para um Espaço de Nomes, exceto `xml` e `xmlns` [5]. O prefixo `xml` é, por definição, ligado ao Espaço de Nomes `http://www.w3.org/XML/1998/namespace`. Já o prefixo `xmlns` é usado para declarar o Espaço de Nomes `http://www.w3.org/2000/xmlns/`.

Existe uma convenção sobre a criação de Espaço de Nomes que diz que: para se criar páginas *Web* para descrever uma linguagem de marcação (e o significado dos marcadores) deve-se usar o URL da página *Web* como um URI para o Espaço de Nomes. Por exemplo, criar um Espaço de Nomes para Esquema XML:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
```

[11] Por definição, um URI é uma série de caracteres usados para diferenciar nomes.

Este pedaço de código demonstra como criar um Espaço de Nomes para ser usado em um documento XML (*Namespace* XML). Para este exemplo usa-se a palavra-chave `xmlns` seguida por `":"` e pelo nome `xsd` (`xmlns:nome`) para criar o prefixo do Espaço de Nomes `xsd` [5]. O valor vinculado ao atributo `xmlns:xsd` é um URL, ou seja, um tipo de URI. Com isso, o prefixo `xsd` é dito ser um nome local que representa o URI `http://www.w3.org/2000/10/XMLSchema`. Este URL possui os nomes dos marcadores que podem ser usadas e o significado de cada uma. O documento que define este Espaço de Nomes é encontrado em [23].

Para eliminar a necessidade de colocar um prefixo de Espaço de Nomes em cada elemento, autores podem especificar um Espaço de Nomes padrão para um elemento e todos seus elementos filhos. O Código 13 mostra um documento XML que faz uso de Espaço de Nomes.

Código 13 Um documento XML que faz uso de Espaço de Nomes padrão.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <diretorio xmlns="http://www.exemplo.org/texto/"
4             xmlns:imagem="http://www.exemplo.org/imagem/">
5
6    <arquivo nomearquivo="agenda.xml">
7      <descricao>Uma lista de nomes</descricao>
8    </arquivo>
9    <imagem:arquivo nomearquivo="flamengo.jpg">
10     <imagem:descricao>A foto do Flamengo campeao</imagem:descricao>
11     <imagem:tamanho altura="200" largura="100"/>
12   </imagem:arquivo>
13
14 </diretorio>
```

No Código 13, linha 3, foi declarado o Espaço de Nomes padrão, usando o atributo `xmlns` com um URI como seu valor. Uma vez que o Espaço de Nomes padrão foi declarado, os elementos filhos que são parte deste Espaço de Nomes não necessitam do prefixo do Espaço de Nomes. O Espaço de Nomes padrão é aplicado a todos os elementos contidos no elemento `diretorio`. O elemento `arquivo` está no Espaço de Nomes correspondente ao URI "`http://www.exemplo.org/texto/`", ou seja, o seu conteúdo é formado por documentos texto. Já o conteúdo do elemento `imagem:arquivo` é constituído de figuras, uma vez que foi definido o Espaço de Nomes `imagem`, definido na linha 4, para descrever figuras.

Uma outra vantagem do uso de Espaço de Nomes se refere à reutilização de marcadores. Por exemplo, caso exista um Espaço de Nomes que define marcadores sobre medicina e um autor necessite criar um documento XML com informações sobre medicina, ao invés de criar seus próprios marcadores, ou seja, "reinventar a roda", ele pode utilizar os marcadores já definidos e usados por vários outros autores. Com isso, ele ganha em tempo de criação, não precisa testar os marcadores, já que eles já foram testados por vários outros autores, e seu documento pode ser entendido por mais usuários.

Na Seção 8, são mostradas algumas linguagens baseadas em XML que estão sendo estudadas e recomendadas pelo W3C.

8 Tecnologias Adicionais

Além das tecnologias discutidas nas seções anteriores, existem outras linguagens baseadas em XML que estão sendo estudadas e recomendadas pelo W3C. Dentre estas tecnologias pode-se destacar: *Xlinks*, *XPointers*, *XQuery* e RDF. Nas próximas Subseções estas tecnologias são apresentadas de maneira sucinta.

8.1 XML *Linking Language - XLink*

XML *Linking Language* (*XLink*) permite que elementos sejam inseridos dentro de documentos XML para criar e descrever ligações entre recursos. Ela usa a sintaxe XML para criar as estruturas que podem descrever ligações unidirecionais simples, como existe em HTML, ou ligações mais sofisticadas.

XLink é capaz de ligar mais do que documentos. Ela liga recursos que podem incluir documentos, áudio, vídeo etc.. Por exemplo, um trecho de um filme poderia estar ligado ao áudio correspondente a ele.

Com a *XLink*, recursos podem ser ligados de várias formas diferentes. Um ligação simples (*simple link*) liga um recurso a outro, como ocorre em HTML. Os elementos *XLink* que especificam informações de ligações são chamados de **elementos de ligações** (*linking elements*). Por exemplo, pode-se utilizar uma ligação simples entre o `documento1` e `documento2`. O elemento de ligação `preco` no `documento1`, por exemplo, deve ser declarado como:

```
<preco xlink:type="simple"
   xlink:href="http://www.exemplo.org/listapreco.xml">
```

O `documento2` está sendo referenciado pelo URI `http://www.exemplo.org/listapreco.xml`. Neste cenário, o elemento de ligação, `preco`, que referencia o `documento2` é chamado de **recurso local**. O recurso referênciado é chamado de **recurso remoto**. Maiores detalhes sobre ligações mais complexas usando *XLink* estão em DeRose, Maler e Orchard [8].

8.2 XML *Pointer Language - XPointer*

A linguagem *XPointer* é usada como um fragmento identificador para alguma referência URI que localiza recursos de mídias da Internet cujo tipo é `text/xml`, `application/xml`, `text/xml-external-parsed-entity` ou `application/xml-external-parsed-entity` [10].

XPointer é dividida em um arcabouço (para especificar a localização de esquemas) e três esquemas: `element()`, `xmlns()` e `xpointer()`. O arcabouço e os dois primeiros esquemas, `element()` e `xmlns()`, formam a recomendação (pelo W3C) da *XPointer* [10].

O esquema `xpointer()`, que é baseado na *XPath*, está ainda em fase de desenvolvimento [10]. Ele Permite endereçamento dentro de estruturas internas de documentos XML. Além disso, permite percorrer árvores subjacentes a um documento e a seleção de suas partes internas com base em várias propriedades como, por exemplo, tipo de elemento, valores de atributos e posição relativa.

Usando *XPointer* com *XLink*, pode-se ligar partes específicas de um recurso, ao invés de ligar todo o recurso. Ela pode fazer ligações para localizações específicas. Além disso, ela também permite fazer busca em documentos XML usando o emparelhamento cadeias de caracteres. Maiores detalhes sobre *XPointer* estão em Grosso et al. [10].

8.3 XML *Query Language - XQuery*

Com o crescimento de informações armazenadas, intercambiadas e apresentadas usando XML, torna-se necessário algum mecanismo eficiente para fazer buscas sobre base de dados XML. *XQuery* foi projetada justamente com este objetivo.

Ela foi projetada para ser uma linguagem onde as consultas podem ser feitas de forma concisa e de fácil entendimento. Ela também é bastante flexível para realizar consultas em várias fontes de informações XML, incluindo banco de dados e documentos.

O *Query Working Group*, que faz parte do W3C, identificou dois requisitos para a sintaxe da *XQuery*: 1) sintaxe de consulta não baseada na XML e 2) sintaxe de consulta baseada na XML. *XQuery* foi projetada para satisfazer o primeiro destes requisitos. A sua sintaxe se assemelha às especificações de caminhos no ambiente UNIX (`root/directory/subdirectory`) [7]. Ela é derivada de uma linguagem de consulta XML chamada de Quilt, que adotou características de várias outras linguagens, incluindo *XPath* e SQL. Maiores detalhes sobre *XQuery* estão em Boag et al. [4].

8.4 *Resource Description Framework* - RDF

RDF constitui-se em uma arquitetura genérica de metadados que permite representar informações sobre recursos na *World Wide Web* (WWW ou *Web*), tais como título, autor e data de atualização de uma página *Web*, por exemplo. Além disso, RDF também pode ser usado para representar informações sobre coisas que podem ser identificadas na *Web*, mesmo que elas não possam ser recuperadas, como informações sobre itens acessíveis de um mercado *on-line* (por exemplo: preço e marca de um produto).

RDF foi projetado para situações onde as informações necessitam ser processadas por aplicações, em lugar de somente serem visualizadas por pessoas. Ele oferece uma estrutura comum para expressar informações que podem ser trocadas entre diferentes aplicações sem perda de significado [13]. Portanto, é uma tecnologia fundamental para a Web Semântica.

RDF se baseia no princípio de identificação de objetos usando identificadores *Web*, também chamados de URIs, e na descrição de recursos em termos de propriedades e valor da propriedade. Com isso, ela possui a capacidade de representar declarações simples sobre recursos como um grafo de nós e arcos representando os recursos e suas propriedades e valores [13].

A sua sintaxe é baseada em XML (chamada de RDF/XML). Esta sintaxe é processável por máquina e, usando URIs, pode-se ligar pedaços de informações através da *Web*. URIs em RDF podem se referir a qualquer coisa identificável, incluindo objetos que não podem ser recuperados diretamente na *Web*, como por exemplo uma pessoa.

Maiores detalhes sobre RDF estão em Manola e Miller [13]. Na próxima Seção, é discutido sobre o emprego da linguagem Java com XML. Além disso, são discutidas as APIs DOM e SAX.

9 Java e XML

A linguagem Java foi desenvolvida no início da década de 90 pela *Sun MicroSystems*. O objetivo inicial de Java foi a criação de uma linguagem independente de plataforma para ser usada em dispositivos eletrônicos.

Java é uma linguagem orientada a objetos independente de plataforma, possui um forte apelo em relação à Internet, é robusta, segura e com múltiplas linhas de execução. É uma linguagem independente de plataforma porque o compilador Java não gera um código executável

específico para um sistema, mas sim um código intermediário (*bytecode*), que é interpretado pela Máquina Virtual Java. Um programa Java compilado pode ser executado em qualquer ambiente onde existe uma máquina virtual Java que traduz as instruções em *bytecode* para este ambiente específico.

Java oferece vários recursos para se trabalhar com dados no formato XML. Além disso, a maioria das ferramentas para tratar XML são implementadas em Java. Isto é devido ao fato de Java e XML serem complementares, isto é, o código em Java e os dados em XML são ambos independentes de plataforma, são altamente portáveis, são padrões da indústria, extensíveis, reutilizáveis e centrados na *Web*. Por estas características, os dois fazem um par ideal para computação *Web*.

Com o objetivo de reunir as tecnologias usadas no desenvolvimento de aplicações que utilizam Java e XML, a *Sun MicroSystems* desenvolveu a `API JAXP` (*Java API for XML Processing*), discutida na próxima Subseção.

9.1 API JAXP

A API JAXP [14] oferece suporte básico para tratamento de arquivos XML na linguagem Java e provê um serviço de interface para analisadores sintáticos. Os principais componentes da API JAXP são definidos no pacote `javax.xml.parsers`. Esse pacote contém duas fábricas de classes independentes de fornecedor: `SAXParserFactory` e `DocumentBuilderFactory` que contém, respectivamente, o `SAXParser` e o `DocumentBuilder` [15]. A API também oferece a possibilidade de se fazer uso de implementações XML de outros fornecedores.

A API JAXP é utilizada para leitura, criação, manipulação e transformação de dados XML. Um documento XML pode ser criado, lido ou manipulado através das APIs DOM ou SAX, as quais são apresentadas nas duas próximas Subseções.

9.1.1 API SAX

A SAX (*Simple API for XML*) é dirigida a evento, ou seja, é necessário um mecanismo de acesso seqüencial que faz processamento de elemento a elemento. A biblioteca que define esta API é a `org.xml.sax`. Esta biblioteca define os componentes básicos da SAX.

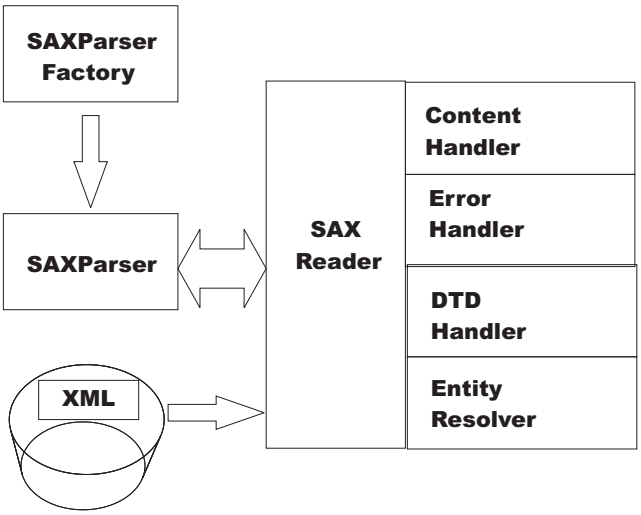

Figura 3: O funcionamento da API SAX [15].

O esquema de manipulação de documentos XML usando a API SAX de Java pode ser visto na Figura 3. O processo é iniciado com a criação da instância do objeto `SAXParser` por meio da classe `SAXParserFactory`. Para fazer a leitura é chamado o método `parser()`. O analisador sintático empacota o objeto `SAXReader`. Quando o analisador sintático chama o método `parse()`, o leitor, então, invoca um dos métodos de resposta implementados. Estes métodos são definidos pelas *interfaces*: `ContentHandler`, `ErrorHandler`, `DTDHandler`, `DocumentHandler` e `EntityResolver`. O analisador SAX está definido nos seguintes pacotes: `org.xml.sax`, `org.xml.sax.ext`, `org.xml.sax.helpers` e `javax.xml.parsers`.

Código 14 Código Java utilizando a API SAX.

```
1  import java.*;
2  import org.xml.sax.*;
3  import org.xml.sax.helpers.DefaultHandler;
4  import javax.xml.parsers.SAXParserFactory;
5  import javax.xml.parsers.ParserConfigurationException;
6  import javax.xml.parsers.SAXParser;
7
8  public class AgenteMonitor extends DefaultHandler{
9   ...
10  SAXParserFactory factory = SAXParserFactory.newInstance();
11  factory.setValidating(true);
12  SAXParser saxParser = factory.newSAXParser();
13  saxParser.parse( "Agenda1.xml", new ImplementaSAX());
14  ...
15 }
16 // Metodos do SAX DocumentHandler
17 class ImplementaSAX extends DefaultHandler {
18  public void startDocument() throws SAXException {
19   ... }
20
21  public void endDocument() throws SAXException {
22   ... }
23
24  public void startElement(String namespaceURI, String localName,
25              String qName, Attributes attrs) throws SAXException {
26   // "qName" contém o nome de um elemento
27   // Estou procurando o elemento chamado "Itens"
28   if (qName.equals("Itens"))
29   {
30     if (attrs != null)
31     {
32       // "attrs.getLength()" contém a quantidade de atributos que
33       // determinado elemento possui
34       for (int i = 0; i < attrs.getLength(); i++) {
35         // "attrs.getQName(i)" possui o nome de um atributo
36         // "attrs.getValue(i)" possui o valor do atributo
37         if ( (attrs.getQName(i).equals("Dia"))
38                       && (attrs.getValue(i).equals(21)) )
39           {   // O valor do atributo Dia é impresso
40               System.out.println(attrs.getValue(i);
41           } } } }
42  }
43
44  public void endElement(String namespaceURI, String localName,
45                         String qName) throws SAXException {
46    ... }
47
48  public void characters(char buf[], int offset,
49                         int len) throws SAXException {
50   String s = new String(buf, offset, len);
51   System.out.println(s);
52  } }
53 }
```

O Código 14 mostra a manipulação de um documento XML utilizando um analisador sintático baseado na SAX, implementado em Java [12]. Na linha 10, é criado o objeto `factory`, instância da classe `SAXParserFactory` (pacote `javax.xml.parsers`). Na linha 11, o objeto `factory` é configurado para validação de um analisador sintático; ele é `true` se o analisador sintático é válido ou `false` se o analisador sintático é não válido. Na linha 12, é criado o objeto `parser` baseado na SAX, chamado de `saxParser`. Na linha 13, o `parser` é chamado passando-se como argumento um documento XML e uma instância da classe `ImplementaSAX`. O analisador sintático lê o documento XML e invoca os métodos apropriados.

Na linha 17, é implementada a classe `ImplementaSAX`, que estende a classe `DefaultHandler`. A classe `DefaultHandler` implementa quatro *interfaces*: 1) `EntityResolver`: para manipular entidades externas; 2) `DTDHandler`: para manipular notações e entidades não analisadas; 3) `DocumentHandler`: para manipulação de eventos analisados; e 4) `ErrorHandler`: para manipulação de erros. A *interface* implementada no Código 14 é a interface `DocumentHandler`.

Da linha 18 até a linha 52, tem-se os métodos invocados pelo analisador sintático SAX:

- `startDocument` (linha 18): Este método é invocado quando o analisador sintático encontra o início de um documento XML.
- `endDocument` (linha 21): Este método é invocado quando o analisador sintático encontra o fim de um documento XML.
- `startElement` (linha 24): Este método é invocado quando o marcador de início de um elemento é encontrada.
- `endElement` (linha 44): Este método é invocado quando o marcador de fim de um elemento é encontrada.
- `characters` (linha 48): Este método é invocado quando caracteres de texto são encontrados.

O Código 14 mostra a navegação em um documento XML definido no DTD do Código 4. Quando um elemento é encontrado, o método `startElement` é invocado, neste momento, na linha 28, é comparado o valor do atributo `qName` com `Itens`, ou seja, o elemento procurado possui o nome `Itens`. Caso a comparação seja falsa, nada é feito. Caso contrário, verifica-se se o elemento possui atributos, linha 30. Nas linhas 37 e 38, é verificado se o elemento `Itens` possui o atributo `Dia` com o valor `21`. Se esta comparação for verdadeira é impressa, na linha 40, uma mensagem com o valor do atributo procurado.

SAX requer pouca utilização de memória, uma vez que ela não constrói uma representação interna dos dados na memória. Por isso, SAX é utilizada em leitura de dados que requer rapidez e fluxos de dados consecutivos, não permitindo navegar arbitrariamente em um conjunto de informações.

O Código 14 é bastante simples, contudo, o seu objetivo é mostrar algumas funcionalidades que um analisador sintático SAX oferece na manipulação de documentos XML. Mais detalhes estão em [24].

[12] Este código possui apenas os trechos que exemplificam o uso da SAX.

9.1.2 API DOM

O DOM (*Document Object Model*) está definido pelo W3C como uma plataforma e uma interface neutra de linguagem que permite que programas possam acessar e alterar dinamicamente o conteúdo, estrutura e estilo de um documento [16]. Ele oferece um conjunto de objetos padrão para representar documentos XML, um modelo padrão de como estes objetos podem ser combinados e uma *interface* padrão para acessar e manipular estes objetos [16].

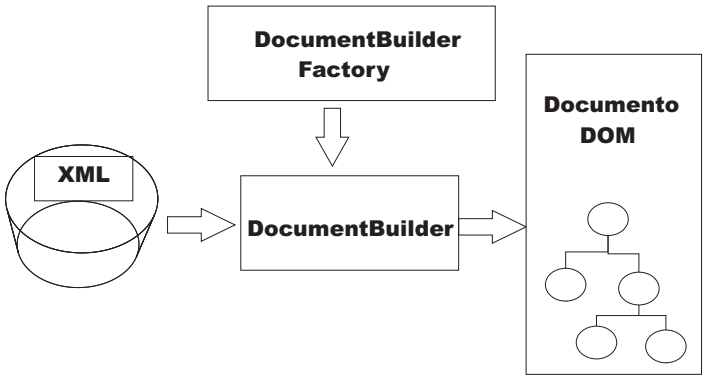

Figura 4: O funcionamento da API DOM [15].

O DOM é uma API para manipulação de documentos XML bastante utilizada nos dias atuais. Ele apresenta um documento XML em uma forma baseada em objetos, tornando-o simples para ser manipulado por programadores Java. Por isso, pode-se dizer que DOM é uma apresentação orientada a objetos para documentos XML. Ele fornece uma estrutura em árvore dos objetos e pode ser utilizado para manipular estruturas de objetos hierárquicos bem como seus encapsulamentos [15].

Um analisador sintático baseado em DOM permite que os dados em um documento XML possam ser criados, acessados e modificados pela manipulação dos nós em uma árvore DOM. Na Figura 4 é mostrado o esquema para manipulação de documentos XML usando a API DOM de Java. É por meio da classe `javax.xml.parsers.DocumentBuilderFactory` que se consegue uma instância para `DocumentBuilder` que é responsável pela criação do documento em forma de hierarquia. A implementação do DOM é definida nos seguintes pacotes: `org.w3c.dom` e `org.xml.sax.ext`.

Código 15 Código Java utilizando a API DOM.

```
1  import java.util.Date;
2  import java.io.*;
3  import java.util.*;
4  import javax.xml.parsers.*;
5  import org.xml.sax.*;
6  import org.w3c.dom.*;
7
8  public class AgenteMonitor
9  {
10   public AgenteMonitor(String horario[])  {
11     ...
12     // Criar árvore na memória com a estrutura do arquivo XML (DOM)
13     try {
14        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
15        DocumentBuilder db = dbf.newDocumentBuilder();
16        // Associa o arquivo XML árvore na memória
17        Document doc = db.parse( "Agenda.xml" );
18
19        // Acesso aos elementos da árvore através da classe Element
20        Element dE = doc.getDocumentElement();
21        // Busca os elementos cujo nome é "Itens"
22        NodeList clTb = dE.getElementsByTagName("Itens");
23
24        // Navegando na lista dos elementos itens contidos no arquivo.
25        // "clTb.getLength()" contém quantos elementos com nome "Itens"
26        // existe no arquivo .xml percorrido
27        for( int i = 0; i < clTb.getLength(); i++) {
28          Element aE = (Element) clTb.itens( i );
29          // É realizada uma conversão
30          String clDia  = aE.getAttribute("Dia");
31          // Retorna o valor do atributo 'Dia'
32          // O valor do atributo Dia é impresso
33          System.out.println(clDia);
34        }
35     }
36     catch (Exception e) {
37        System.out.println(e.getMessage());
38        System.out.println("ERRO");
39        System.exit(1);
40     }
41   }
42 }
```

O Código 15 mostra a manipulação de um documento XML, definido no DTD do Código 4, utilizando-se um analisador sintático baseado no DOM, implementado em Java[13]. Este código possui as mesmas funcionalidades do Código 14. Pode-se perceber que o código é bem mais "limpo" e claro do que o código implementado em um analisador sintático baseado em SAX.

[13]Este código possui apenas os trechos que exemplificam o uso do DOM.

JAXP usa a classe `DocumentBuilderFactory` para criar um objeto `DocumentBuilder`. A classe `DocumentBuilder` fornece uma interface padrão para um analisador sintático XML. Na linha 14, é criado e nomeado um objeto `DocumentBuilderFactory` chamado de `dbf`.

Na linha 15, cria-se um novo objeto `DocumentBuilder` e o nomeia como `db`. Este objeto fornece uma *interface* para carregar e analisar documentos XML. Na linha 17, o método `parse` é invocado para carregar e analisar o documento XML armazenado no arquivo `Agenda.xml`. Se a análise sintática for realizada com sucesso, um objeto `Document doc` é retornado. Este objeto contém os nós representando cada parte do documento `Agenda.xml`. Se a análise sintática falhar, uma exceção é gerada, `SAXException`.

Na linha 20, o método `getDocumentElement` é invocado para obter o nó raiz do documento XML. Na linha 22, é utilizado o método `getElementsByTagName()` para obter todos os elementos do documento que possuem o nome `Itens`. Na linha 28, é feita uma conversão de um tipo `Node` para um tipo `Element`. A classe `Element` herda da classe `Node`.

Na linha 30, é retornado o valor do atributo `Dia`. Na linha 33, o valor do atributo `Dia` é impresso. Comparando o Código 14 com este Código 15, pode-se perceber que a API DOM é mais fácil de usar e entender do que a API SAX.

O DOM é ideal para aplicações interativas porque todo o modelo de objetos fica na memória, onde pode ser acessado e manipulado pelo usuário. Por esta razão, a construção de um documento DOM requer mais utilização da CPU e da memória, o que não ocorre com SAX. O Código 15 mostra como se pode percorrer um documento XML utilizando DOM. Além desta funcionalidade, o DOM permite criar um documento e modidicar um existente. Mais detalhes estão em [22].

10 Considerações Finais

Com o crescimento da *Web* como o maior meio de armazenamento de informações, as limitações da linguagem HTML começaram a ser percebidas. A linguagem XML surgiu como uma combinação de poder e extensibilidade e com a simplicidade requerida pela comunidade *Web*. Isto é possível porque ela é uma linguagem de marcação de dados que define um formato para descrever dados estruturados, permitindo um número ilimitado de marcadores, cada um deles indicativo, não de como algo deve ser exibido, e sim do que significa. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas.

Além disso, existem várias outras vantagens no uso da linguagem XML, dentre estas vantagens pode-se destacar algumas:

- Múltiplas representações: usando-se um documento XSL (*Extended Style Language*) é possível criar múltiplas representações da mesma informação.
- Buscas mais eficientes: os dados em XML podem ser unicamente rotulados, o que permite, por exemplo, que uma busca por livros seja feita em função do nome do autor e do título de um livro.
- Computação e manipulação local: os dados XML recebidos por uma aplicação podem ser analisados, editados e manipulados de acordo com o interesse da aplicação. A separação da *interface* visual dos dados permite a criação de aplicações mais poderosas, simples e flexíveis.

- Padronização: um aspecto muito importante, principalmente, quando envolve a representação e intercâmbio de informações. XML oferece este aspecto, uma vez que ela é baseada em outras tecnologias padrão como o SGML para a sintaxe, URIs para os identificadores, EBNF para a gramática e o *Unicode* para a representação dos caracteres [15].
- Manutenibilidade: é independente de qualquer plataforma, com isso, ela pode ser utilizada para produzir diferentes resultados para diferentes dispositivos sem a necessidade de mudanças no conteúdo original. Isso leva a uma maior eficiência e maior facilidade de manutenção, uma vez que não há o trabalho extra para o controle de versão nem para a realização de modificações específicas para uma mídia.
- Facilidade na troca de dados: permite que a troca de dados entre diferentes sistemas seja feita de maneira simples e eficiente.
- Extensibilidade: uma vez que a XML é uma meta-linguagem, ela fornece meios para o desenvolvimento de linguagens de marcação para qualquer área, desde a medicina ao comércio eletrônico.

Além destas vantagens, ela também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via *Web*.

11 Agradecimento

Ao Prof. Dr. João Carlos da Silva pela avaliação do presente texto e pelas sugestões feitas, as quais muito contribuiram para a melhoria do texto original.

Referências

[1] BERGLUND, A. **Extensible Stylesheet Language (XSL) Version 1.1**. `http://www.w3.org/TR/2003/WD-xsl11-20031217/`, acessado em Julho de 2005, Dezembro 2003.

[2] BERGLUND, A; BOAG, S; CHAMBERLIN, D; FERNÁNDEZ, M. F; KAY, M; ROBIE, J; SIMÉON, J. **XML Path Language (XPath) 2.0**. `http://www.w3.org/TR/2004/WD-xpath20-20041029/`, acessado em Julho de 2005, Outubro 2004.

[3] BERNERS-LEE, T; HENDLER, J; LASSILA, O. **The Semantic Web**. Scientific American, May 2001.

[4] BOAG, S; CHAMBERLIN, D; FERNÁNDEZ, M. F; FLORESCU, D; ROBIE, J; SIMÉON, J. **XQuery 1.0: An XML Query Language**. `http://www.w3.org/TR/2004/WD-xquery-20041029/`, acessado em Julho de 2005, Outubro 2004.

[5] BRAY, T; HOLLANDER, D; LAYMAN, A; TOBIN, R. **Namespaces in XML 1.1**. `http://www.w3.org/TR/2004/REC-xml-names11-20040204/`, acessado em Julho de 2005, Fevereiro 2004.

[6] BRAY, T; PAOLI, J; SPERBERG-MCQUEEN, C. M; YERGEAU, F. **Extensible Markup Language (XML) 1.0**. `http://www.w3.org/TR/2004/REC-xml-20040204/`, acessado em Julho de 2005, Fevereiro 2004.

[7] DEITEL, H; DEITEL, M; PAUL, J. **XML - How to Program**. Prentice-Hall, Inc., 2000.

[8] DEROSE, S; MALER, E; ORCHARD, D. **XML Linking Language (XLink) Version 1.0**. `http://www.w3.org/TR/xlink/`, acessado em Julho de 2005, Junho 2001.

[9] FERREIRA, A. B. H. **Dicionário Aurélio Básico da Língua Portuguesa**. Nova Fronteira, 1th edition, 1988.

[10] GROSSO, P; MALER, E; MARSH, J; WALSH, N. **XPointer Framework**. `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`, acessado em Julho de 2005, Março 2003.

[11] HAROLD, E. R. **The XML Bible**. IDG Books, 2ª edition, 1999.

[12] KAY, M. **XSL Transformations (XSLT) Version 2.0**. `http://www.w3.org/TR/2004/WD-xslt20-20041105/`, acessado em Julho de 2005, Novembro 2004.

[13] MANOLA, F; MILLER, E. **RDF Primer**. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`, acessado em Julho de 2005, Fevereiro 2004.

[14] MICROSYSTEMS, S. **Java API for XML Processing (JAXP)**. `http://java.sun.com/xml/jaxp/index.jsp`, acessado em Julho de 2005, 2004.

[15] MORAES, L. **XML e Java: uma Perspectiva do Programador**. Monografia de Especialização. Instituto de Informática - UFG, 2003.

[16] RAMBHIA, A. M. **XML Distributed Systems Design**. SMS, USA, 2002.

[17] RECORDARE. **MusicXML Definition**. `http://www.recordare.com/xml.html`, acessado em Julho de 2005, 2005.

[18] VLIST, E. V. D. **XML** *Schema*. O'Reilly, USA, 1ª edition, 2002.

[19] W3C. **Cascading Style Sheets Home Page**. `http://www.w3.org/Style/CSS/`, acessado em Julho de 2005, 2004.

[20] W3C. **HyperText Markup Language (HTML) Home Page**. `http://www.w3.org/MarkUp/`, acessado em Julho de 2005, 2004.

[21] W3C. **W3C Math Home**. `http://www.w3.org/Math/`, acessado em Julho de 2005, 2004.

[22] W3C. **Document Object Model (DOM)**. `http://www.w3.org/DOM/`, acessado em Julho de 2005, 2005.

[23] W3C. **Espaço de Nomes para XMLSchema**. `http://www.w3.org/2000/10/XMLSchema`, acessado em Julho de 2005, 2005.

[24] W3C. **W3C Home Page**. `http://www.w3.org/`, acessado em Julho de 2005, 2005.