XML

XML, ou Extensible Markup Language, é um padrão da W3C (World Wide Web Consortium) para definição de documentos estruturados. Na verdade, XML não representa uma linguagem específica mas um conjunto de regras para a construção de documentos. Documentos que aderem às regras do XML têm uma estrutura parecida com páginas HTML mas não estão presos a um vocabulário limitado de elementos.

Várias aplicações do XML têm sido desenvolvidas. Aplicações são linguagens, criadas usando a sintaxe XML, mas com regras específicas que definem seus elementos, atributos e organização estrutural. Várias aplicações surgem a cada ano e muitas já se tornaram padrões.

Inicialmente criada visando a Web, XML tem ido além e encontrado aplicações nas mais variadas áreas do conhecimento. Onde há informação que pode ser organizada de maneira estruturada pode haver uma aplicação para XML. O resultado é que, por aderirem a regras gerais de organização estrutural, documentos XML são mais fáceis de compartilhar. XML está se consolidando como o formato padrão para armazenar a maior parte dos diferentes tipos de documentos usados nos computadores, desde textos, planilhas e mensagens a gráficos vetoriais, animações e correio de voz.

Soluções para ampliar o poder do XML estão sendo desenvolvidas usando o próprio XML. As principais tecnologias para transformação de documentos (XSL), ligações de hipermídia (Xlink) e validação (XML-Schema) são aplicações do XML. Isto permite que as mesmas aplicações possam analisar, validar e processar esses dados muito mais facilmente.

Para manipular com documentos XML foram padronizadas interfaces de programação (bibliotecas) implementadas em várias linguagens como C++, Java, VB, Delphi, Perl, Python e JavaScript. Em todas essas linguagens, os nomes dos objetos e das funções usadas para pesquisar, alterar e criar documentos XML são iguais, o que facilita ainda mais a utilização de documentos XML em aplicações de software.

Um documento XML descreve a estrutura dos dados, apenas. Não interessa como os dados serão apresentados. Sendo assim, XML reforça a separação entre os dados, seu significado (o que representam) e como devem ser expostos (sua apresentação) tornando mais fácil a sua utilização em meios diferentes.

Linguagens como RTF (Rich Text Format) são totalmente voltadas à formatação da página (descreve: isto tem 24pt, fonte Helvetica). HTML, embora descreva elementos por sua função (descreve: isto é um título de nível 1), tem sua função ligada à forma como os dados serão apresentados. XML, por sua vez, não oferece dica algma sobre formatação dos dados. O mais importante é representar os dados da forma mais descritiva possível (descreve:isto é um nome do capítulo). XML não diz como representar. XML diz o que é.

Mesmo que uma página XML não informe como os dados serão apresentados, ela ainda assim pode ter a mesma aparência que uma página HTML ao ser visualizada em um browser. Isto é possível com folhas de estilos – documentos que lidam exclusivamente com apresentação de dados. Regras de estilo são associadas aos elementos para gerar uma página para impressão, por exemplo. Ao manter-se independente da apresentação, XML garante maior facilidade na geração de visões diferentes dos mesmos dados. Folhas de estilo podem ser usadas para transformar um mesmo XML em HTML, WML (Wap), PDF ou PostScript automaticamente.

Um dos problemas do HTML é o número limitado de elementos, o que impede uma melhor identificação das informações. É quase impossível realizar uma busca na Internet para achar, por exemplo, hotéis na ilha de Java. Se os dados estivessem em XML seria possível um autor ou o indexador descrever os dados de maneira mais clara:

```
<linguagem>Java</linguagem>
<indonesia><ilha>Java</ilha></indonesia>
```

E evitar que o usuário recebesse uma lista imensa de links sobre programação ao tentar programar sua viagem de férias. Com HTML isto não é possível.

Como funciona XML?

Um documento XML é formatado por uma coleção de elementos XML, organizados de uma maneira hierárquica. Cada elemento pode conter texto ou outros elementos. Os elementos são representados por etiquetas como em HTML. A listagem a seguir mostra um exemplo de documento XML bem formatado. Os nomes dos elementos estão em negrito. Os atributos são os nomes que estão dentro dos elementos mas não estão em negrito. Atributos sempre têm o formato nome=”valor”.

```
<bilhete codigo=”XPTO80”>
  <voo transportador=”JH” numero=”3241” de=”GUA” para=”PEC” />
  <passageiro>
    <sobrenome>Silva</sobrenome>
    <prenome>José</prenome>
  </passageiro>
</bilhete>
```

A especificação XML não define um vocabulário de nomes para elementos ou atributos como no HTML. Apenas define regras sobre como os elementos devem ser representados. O autor está livre para escolher os nomes mais adequados para etiquetar seus dados.

XML é só isto. Uma especificação que estabelece regras para construir documentos com marcação semelhante a HTML, mas sem definir nenhum elemento. Daí vem o nome “Extensible”, pois diferentemente de linguagens como HTML, documentos XML, podem ser estendidos com novos elementos.

Documentos válidos e bem formatados

As regras de formatação estabelecidas pela especificação XML são essenciais para que programas que entendem XML possam processar os documentos. Documentos que seguem as regras são documentos XML *bem formatados*.

As regras do XML são insuficientes para atribuir valor ao conteúdo estruturado do documento. Uma aplicação XML requer um esquema que descreva todos os seus elementos, atributos, valores permitidos, e outras regras de construção. Para validar os dados de uma determinada aplicação do XML existem os *esquemas*, que definem todo o vocabulário permitido para elementos e atributos, além de determinar os tipos de dados que podem ser usados, a ordem e organização hierárquica do elementos. Documentos que aderem não só às regras de formação do XML mas também às regras de um esquema de validação são chamados de documentos válidos.

Quando se estabelece, através de um esquema, um vocabulário limitado e uma gramática definida para a construção de documentos XML, uma sub-linguagem, de aplicação específica, está sendo definida. Linguagens produzidas a partir de XML são chamadas de aplicações XML. A linguagem XHTML, por exemplo, é XML com vocabulário de elementos igual ao HTML 4.0. Aplicações que interpretam XHTML esperam encontrar elementos como **<p>**, **<h1>** em documentos XHTML, mas um outro

programa que só entende XML poderá abrir o documento XHTML, navegar por sua estrutura, identificar seus componentes, manipulá-los e transformar o documento. Em outras palavras, um documento XHTML é também XML. As suas regras de formatação são ditadas pela especificação XML. Os nomes dos seus elementos e atributos e outras regras adicionais são ditadas pela especificação do XHTML. XHTML é uma aplicação do XML. De forma similar, o exemplo mostrado anteriormente que representa um bilhete de passagem aérea é outra aplicação do XML, que tem provavelmente algumas regras adicionais. Talvez sejam algo como:

- Os elementos permitidos são **bilhete**, **voo**, **passageiro**, **sobrenome**, **prenome**.
- O elemento bilhete contém pelo menos um **voo** e exatamente um **passageiro**.
- O elemento **passageiro** deve ter um elementos **sobrenome** e um elemento **nome**.
- Os atributos **de** e **para** de **voo** contém valores que podem ser escolhidos de uma lista. A lista contém todos os códigos de aeroportos.

Como especificar uma aplicação XML

Atualmente existe mais de uma maneira de especificar uma linguagem. A mais simples, que é descrita na especificação XML, é o DTD, ou *Document Type Definition*. O DTD define a gramática que permite validar um documento XML de acordo com a aplicação a que ele se destina. No DTD estão definidos todos os elementos legais, atributos, regras de uso, etc. Documentos XML que combinam com o DTD são documentos válidos. A listagem abaixo representa um possível DTD para a linguagem que descreve bilhetes de passagens aéreas:

```
<!ELEMENT bilhete (voo, passageiro ) >
<!ATTLIST bilhete codigo NMTOKEN #REQUIRED >

<!ELEMENT passageiro ( sobrenome, prenome ) >
<!ELEMENT prenome ( #PCDATA ) >
<!ELEMENT sobrenome ( #PCDATA ) >

<!ELEMENT voo EMPTY >
<!ATTLIST voo de NMTOKEN #REQUIRED >
<!ATTLIST voo numero NMTOKEN #REQUIRED >
<!ATTLIST voo para NMTOKEN #REQUIRED >
<!ATTLIST voo transportador NMTOKEN #REQUIRED >
```

É importante observar que a sintaxe usada no DTD não é XML, embora seja semelhante (é sintaxe SGML). Em negrito estão as definições de elementos. <!ATTLIST> representa definições de atributos.

A validade de um documento depende do DTD. Um documento bem-formatado que é validado de acordo com um DTD pode não ser válido de acordo com outro DTD. Existe portanto uma ligação entre o documento e seu DTD. O DTD especifica a linguagem e representa uma classe de documentos. Cada documento XML que estiver de acordo com

as regras do DTD é validado por ele. Um documento SVG, por exemplo, pode ser validado com um DTD do SVG. Uma aplicação que precise gerar documentos SVG pode usar o DTD para garantir que os documentos gerados serão válidos e conseqüentemente compreendidos por outras aplicações que estendem SVG.

Mas nem todos os documentos precisam ser validados. DTDs são opcionais. Um documento bem formatado pode ser transformado por uma folha de estilos visando a formatação do documento para impressão. Mas se o documento contiver elementos que a folha de estilos não conheça, a formatação poderá não ser realizada como desejada.

DTDs não resolvem todos os problemas de validação. Um programa que lê ou gera documentos XML frequentemente faz alguma validação adicional dos dados. DTDs por exemplo não podem garantir que o conteúdo de um elemento seja um número entre 0 e 100.

Como produzir XML?

Documentos XML podem ser produzidos em qualquer editor de textos. Existem vários editores especializados em documentos XML. Como documentos maiores geralmente são gerados por software e os menores geralmente são mais facilmente editados em editores comuns, alguns editores especializados trazem mais trabalho que benefícios. Por *default*, XML representa os dados como UTF-8 (*Unicode Transformation Format*) mas vários outros formatos são suportados. Qualquer editor de textos que não tenha problemas com caracteres que não sejam ASCII é suficiente para escrever XML.

Não existe uma extensão de nome de arquivo padrão para documentos XML. A extensão *.xml* é usada para documentos genéricos mas é comum se usar uma extensão que descreva a aplicação do XML (como .svg, .xsl ou .fo). Um documento XML pode sequer ser um arquivo. Pode ser o fluxo de dados que está sendo recebido da rede por um browser. Pode ser um registro em um bando de dados. Para identificá-lo, existem vários diferentes tipos MIME que relacionam o conteúdo como simplesmente XML (application/xml e text/xml) ou alguma aplicação específica do XML (image/svg+xml).

Como validar?

Existem dois tipos de processadores XML (*parsers*): *validadores* e *não validadores*. Os primeiros comparam o documento XML com seu DTD e os outros ignoram o DTD. Se o *parser* detectar um erro no documento ele pára o processamento e acusa um erro. O erro pode ser um erro de formatação caso o documento não seja um documento XML ou não esteja bem formatado) ou um erro de validação (caso seja bem formatado mas não esteja de acordo com o DTD).

Parsers lêem o documento XML e montam, internamente, uma estruturada de árvore que pode ser manipulada através de linguagens de programação. Para simplesmente validar documentos XML há várias ferramentas de domínio público.

Para que serve XML?

Serve como formato para gravação ou transmissão para quaisquer tipos de dados que possam ser eficientemente representados como texto. Isto inclui websites, texto formatado, figuras vetoriais, planilhas, tabelas de dados, roteiros de animação, instruções de processamento, etc. XML não é um bom formato para representar dados como imagens, vídeos, som e outros formador multimídia que consistem de grandes quantidades de informação, geralmente comprimida, onde o espaço ocupado pelos dados é crítico.

Usar XML no lugar de um formato proprietário traz várias vantagens como:

separação da estrutura da apresentação, informação semântica, independência de plataforma, facilidade de geração de visões diferentes para os dados, facilidade de leitura por pessoas e maquinas, facilidade de compartilhamento de dados entre aplicações diferentes por se um padrão.

História do XML

XML é um subconjunto de uma especificação bem mais abrangente chamada de SGML – *Standard Generalized Markup Language*, desenvolvida no final da década de 70 e que se tornou padrão ISO8879 em 1986. SGML foi e é usado em várias aplicações onde estrutura de dados é importante. A aplicação mais popular do SGML é a linguagem HTML.

Assim como XML. SGML não define seu vocabulário de elementos, apenas regras de construção de documentos bem-formatados. Mas SGML oferece muita flexibilidade e muito mais regras que, embora pouco usadas na maior parte das aplicações, teriam que ser levadas em conta na definição de qualquer nova aplicação.

XML foi proposto como uma versão light de SGML, com o objetivo de ser aplicada na Web. A aplicação ficou muito mais simples. O documento que especifica XML tem 10% do tamanho do documento isso que especifica o SGML. E isto acabou atraindo não só quem tinha interesse em usar a linguagem na Web mas quem já usada SGML. Muitas aplicações populares do SGML, abertas e proprietárias estão sendo reescritas em XML.

Fundamentos

Agora será mostrado os diferentes tipos de estruturas que existem num documento XML e como construir um documento bem formatado:

```
<nome>Fulano de Tal</nome>
```

A unidade de informação dentro de um documento XML é o *elemento*. Um elemento é formado por duas *etiquetas* (ou *tags*, em inglês) que atribuem algum significado ao conteúdo. O significado é abstrato. O elemento só precisa ter significado para as aplicações que irão compartilhá-lo. Construir um documento XML para uma aplicação é atribuir nomes significativos (do ponto de vista da aplicação) a pedaços de informação, ou seja, etiquetá-los. Os nomes devem descrever o que representa a informação (o que ela é) e não como ela deve ser formatada para exibição.

Todo elemento XML que é usado para etiquetar algum bloco de informação possui um par de etiquetas que contém o elemento.

Elementos que têm conteúdo vazio podem ser representados com uma única etiqueta, terminando em **/>**. Os dois elementos abaixo são equivalentes:

```
<vazio/>
```

ou

```
<vazio></vazio>
```

Um elemento (não vazio) é composto por uma etiqueta inicial, seu conteúdo e uma etiqueta final. O conteúdo pode ser composto de texto, outros elementos XML ou de texto e elementos misturados (conteúdo minto). O documento abaixo contém um elemento raiz <contato> que contém outros elementos <nome>, <email> e <telefone> que por sua vez contém outros elementos <ddd> e <numero>.

```
<contato>
  <nome>Fulano de Tal</nome>
  <email>fulano@site.com.br</email>
  <telefone>
    <ddd>11</ddd>
    <numero>5234-0394</numero>
  </telefone>
</contato>
```

Documentos XML contém texto, e apenas texto, portanto, podem ser abertos em qualquer editor de textos. O ideal é usar algum editor que destaque os nomes dos elementos, atributos e texto em cores diferentes para facilitar a visualização mas não é necessário usar um editor especialmente criado para manipular XML.

Se os documentos XML forem armazenados como arquivos, devem usar uma extensão significativa, embora seja opcional. O default é usar .xml, mas, se você estiver usando alguma aplicação popular do XML (por exemplo, SVG) , usar a extensão padrão para essa aplicação (.svg). Usar uma extensão padrão ajuda os editores de texto que reconhecem XML a melhorar a apresentação do código com cores e endentações.

Partes de um documento

Um documento XML pode ser dividido em duas partes. A parte inicial, ou prolog, é tudo o que aparece antes do elemento raiz. Um documento XML vem formatado só possui um elemento raiz, portanto, os elementos do mesmo nível que aparacem antes dele fazem parte do prolog. Abaixo segue um documento XML que ilustra o prolog e a raiz (o corpo).

```
<?xml version="1.0" encoding="iso-8859-1" ?>      <--- Declaração XML
<!-- Isto é um comentário -->                     <--- Comentário
<?comando tipo="simples" parametro ?>             <--- Instrução de Processamento
<!DOCTYPE cartao-simples SYSTEM "cartoes.dtd">     <--- Declaração de tipo de documento
<cartao-simples>                                  <--- Elemento do documento
  <logotipo href="/imagens/logo14bis.gif" />      <--- Elemento com atributo
  <nome>Fulano de Tal</nome>
  <endereco>Rua Bela Vista, 45 - fundos
  Centro - 04938-201 - São Paulo - SP</endereco>
  <email>fulano@terra.com.br</email>
  <telefone tipo="residencial">
    <ddd>11</ddd>
    <numero>5234-0394</numero>
  </telefone><!DOCTYPE>
</cartao-simples>
```

O prolog pode consistir de uma instrução de processamento padrão, chamada de declaração XML, uma declaração de tipo de documento, que começa com <!DOCTYPE>, comentários e zero ou mais instruções de processamento (que, por default, são ignoradas como os comentários). Tudo é opcional. A declaração XML também é opcional mas, no documento mostrado tornou-se obrigatória porque define um tipo de codificação (isso-8859-1) de texto diferente do default (UTF-8). A declaração de <!DOCTYPE> é opcional

em documentos bem formatados mas obrigatório se o documento for validado por um DTD.

O corpo do documento começa no seu elemento raiz e também pode conter comentários em qualquer lugar (menos dentro das etiquetas). A sua estrutura representa uma árvore. Cada elemento, portanto, é considerado um nó.

Os elementos contidos dentro de elementos são os nós filhos (child nodes). O nó dentro do qual um nó filho está contido é seu nó pai (parent node). Todos os nós, com exceção do nó raiz tem um nó pai. Os nós que não tem filhos (nós de texto puro ou elementos vazios) são nós folha (leafs). Os elementos de mesmo nível são chamados de nós irmãos (siblings). Elementos podem conter atributos que também são chamados de nós, embora tenha um tratamento especial por parte dos processadores.

Documentos XML também podem conter entidades de substituição, que são símbolos do tipo &nome; onde nome é usado para identificar a entidade. Entidades representam texto ou fragmentos de XML que serão embutidos no documento quando ele for processado. Caso um documento não possua esquema (DTD), ele só poderá usar as entidades default do XML, que inclui caracteres especiais identificados através de seu código Unicode e cinco entidades para imprimir os caracteres <, >, “, ' e &. Por exemplo, Para escrever a expressão matemática (a < b) em XML é preciso usar (a &lt b).

Elementos e atributos

A etiqueta inicial de um elemento é identificada pelo nome do elemento e pode ter zero ou mais atributos. Não deve haver espaço entre o < e o nome do elemento na etiqueta inicial nem entre o </ e o nome na etiqueta final. Pode haver ou não espaço antes do > em qualquer etiqueta. No caso dos elementos vazios, pode haver espaço antes do />. A etiqueta final só contém o nome do elemento precedido de uma barra.

Atributos tem sempre a forma *nome=”valor”* e só podem aparecer na etiqueta inicial. O valor pode aparacer entre aspas ou apóstrofes.

Os nomes dos elementos nas etiquetas inicial e final têm que ser lexicograficamente iguais, ou seja, um nome expresso em minúsculas é diferente do mesmo nome expresso em maiúsculas ou com uma capitular. Os três elementos abaixo não são bem formatados por este motivo:

```
<Profissão>Analista</profissão>
<TR><TD>item um</td></tr>
<tamanho>123.45</TAMANHO>
```

Há várias maneiras de representar a mesma informação em XML.

```
<data>15/04/1945</data>
<data dia=”15” mês=”04” ano=”1945” />
<data>
  <dia>15</dia>
  <mes>04</mes>
  <ano>1945</ano>
</data>
```

Essa liberdade de organização dos dados levanta uma questão de design: quando representar uma informação como elemento e quando representá-la como atributo?

Em geral, a escolha deve ser determinada pela forma como os dados serão utilizados pelos programas que irão processá-los. Há vários motivos para preferir elementos sobre atributos e vice-versa. Alguns motivos tem motivação baseada em eficiência outros em legibilidade.

Atributos geralmente são usados para representar pequenas quantidades de dados. Um texto longo deve sempre ser representado como um elemento. Atributos não podem ser repetidos, portanto, são inadequados para representar informações que podem se repetir. Por exemplo,, no exemplo mostrado, um cartão de visita pode ter mais de um <telefone> ou <email> e isto é melhor representado como um elemento. Atributos não servem para dados hierárquicos. Para representar um telefone como um atributo a sua estrutura (<ddd> e <numero>) seria perdida. A ordem dos atributos não pode ser controlada mas a ordem dos elementos pode, portanto, dados seqüenciais, meta-dados e outras informações que não fazem sentido algum fora do elemento correspondente devem ser atributos. Atributos também são preferidos em algumas aplicações do XML que oferecem as duas formas para representar dados como uma forma de economia de espaço, em sacrifício da legibilidade (o SVG é um exemplo).

Identificadores permitidos para elementos e atributos

Nem todos os caracteres aceitos em documentos XML podem ser usados para construir nomes de elementos e atributos, embora a especificação seja flexível e admita inclusive caracteres de todo o alfabeto Unicode (o que inclui caracteres japoneses, chineses , árabes ,etc).

Nomes de elementos e atributos podem conter qualquer caractere alfanumérico de qualquer alfabeto mais os caracteres sublinhado "_", hífen "-" e ponto ".". Outros símbolos não são permitidos. O caractere ":" (dois pontos) é usado dentro de nomes de elementos apenas para separar o prefixo de um namespace do seu nome local.

Elementos e atributos não podem começar com um número, um ponto ou um hífen. Podem começar apenas com letras, ideogramas e sublinhados. Nomes não podem ter espaços. Não há limites para o tamanho do nome. Os elementos abaixo são bem formatados:

```
<_1_/>
<cdd:gen.inf cdd:cod="005">Introdução a XML</cdd:gen.inf>
```

Já esses outros elementos não são permitidos

```
<3-intro>Fundamentos</3-intro>
<cartão de crédito>321654987987654</cartão de crédito>
```

Conteúdo misto

Nos exemplos mostrados até agora, os elementos sempre têm contido ou outros elementos ou texto puro. É comum, principalmente em documentos que descrevem textos como artigos, livros, páginas Web, a existência de elementos que contém texto e elementos misturados. Um exemplo é mostrado a seguir. O elemento <paragrafo> contém tanto texto como elementos <definicao>.

```
<trecho>
  <secao>2</secao>
  <paragrafo>A unidade de informação dentro de um documento XML
    é o <definicao>elemento</definicao>. Um elemento é formado
    por duas <definicao>etiquetas</definicao> que atribuem algum
    significado ao conteúdo. </paragrafo>
</trecho>
```

Outros elementos

Esta seção descreve alguns outros elementos opcionais que podem aparecer em um documento XML.

Referências de entidade

Alguns caracteres não podem ser lidos em um documento XML. Esses caracteres incluem o sinal de menor < e o e-comercial &. Para produzir esses caracteres é preciso recorrer a referências de entidade. Referências de entidade são consideradas elementos de marcação. Quando uma aplicação interpreta um documento XML, as entidades são substituídas pelo seu conteúdo de substituição.

Há cinco referências pré-definidas em XML e, através de um DTD pode-se definir mais outras. As pré-definidas são:

- *<* que corresponde a <
- *>* que corresponde a >
- *&* que corresponde a &
- *"* que corresponde a “
- *'* que corresponde a '

Apenas < e & são obrigatórios. As outras são opcionais. ' e " são úteis dentro de strings (por exemplo, dentro de atributos) mas sempre é possível usar aspas dentro de valores entre apóstrofes ou apóstrofes dentro de valores entre aspas e evitar problemas sem reduzir a legibilidade do documento.

Referências de caracteres

Através de uma sintaxe semelhante à usada para incluir entidades é possível incluir caracteres Unicode. A diferença é que no lugar do & escreve-se &#, seguido do número do caractere na tabela Unicode (geralmente em hexadecimal). A sintaxe é &#xNNNN; onde NNNN é o código Unicode do caractere desejado, em hexadecimal. Os números 00NN correspondem ao alfabeto ISO-Latin-1 (ISO-8859-1). Há uma tabela com todos os caracteres Unicode em www.unicode.org/charts/.

Seções CDATA

Seções CDATA servem para embutir, em qualquer lugar da página, texto que deve ser tratado como caractere e que não pode ser tratado como XML. O conteúdo dentro de blocos CDATA é tratado como texto crú e não é interpretado. É ótimo para incluir texto que contém sinais < e > que precisam ser escapados em XML.

Uma seção CDATA é identificada por um bloco que começa com <!CDATA[e termina com]]>. O bloco pode conter qualquer coisa menos a seqüência]]> e o texto será

preservado da maneira como foi definido. O exemplo a seguir mostra um trecho de um documento que usa o bloco CDATA:

```
<titulo>Curso de XML</titulo>
<exemplo>Considere o seguinte trecho de XML: <![CDATA[
  <empresa>
    <nome>José & Silva S/A</nome>
  <empresa>
]]></exemplo>
```

O trecho em negrito não será considerado XML mas texto crú. Se não fosse usado CDATA todos os símbolos <, > e & teriam que ser substituídos por entidades <, > e &.

Instruções de processamento

Instruções de processamento servem para embutir instruções que serão utilizadas por algum programa que irá interpretar a página. Se um programa não reconhecer a instrução ele simplesmente ignora. Se reconhecer, ele pode utilizar os dados contidos na instrução para realizar alguma tarefa especial.

As instruções têm duas partes. O nome ou alvo (target), que identifica a instrução e permite que programas localizem-na, e os dados, que ocupam o resto da instrução.

```
<?nome-do-alvo área de dados ?>
```

A maneira como a instrução utiliza o espaço de dados é totalmente dependente da aplicação que irá utilizá-la. Por exemplo, um programa poderia utilizar uma instrução SQL guardada em uma instrução de processamento:

```
<?query-sql select nome, email from agenda where id=25 ?>
```

Instruções de processamento podem aparecer em qualquer lugar do documento. Um programa pode exigir que determinada instrução esteja em determinados lugares do documento.

Declaração XML

A declaração XML é sempre opcional mas é recomendado que documentos XML comecem com essa declaração. Ela passa a ser obrigatória quando o documento utiliza caracteres que pertencem a um alfabeto diferente do Unicode UFT-8. Nessas situações deve haver uma instrução de processamento antes dos caracteres a serem utilizados. Como na maior parte das vezes, um documento usa um único formato de codificação, aproveita-se a declaração XML para informar o alfabeto usado através do atributo encoding.

```
<?xml encoding="iso-8859-5" ?> <!-- a partir daqui, texto em
                  cirílico -->
```

Documento Bem-formado

Um documento bem formado segue algumas regras básicas, independente das regras que seu DTD possa ditar. Essas regras incluem:

- ter um único elemento raiz

- etiquetas iniciais e finais combinam (levando em conta que caracteres maiúsculos são diferentes)
- elementos bem aninhados
- valores de atributos entre aspas ou apóstrofes
- atributos não repetidos
- identificadores válidos para elementos e atributos
- comentários não devem aparecer dentro das etiquetas
- sinais < ou & nunca devem ocorrer dentro dos valores dos atributos ou nos nós de texto do documento.

Para garantir que seu documento é bem formado, rode um parser (interpretador XML) antes de publicá-lo. Não precisa ser um parser validador. Até mesmo um browser que suporta XML como o Firefox ou IE podem ser usados. Algumas outras alternativas é usar um parser de linha de comento como o Xerces (xml.apache.org/Xerces/).

Namespaces

Como XML não limita os elementos que podem ser usados em uma página, pode-se construir um documento misturando vocabulários de uma, duas ou mais aplicações XML e aproveitar os benefícios de todas juntas.

Um problema que surge nesse momento é o que fazer se houver um conflito. Por exemplo, suponha que você queira misturar a aplicação que define cartões de visita com outra que contém os dados da empresa. E então você descobre que aquela aplicação também tem os elementos <nome> e <endereco> só que são usados para descrever o nome e endereço da empresa. O que fazer?

Antes da existência dos namespaces não havia outra alternativa a não ser reescrever a aplicação. Chegou a ser sugerido que se usasse um nome mais longo e que ele ti esse um prefixo, separado do nome principal por um ponto, por exemplo:

```
<MeusDocsXML.nome>Fulano de Tal</MeusDocsXML.nome>
```

Mesmo assim, poderia haver conflito. Não haveria como garantir que nunca houvesse um outro elemento com o mesmo nome. Além disso, os arquivos se tornariam grandes e difíceis de ler.

A solução para esses problemas está na especificação XML Namespaces. Ela foi proposta algum tempo depois que o padrão XML já estava consolidado, mesmo assim, não foi necessário criar uma nova versão de XML (a versão continua sendo a 1.0). Pela especificação, um elemento poderia ter um identificador unívoco para cada elemento e atributo da sua aplicação. Esse identificador seria declarado em cada elemento e valeria para todos os elementos que contivesse.

Mas às vezes elementos de aplicações diferentes se misturam de uma maneira mais difícil de isolar. Eles podem se entrelaçar. Você pode ter o elemento de uma aplicação dentro do elemento de outra. Nesses casos seria necessário identificar cada elemento com seu namespace e não simplesmente “herdar” de um escopo.

Esse identificador se somaria ao nome do elemento formando um nome maior que jamais se repetiria. Para evitar que o identificador tivesse que ser repetido em cada elemento (o identificador poderia ser muito grande, ter caracteres ilegais e ser difícil de ler) foi estabelecido que o identificador seria associado ao nome através de um prefixo, usando o sinal de : (dois pontos) como separador. Como”:” já fazia parte dos caracteres lefais para nomes de elementos, a especificação não precisou ser mudada.

Portanto, criar um namespace significa declarar, dentro do escopo de um elemento (que pode ser o elemento raiz) uma URI como identificador de um espaço onde certos nomes serão reconhecidos. A URI é um identificador unívoco, portanto é uma boa escolha. Foi criado um atributo especial, que pode ser usado em qualquer elemento, para realizar essa declaração: o atributo xmlns. Veja um exemplo:

```
<raiz>
  <empresa xmlns="urn:cgc:01.234.567/0001-89/acme">
    <nome>Acme S.A.</nome>
    <endereco>Rua Boa Vista, 22 – Centro</endereco>
  </empresa>
  <cartao>
    <nome>Beltrano do Prado</nome>
    <endereco>Rua Bela Vista, 45 – Bom Retiro
    02837-102 – São Paulo - SP</endereco>
  </cartao>
</raiz>
```

Agora o nome e endereço de empresa não mais entram em conflito com o nome e endereço de cartão. A URI usada foi o CGC da empresa, que é unívoco de forma que não deve aparacer outro elemento nome que seja igual. O elemento nome da empresa agora se chama:

```
<{urn:cgc:01.234.567/0001-89/acme}:nome>
```

Mas se eu quiser incluir a empresa dentro do cartão de visitas? Nesse caso eu não teria um escopo bem definido pois os elementos se misturariam. Neste caso, pode-se usar o prefixo do namespace. A declaração via atributo xmlns, então tem que conter o prefixo da forma xmlns:prefixo.

```
<cartao xmlns:empresa="urn:cgc:01.234.567/0001-89/acme">
  <nome>Beltrano do Prado</nome>
  <endereco>Rua Bela Vista, 45 – Bom Retiro
  02837-102 – São Paulo - SP</endereco>
  <empresa:nome>Acme S.A.</empresa:nome>
  <empresa:endereco>Rua Boa Vista, 22 –
  Centro</empresa:endereco>
</cartao>
```

O namespace vale agora para todo o elemento e seus descendentes, desde que eles tenham um prefixo empresa:. Os elementos que não têm prefixo continuam pertencendo ao namespace *default*.

Pode-se declarar um namespace em qualquer elemento. Ele se aplica ao próprio elemento e aos descendentes. Se a declaração foi feita com prefixo, o próprio elemento que declara o namespace também precisa ter o prefixo. No exemplo abaixo, declaramos outro namespace, no mesmo elemento <cartao>, para que identificar os elementos que pertencem ao namespace do cartão. Como URI foi usada uma URL:

```
<car:cartao xmlns:empresa="urn:cgc:01.234.567/0001-89/acme"
            xmlns:car="http://www.provedor.com/belfin">
  <car:nome>Beltrano do Prado</car:nome>
  <car:endereco>Rua Bela Vista, 45 – Bom Retiro
  02837-102 – São Paulo - SP</car:endereco>
  <empresa:nome>Acme S.A.</empresa:nome>
  <empresa:endereco>Rua Boa Vista, 22 –
  Centro</empresa:endereco>
</car:cartao>
```

A URL serve apenas como identificador. Não tem relação alguma com uma URL real (não precisa sequer existir).

Com namespaces um nome de elemento agora tem duas partes: a parte local e o prefixo. O nome formado pela parte local e pelo prefixo é o nome qualificado. Como DTDs existem antes da invenção dos namespaces, eles não reconhecem a parte local e o prefixo mas reconhecem o nome qualificado. Portanto, para validar o documento, preciso levar em conta o prefixo.

Atributos também podem ser misturados com XML. Um exemplo comum são os atributos da especificação Xlink. Xlink é uma especificação só de atributos, portanto têm um namespace definido (a URL http://www.w3c.org/1999/xlink) e geralmente recebe o prefixo xlink. É preciso declarar o namespace sempre (e, nesse caso, pode até ser com outro prefixo). O namespace também pode ser declarado no DTD, para que os autores das páginas não precisem fazê-lo.

Atributos que começam com o prefixo xml: são reservados e podem ser aplicados em qualquer elemento de qualquer aplicação do XML. Os mais importantes são xml:lang, que recebe um nome de duas letras (ou duas letras, um hífen e outras duas letras – Ex: "pt-BR") que representa o idioma em que está escrito o conteúdo do elemento. Outro atributo é xml:space, que pode receber o valor "preserve" para manter os espaços extras (tabulações, novas-linhas, etc) dentro do elemento.

Validação

Para criar uma aplicação XML é preciso definir um vocabulário limitado de elementos e atributos e regras de formatação que sejam comuns a todos os documentos escritos na nova linguagem. Em outras palavras, é preciso definir uma classe de documentos. Uma das formas de fazer essa especificação é através da construção de um DTD ou Document Type Definition. O DTD utiliza uma sintaxe SGML, diferente da sintaxe XML. Em compensação utiliza poucos elementos (dos quais apenas três ou quatro são mais usados). Outra saída é usar uma linguagem-aplicação do XML como o XML Schema, mais abrangente. Aqui veremos como usar o DTD para especificar uma aplicação XML.

Porque validar?

Embora XML admita grande liberdade na definição e disposição dos elementos, não requer que todos os programas que usem dados em XML tenham que lidar com toda essa flexibilidade. Programas que aceitem qualquer documento XML de forma genérica, embora possam ler sua árvore de dados, inserir e remover elementos, não são capazes de entender o que fazer dos dados que encontra em determinados elementos, nem de dizer se faz algum sentido certos elementos estarem contidos em outros elementos.

È importante que, dentro de uma aplicação específica da linguagem XML, elementos sejam submetidos a uma especificação mais rigorosa que descreva a estrutura de documentos do mesmo tipo. Uma definição do tipo do documento é importante para programas que esperam lidar com dados organizados de uma maneira padronizada. É essencial para que se possa criar documentos novos.

Por exemplo, um programa que lê uma lista de registros de uma mala direta composta de uma lista de elementos <cliente> deve saber distinguir um elemento <nome> de um elemento < fax> para que saiba qual usar. Se o documento contiver elementos que ele não conhece, ou se todos os <fax> aparecerem amontoados em um lugar em vez de estarem dentro de um <cliente>, junto com o <nome>, o programa poderá não funcionar da forma esperada. Uma descrição da estrutura e sintaxe geral dos documentos válidos por essa aplicação específica poderia ser usada por programas geradores para que eles sempre gerassem documentos válidos, com uma certa margem de flexibilidade que não prejudicasse os programas que tivessem que ler os dados. A mesma descrição pode ser usados por programas consumidores, para validar o documento e decidir se rejeita possíveis erros ou se os ignora.

O esquema da linguagem, portanto, é o modelo genérico ou especificação que descreve toda a classe de documentos que serão considerados documentos da linguagem. Cada documento individual é uma instância que pode ou não usar todo o vocabulário de elementos previstos na aplicação. O esquema estabelece limites para a construção de novas instâncias e permite que se construa programas que saibam lidar com documentos que ainda não foram criados.

O esquema define todos os elementos que são válidos na linguagem, onde podem ser usados no contexto dos outros elementos, quantas vezes podem aparecer, em que ordem, se podem conter texto ou apenas determinados elementos, se têm atributos, que valores podem ter os atributos, se são obrigatórios.

O processo de validação compara um documento com seu esquema associado. O programa pode então decidir o que fazer. Ignorar os erros ou rejeitar o documento.

Validação com DTD

Um documento XML válido inclui uma declaração de tipo de documento (document type declaration) que identifica o seu DTD. O DTD pode ser um arquivo localizado no sistema do usuário ou em algum lugar da rede ou estar embutido em algum lugar do programa. O segundo caso é comum em programas criados para lidar com aplicações específicas do XML.

DTD representa uma classe de documentos XML. É a especificação da linguagem usada por uma determinada aplicação do XML. Pode conter itens que o documento não está usando, mas o documento não pode conter itens que não estão definida no DTD. O DTD, portanto, representa o universo de todos os documentos que podem ser criados. Um documento pode, no máximo, ter uma estrutura idêntica à definida no DTD.

DTSs são limitados. Não descrevem totalmente o documento. Oferece apenas mecanismos para uma validação genérica. Detalhes como faixa de valores numéricos aceitáveis para um campo ou tipos de dados específicos (como por exemplo, um atributo que só aceita URLs) não são controlados pelo DTD. Uma aplicação freqüentemente irá precisar validar detalhes mais específicos ao gerar um XML usando recursos de programação.

A validação é um procedimento opcional. A maior parte dos documentos não irá precisar de um DTD. O DTD oferece a possibilidade de trabalhar com uma classe de documentos. Nas situações onde isto não é necessário, onde o documento é manipulado individualmente, o DTD é dispensável. É possível, por exemplo, aplicar um estilo de

desenhar uma página específica com dados obtidos de um documento XML sem DTD. Será mais difícil, sem DTD, fazer um programa que aplique o estilo automaticamente em qualquer página.

Um exemplo simples

Considere o documento XML abaixo. Este documento é uma instância específica de uma classe de documentos que descreveremos com um DTD:

```
<pessoa>
  <nome>
    <prenome>Fulano</prenome>
    <sobrenome>de Tal</sobrenome>
  </nome>
  <profissao>Programador</profissao>
  <profissao>Analista de Suporte</profissao>
</pessoa>
```

Suponha que esta instância específica se chame pessoa.xml. O DTD abaixo pode ser usado para descrever essa instância:

```
<!ELEMENT pessoa (nome, profissao*)>
<!ELEMENT nome (prenome, sobrenome)>
<!ELEMENT prenome (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT profissao (#PCDATA)>
```

O DTD pode ser armazenado em um arquivo separado, ou pode ser embutido no próprio arquivo .xml. Como representa uma classe de documentos, faz mais sentido defini-lo como um arquivo externo para que possa ser reutilizado. Poderíamos chamá-lo de pessoa.dtd, por exemplo. A extensão .dtd é geralmente usada e o tipo MIME, caso o DTD seja enviado em um data stream, deve ser application/dtd.

Todas as linhas desse DTD simples contém uma declaração de elemento. A declaração informa o nome do elemento e, entre parênteses, o que ele pode conter. O elemento raiz não é identificado dentro do DTD mas na declaração <!DOCTYPE> de cada documento que implementa o DTD. No prolog do documento pessoa.xml deve haver a seguinte declaração:

```
<!DOCTYPE pessoa SYSTEM "http://pessoas.com/pessoa.dtd">
```

Observe que não há "=" entre SYSTEM e a URL que aponta para o DTD. A sintaxe usada não é XML.

De acordo com o DTD mostrado, documentos válidos devem conter um elemento pessoa, que é o elemento raiz do documento (de acordo com a declaração DOCTYPE). Elementos <pessoa> devem conter necessariamente um nome, seguido de zero ou mais elementos <profissao>. Cada elemento <nome> deve conter um elemento <prenome> seguido de um elemento <sobrenome>. Os elementos <prenome>, <sobrenome> e <profissao> devem conter texto (Parsed Character Data). Esse texto poderá conter entidades que serão processadas (parsed) antes do uso.

De acordo com o DTD, o documento abaixo é válido, pois <profissao> é opcional:

```
<pessoa>
  <nome>
    <prenome>Beltrano</prenome>
    <sobrenome>da Silva</sobrenome>
  </nome>
</pessoa>
```

Porém este outro não é porque omite o sobrenome:

```
<pessoa>
  <nome>
    <prenome>Ciclano</prenome>
  </nome>
  <profissao>Scheduler</profissao>
</pessoa>
```

Documentos válidos sempre contêm uma referência para o DTD. Esta referência pode ser do tipo SYSTEM, e apontar para um arquivo localizado em algum lugar do sistema ou da rede, como foi mostrado, ou do tipo PUBLIC. Declarações do tipo PUBLIC possuem uma identificação que associa o documento a uma DTD conhecida (geralmente um padrão). O software deve reconhecer o identificador. É comum oferecer também uma URL alternativa para o caso do software não reconhecer o identificador:

```
<!DOCTYPE HTML PUBLIC "-//WC3//DTD HTML 4.0//EN"
                    "http://www.w3.org/TR/REC-html40/strict.dtd">
```

O exemplo acima é o DOCTYPE usado nos documentos HTML 4.0. O browser reconhece o identificador PUBLIC. O DTD completa o HTML 4.0 (não é XML) pode ser encontra em http://ww.w3.org/TR/1998/REC-html40-19980424/sgml/dtd.html.

Um DTD pode ser embutido em um documento. Pode também ser parcialmente embutido, ou seja, pode devinir parte da especificação localmente e parte externamente. O exemplo abaixo mostra um DTD embutido:

```
<?xml version="1.0" encoding="iso-885901" ?>
<!DOCTYPE pessoa [
  <!ELEMENT pessoa (nome, profissao*)>
  <!ELEMENT nome (prenome, sobrenome)>
  <!ELEMENT prenome (#PCDATA)>
  <!ELEMENT sobrenome (#PCDATA)>
  <!ELEMENT profissao (#PCDATA)>
]>
<pessoa>
  <nome>
    <prenome>Beltrano</prenome>
    <sobrenome>da Silva</sobrenome>
  </nome>
```

```
</pessoa>
```

Para embutir parcialmente:

```
<!DOCTYPE pessoa SYSTEM “pessoa.dtd”[
  <!ELEMENT nome (#PCDATA)>
]>
```

O DTD local não pode sobrepor declarações de elementos feitas em documentos externos mas pode redefinir entidades. O código acima só seria aceito se o DTD externo não declarasse <nome>.

Agora veremos a seguir maiores detalhes de cada um dos elementos de um DTD.

<!ELEMENT>

Cada elemento da página precisa ser declarado usando <!ELEMENT>. A sintaxe é:

```
<!ELEMENT nome_do_elemento (conteudo) >
```

O conteúdo pode ser (1) #PCDATA, (2) uma seqüência de elementos filho, (3) uma lista de elementos de onde um pode ser escolhido, (4) conteúdo misto, (5) a palavra EMPTY ou (6) a palavra ANY.

1) #PCDATA (parsed character data) indica que o elemento pode conter texto, entidades, mas nenhum elemento.

2) Uma seqüência de elementos filho (cada um deles também terá que ser declarado). Pode-se especificar quantos elementos-filho serão aceitos usando os símbolos opcionais *, +, e ? após o nome do elemento. A ausência desses sinais significa que o elemento deve aparacer exatamente uma vez, na posição indicada relativa aos outros elementos da lista. O significado dos símbolos é o seguinte:

- * zero ou mais
- + um ou mais
- ? zero ou um

Por exemplo, de acordo com a declaração <!ELEMENT> abaixo:

```
<!ELEMENT cartao (nome, telefone+, email*, website?)>
```

Um cartão pode conter exatamente um nome, um ou mais telefones, zero ou mais email e um ou nenhum website. Neste outro exemplo, um <trem> tem exatamente uma locomotiva mas um número ilimitado de vagões:

```
<!ELEMENT trem (locomotiva, vagoes*)>
```

3) Lista de elementos para escolha. Uma lista de elementos separados por | indica que somente um deles poderá ser usado. Por exemplo:

```
<!ELEMENT trecho (ferroviário | aéreo | rodoviário |
fluvial)>
```

Usando os parênteses, pode-se elaborar escolhas mais complexas:

```
<!ELEMENT circulo (centro , ( raio | diâmetro))>
```

Um <circulo> deve conter um <centro> seguido de ou um <raio> ou um <diâmetro>. Outro exemplo:

```
<!ELEMENT ponto ((x, y) | (r, θ))>
```

Um <ponto> pode ser representado por suas cartesianas <x> e <y> ou por suas coordenadas polares <r> e <θ>.

A seguinte declaração

```
<!ELEMENT nome (prenome | sobrenome | (prenome | inicial*,
sobrenome))>
```

aceita que <nome> possa conter somente um <prenome>, somente um <sobrenome> ou <nome) seguido de uma ou mais iniciais opcionais, seguidas de um <sobrenome>.

4) Conteúdo misto. Consistem de uma escolha contendo #PCDATA, que deve ser o primeiro item da lista. Toda a lista deve terminar com um *. A declaração de conteúdo misto declara quais os elementos que podem aparacer dentro do bloco mas não garante que eles irão aparecer, não os limita nem estabelece qualquer ordem.

```
<!ELEMENT nome (#PCDATA | enfase)* >
```

A declaração acime aceita textos deste tipo:

```
<nome>Pedro de <enfase>Orleans e Bragança</enfase></nome>
```

Não se pode impor limite nem ordem aos outros elementos. Não se pode usar parênteses internos para definir grupos, ordens, etc.

5) Elementos vazios devem ser declarados como EMPTY.

```
<!ELEMENT nome EMPTY>
```

A declaração acime permite elementos como

```
<nome prenome=”José” />
```

Quaisquer atributos precisam ser declarados usando <!ATTLIST>.

6) ANY pode ser usado para indicar que o elemento pode conter qualquer coisa. É útil no processo de criação de um DTD mas não tem muita utilidade na fase de uso.

<!ATTLIST>

Cada elemento que possua atributos deve tê-los declarados em um elemento<!ATTLIST>. Há mais de uma sintaxe permitida. Para declarar um atributo use:

```
<!ATTLIST elemento atributo tipo valor_default >
```

Para declarar múltiplos atributos de um mesmo elemento pode-se definir vários elementos <!ATTLIST> ou um só com atributo, tipo e valor default em seqüência, da forma:

```
<!ATTLIST elemento atributo tipo valor_default
          elemento atributo tipo valor_default ... >
```

Cada declaração está associada a um elemento. Se o mesmo atributo ocorre em vários elementos, sua declaração precisa ser repetida em cada elemento.

Exemplos:

```
<!ATTLIST voo de NMTOKEN #IMPLIED >
<!ATTLIST voo numero CDATA #FIXED “12/3283-1” >
<!ATTLIST voo para (REC | CGH | GRU | SDU) #REQUIRED >
<!ATTLIST voo transportador (RG | JH) “RG” >
```

ou

```
<!ATTLIST voo de NMTOKEN #IMPLIED
              numero CDATA #FIXED “12/3283-1”
              para (REC | CGH | GRU | SDU) #REQUIRED
              transportador (RG | JH) “RG” >
```

ATTLIST: Tipos de dados

O DTD permite controle maior sobre conteúdo de atributos que sobre o conteúdo de elementos, no entanto, só á 10 tipos de dados disponíveis (nenhum para distinguir de strigs). Os tipos são

- CDATA
- NMTOKEN
- NMTOKENS
- Enumerarion
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NOTATION

CDATA (character data) representa qualquer texto arbitrário. Pode conter caracteres de qualquer tipo (exceto > e & que devem usar entidades).

NMTOKEN (name token). É semelhante a um nome de elemento ou atributo, mas tem menos restrições (pode ser, por exemplo, um número). Não pode conter espaços. Todos nome (identificador de elementos ou atributos) legal é um NMTOKEN mas nem todo NMTOKEN é um nome.

NOTOKENS representa um ou mais nmtoken separados por espaços, Por exemplo:

```
<elemento atr=”um dois três 56 21” >
```

Enumeration representa uma lista de NMTOKENS de onde pode-se escolher o valor do atributo. As escolhas são separadas por “|” :

```
<!ATTLIST voo para (REC | CGH | GRU | SDU) #REQUIRED >
```

Elementos do tipo **ID** tem que conter um nome (e não nmtoken) que seja unívoco no documento. Por este motivo, não se pode usar um número como ID. A solução é colocar um prefixo antes do número que seja ou uma letra ou um sublinhado. Elementos só podem ter um tipo ID.

```
<!ATTLIST voo codigo ID #REQUIRED >
<!ATTLIST piloto numero ID #REQUIRED >
```

IDREF é a referência para o ID. É análogo ao HREF. Vários elementos podem ter o mesmo IDREF, mas o ID é único dentro do documento. IDREF é uma solução para relacionar elementos com outros quando eles não podem ser pai e filho. Por exemplo, a seguinte declaração:

```
<!ATTLIST piloto alocado IDREF #REQUIRED >
```

permitiria a ligação entre vôos e pilotos:

```
<aeroporto>
  <voo codigo="RG123">...</voo>
  <voo codigo="RG456">...</voo>
  <piloto numero="S1" alocado="RG123">...</piloto>
  <piloto numero="S2" alocado="RG456">...</piloto>
</aeroporto>
```

IDREFS suporta uma lista de referências de ID. No exemplo mostrado antes, um <piloto> poderia ser alocado em vários vôos se o atributo alocado fosse do tipo IDREFS em vez de IDREF:

```
<aeroporto>
  <voo codigo="RG123">...</voo>
  <voo codigo="RG456">...</voo>
  <piloto numero="S1" alocado="RG123 RG456">...</piloto>
</aeroporto>
```

Um atributo do tipo **ENTITY** recebe uma referência de entidade externa. Uma entidade externa pode ser um arquivo, uma imagem, etc. Entidades externas são diferentes das entidades internas porque não são processadas pelo parser XML. Não é possível usar referências (&nome;) para ter acesso a entidades externas. Elas só são acessíveis em atributos, e não contém outros caracteres além do nome. Suponha que exista uma entidade externa chamada “imagem”, um atributo que aceita este tipo teria a seguinte declaração:

```
<!ATTLIST elemento atributo #REQUIRED >
```

e poderia fazer referência à entidade da seguinte maneira:

```
<elemento atributo="imagem" />
```

ENTITIES aceita uma lista de entidades externas separadas por espaços (da mesma forma que IDREFS e NMTOKENS).

O tipo **NOTATION** é raramente utilizado. Se houver algum elemento NOTATION declarado no DTD, um atributo pode referenciar seu nome. Suponha que exista duas variáveis NOTATION amazon e barnes, respectivamente contendo as URLs das duas livrarias, um ATTLIST poderia usá-las da forma:

```
<!ATTLIST elem atrib NOTATION (amazon | barnes) #REQUIRED >
```

ATTLIST: valores default

Há quatro possíveis valores default:

#REQUIRED: força o autor do documento a definir o atributo.

#IMPLIED: o atributo é opcional.

#FIXED: o atributo tem um valor fixo (pode ser CDATA) que é informado logo em seguida. No exemplo abaixo, o namespace do Xlink foi predefinido no DTD para o elemento <bibliography>, de forma que não será necessário fazê-lo no documento XML:

```
<!ATTLIST biblipgraphy xmlns:xlink CDATA #FIXED
                    "http://www.w3.org/1999/xlink" >
```

Valor literal: parecido com #FIXED. Define um valor default caso o autor do documento não defina o atributo

```
<!ATTLIST voo transportador (RG | JH) "RG" >
```

<!NOTATION>

Serve para descrever uma notação usada no texto, associando-a com uma descrição ou outra informação significativa. Uma aplicação é oferecer uma variável no lugar de um valor que representa um identificador externo. Por exemplo:

```
<!NOTATION amazon SYSTEM "http://www.amazon.com" >
<!NOTATION barnes SYSTEM "http://www.bernesandnoble.com" >
```

A variável pode ser usada em outras partes do DTD.

<!ENTITY>

Além das cinco entidades fornecidas pelo XML, é possível definir novas entidades no DTD usando <!ENTITY>. Há vários tipos de entidades. Existem as entidades processadas (que são interpretadas pelo parser) e as não processadas. Existem entidades definidas internamente (no DTD) ou externamente (em um arquivo externo).

Entidades internas processadas (parsed internal entities)

A referência deve ser um nome e o valor pode ser qualquer texto inclusive trechos bem formados de XML.

```
<!ENTITY empresa "ACME Indústria &amp; Comércio de Coisas S.A.">
<!ENTITY copyright "<table>
  <tr>
    <td>Copyright &#x00A9 2006</td>
  </tr>
</table>">
```

O documento pode conter:

```
<texto> Visite a &empresa; ainda hoje!. </texto>
<div>&copyright;</div>
```

Entidades externas processadas (parsed external entities)

É possível definir uma entidade que contenha um documento XML bem construído externo. A declaração é idêntica à declaração de entidades internas exceto que no lugar do texto de substituição deve haver a palavra SYSTEM seguida de uma URI que aponte para o documento que contém o texto a ser representado pela entidade. Por exemplo:

```
<!ENTITY menu_sup SYSTEM "/fragmentos/menu_superior.xml">
```

A entidade pode agora ser usada dentro do documento como qualquer outra entidade:

```
<texto>Qualquer texto</texto>
&menu_sup;
<elemento> ...
```

Entidades externas não são permitidas dentro de atributos. Para recuperar a entidade externa, o parser deve ser um processador validador (parsers que não fazem validação não têm obrigação de buscar entidades externas).

Entidades externas não processadas (unparsed external entities)

É possível associar entidades com recursos externos que não podem ser processados. Essas entidades não podem ser chamadas a partir de referências mas apenas via valores de atributos (como uma variável para identificar o tipo do recurso externo.

```
<!NOTATION gif SYSTEM "image/gis">
<!ENTITY airbus SYSTEM "/gifs/airbus.gif" NDATA gif>
```

Para usar a referência, um elemento poderia fazer o seguinte:

```
<imagem fonte="airbus">
```

Cabe ao processador do documento fazer alguma coisa com a informação que ele tem (a URI da imagem e o tipo).

Entidades de parâmetros (parameter entities)

Entidades de referência não podem ser usadas dentro do DTD. Para isto existem *entidades de parâmetro*. A única diferença é que entidades de parâmetro são declaradas com um "%" e um espaço antes do nome e, quando são chamadas, são precedidas de um % em vez do &.

Entidades de parâmetro são úteis em situações onde as declarações são muito repetitivas:

```
<!ELEMENT voo de (REC | CGH | GRU | GIG | SDU)>
<!ELEMENT VOO para (REC | CGH | GRU | GIG | SDU)>
```

Para poder repetir menos pode-se declarar as entidades

```
<!ENTITY % aerosp "CGH | GRU">
<!ENTITY % aerorio "GIG | SDU">
<!ENTITY % aeroportos "REC | %aerorio; | %aerosp;">
<!ATTLIST voo de (%aeroportos;) #REQUIRED >
<!ATTLIST voo para (%aeroportos;) #REQUIRED >
```

Entidades de parâmetros externas (módulos)

É possível definir entidades de parâmetro contendo um DTD externo. Desta maneira pode-se importar DTDs menores e compor um maior. Vários DTDs públicos são distribuídos desta maneira. Um exemplo é o XHTML.

Para importar um DTD deve-se usar o elemento <!ENTITY> e atribuir o DTD externo a uma entidade de parâmetro interna, por exemplo:

```
<!ENTITY % tabela_de_voos SYSTEM "voos.dtd">
```

Isto atribui toda a DTD à referência de entidade. Para importar a DTD é preciso chamar a referência. Isto pode ser feito simplesmente escrevendo a chamada em uma linha isolada:

```
&tabela_de_voos;
```

É possível importar também DTDs públicos. Talvez você queira usar HTML dentro do seu XML. O problema é validar o XHTML. Para isto pode-se importar todo o DTD do XHTML usando

```
<!ENTITY % xhtml PUBLIC "-//W3C//DTD XHTML 1.1//EN"
                 "xhtml/xhtml11.dtd">

%xhtml;
```

Uma vez importado o DTD, o XHTML permite que se aplique módulos a quaisquer elementos do seu XML. Os módulos são entidades de parâmetro que foram definidas no DTD importado. Por exemplo, suponha o seguinte documento:

```
<catalogo>
  <produto>Computador Pentium IV VT</produto>
  <configuracoes>
    <table border=1>
      <tr><td>120Gb</td><td>256RAM</td><td>2499,00</td></tr>
      <tr><td>180Gb</td><td>512RAM</td><td>3199,00</td></tr>
    </table>
  </configuracoes>
  <formulario>
    <form action="/enviar.php">
      <input type="texto" value="12584">
      <button type="submit">Enviar codigo</button>
    </form>
  </formulario>
</catalogo>
```

Para aplicar o módulo de tabelas no elemento <configuracoes> e o módulo de formulários no elemento <formulario> pode-se fazer:

```
<!ELEMENT configuracoes ((%xhtml-form.module;)*) >
<!ELEMENT formulario ((%xhtml-table.module;)*) >
```

No site do W3C há uma lista de todos os módulos disponíveis para o XHTML.

Importar o DTD não faz com que o novo documento receba as funcionalidades

existentes na aplicação original, ou seja, importar o módulo de suporte a JavaScript ou de tabelas não vai fazer o documento entender JavaScript ou saber montar uma tabela. Apenas permite que o documento possa ser validado. Porém, o uso desse documento por um programa que entenda XHTML poderá fazer com que ele interprete as partes que conhece.

De forma similar pode-se importar os DTDs de MathML, SVG e outros.

Inclusão condicional

Existem duas diretivas XML que servem para incluir ou ignorar elementos condicionalmente. Na verdade, servem para que de um DTD, se possa ligar ou desligar definições de elementos, atributos e entidades de um outro DTD.

<![IGNORE[...]]> serve para ignorar um trecho da DTD:

```
<![IGNORE [
  <!ELEMENT detalhes (#PCDATA)>
]]>
```

Usar IGNORE é a mesma coisa que não incluir o texto. funciona como um comentário (para comentar código na DTD também pode-se usar comentários XML). <!INCLUDE[...]]> serve indicar o trecho que deve ser interpretado:

```
<![INCLUDE [
  <!ELEMENT detalhes (#PCDATA)>
]]>
```

Usar INCLUDE é a mesma coisa que simplesmente escrever o trecho explicitamente sem o INCLUDE.

Usando da forma acime, INCLUDE e IGNORE parecem inúteis. A sua utilizade consiste em atribuir seus valores a uma entidade de parâmetro.

```
<!ENTITY % detalhamento “IGNORE”>
<![INCLUDE [
  <!ELEMENT detalhes (#PCDATA)>
]]>
```

A entidade pode ser redefinida em DTDs locais que importam DTDs externas e assim “ligar” ou “desligar” uma definição feita externamente:

```
<!ENTITY % detalhamento “INCLUDE”>
```

Namespaces

DTDs não suportam namespaces. Para declarar um elemento não basta usar apenas o nome local. É preciso usar o nome qualificado:

```
<!ELEMENT prefixo:elemento (#PCDATA) >
```