Universidade Federal do Rio Grande do Sul
Instituto de Informática
Programa de Pós–Graduação em Computação

Estudo de esquemas em XML

por
Vanessa de Paula Braganholo

Trabalho Individual

Carlos A. Heuser
Orientador

Porto Alegre, dezembro de 2000

Sumário

Lista de Figuras 4
Lista de Tabelas 5
Lista de Abreviaturas 6
Abstract 7
Resumo 8

1 Introdução 9
1.1 Estudo de Caso 10

2 DTD 13
2.1 Declaração de Elementos 13
2.1.1 Controle de Sequência 15
2.1.2 Controle de Cardinalidade 16
2.2 Declarações de Atributos 16
2.3 Entidades 18
2.3.1 Entidades Pré-Definidas 18
2.3.2 Entidades Gerais 18
2.3.3 Entidades Parâmetro 18
2.3.4 Entidades Externas 19

3 XML Schema 20
3.1 Conceitos Básicos 22
3.2 Derivação de Tipos 25
3.2.1 Derivação por Restrição 25
3.2.2 Derivação por Extensão 27
3.3 Referência 28
3.4 Grupos de Substituição e Tipos Abstratos 29
3.5 População, Unicidade, Chaves e Referência 29
3.6 Valores Nulos 30
3.7 Namespaces 30
3.7.1 Schema Location 31
3.7.2 Importando Tipos 32
3.7.3 Redefinindo Tipos 32

4 RDF 34
4.1 Modelo Básico 34
4.2 Sintaxe 35
4.2.1 Description, about e ID 36
4.2.2 Containers 37
4.2.3 Relações 39
4.3 RDF Schema 39
4.3.1 Sistema de Tipos 40

5 Outros Modelos 47
5.1 DSD 47
5.1.1 Restrições a elementos 47
5.1.2 Declarações de Atributos 47
5.1.3 Tipos *string* 48
5.1.4 Expressões de Conteúdo 48
5.2 DDML 51
5.2.1 Declaração de Elementos 52
5.2.2 Declaração de Atributos 52

6 Comparação e Conclusão 56

Referências Bibliográficas 60

Lista de Figuras

1.1 Modelo de classes para publicações e autores 10
1.2 Exemplo de documento XML 11
1.3 Representação para a população de um tipo em UML 12

2.1 DTD para publicações (publicacao.dtd) 14

3.1 Instância XML que utiliza `schemaLocation` para indicar a localização do esquema a ser utilizado 31

4.1 Grafo RDF para uma sentença 35
4.2 Grafo RDF com propriedade estruturada (complexa) 36
4.3 Exemplo de container 37
4.4 Referenciando um container 38
4.5 Referenciando cada membro de um container 38
4.6 Exemplo da utilização do atributo `aboutEachPrefix` 39
4.7 Relação ternária 39
4.8 Classes e Recursos como Conjuntos e Elementos 44

Lista de Tabelas

2.1 Exemplos de Entidades pré-definidas 18

3.1 Tipos Simples de XML Schema 23
3.2 Facetas definidas em XML Schema 27

4.1 Tipos de Objeto Container de RDF 37

5.1 Operadores aplicáveis à construção de um tipo *string* 48
5.2 Operadores aplicáveis à restrição do conteúdo de um elemento 49

6.1 Comparação entre UML, DTD, XML Schema e RDF 59

Lista de Abreviaturas

DDML	Document Definition Markup Language
DTD	Document Type Definition
DSD	Document Structure Description
RDF	Resource Description Framework
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
URL	Uniform Resource Locator
URI	Unified Resource Identifier

TITLE: A study of Schema Languages for XML

Abstract

DTD, XML Schema, RDF Schema and UML are four proposals for data modeling. This work is based in a case study that presents the modeling of an UML application and maps this application to DTD, XML Schema and RDF Schema. Despite the similarity in their names, RDF Schema and XML Schema have different roles. XML Schema, as DTD, prescribes the order and combination of tags in a XML document. On the other hand, RDF Schema provides only information about the interpretation of the sentences in a RDF data model, and does not constrain the syntactical appearance of a RDF description [BRO 00]. The goal of this work is to compare these four models, in a way that someone who knows one model can easily understand the others. Besides, this work briefly presents the concepts of two other schemata proposals for XML: DSD and DDML.

Keywords: Semi-structured data, schemata, XML, DTD, XML Schema, RDF(S), DSD, DDML

Resumo

DTD, XML Schema, RDF Schema e UML são quatro propostas para modelagem de dados. Esse trabalho baseia-se em um estudo de caso que apresenta a modelagem de uma aplicação UML e faz o mapeamento desta aplicação para DTD, XML Schema e RDF Schema. Apesar da similaridade de seus nomes, RDF Schema possui um papel diferente do de XML Schema. XML Schema, assim como a DTD, restringe a ordem e a combinação das *tags* em um documento XML. Ao contrário, RDF Schema apenas fornece informações sobre a interpretação das sentenças dadas em um modelo RDF e não fornece restrições sintáticas à descrição RDF [BRO 00]. Esse trabalho tem como objetivo comparar estas quatro propostas de modelagem de dados, de modo que uma pessoa que conheça um dos modelos possa compreender as outras mais facilmente. Além disso, esse trabalho apresenta brevemente os conceitos de outras duas propostas de esquemas para XML: DSD e DDML.

Palavras-chave: Dados Semi-Estruturados, Esquemas, XML, DTD, XML Schema, RDF

1 Introdução

Grande parte dos documentos eletrônicos encontrada na Web pode ser considerada uma fonte de dados. Documentos que tratam assuntos como medicina, economia e computação são encontrados aos milhares. Entretanto, esses dados não possuem uma estrutura rígida que os define, e por isso são denominados dados semi-estruturados [ABI 97] [DOR 00]. Apesar disso, podemos identificar três componentes distintos nesses documentos [McG 98]:

1. Dados: as palavras propriamente ditas.

2. Estrutura: o tipo de documento e a organização de seus elementos, por exemplo, um memorando, um contrato. Também, que tipo de elementos ele pode conter e em que ordem eles podem aparecer.

3. Apresentação: o modo em que a informação é apresentada ao leitor: em papel, na tela de um *browser* ou via sintetização de voz. Também, quais fontes ou inflexões de voz são usadas para cada tipo de elemento e assim por diante.

XML (*eXtensible Markup Language*) vem se estabelecendo como um padrão para representação de dados semi-estruturados e vem sendo utilizada para representar dados que serão manipulados por aplicações ou trocados através da Web. Segundo [McG 98], XML é uma linguagem computacional que descreve informações. Já [PIM 00] define XML como uma linguagem de marcação apropriada à representação de dados, documentos e demais entidades cuja essência fundamenta-se na capacidade de agregar informações.

Um diagrama de classes UML (*Unified Modeling Language*) é um padrão para modelagem de dados definido em [OMG 00]. Um documento XML pode representar instâncias das classes definidas em um modelo UML. Por conseqüência, pode-se definir modelos XML que representem esses dados, da mesma forma que um diagrama de classes UML o faz.

O objetivo deste trabalho é apresentar três propostas de esquemas para dados XML definidas pela W3C (*World Wide Web Consortium*): DTD, XML Schema [FAL 00] e RDF(S) [BRI 00], e compará-las com UML, de modo que uma pessoa que domina um dos modelos compreenda mais facilmente os demais. Para isso, supõe-se que o leitor conheça UML.

Os propósitos de UML, DTD, XML Schema e RDF são bastante diferentes. Por isso, não se pode dizer que um dos modelos é "melhor"que outro. Cada um é melhor para o propósito que foi projetado. UML descreve classes de objetos, seus relacionamentos e atributos. DTD é uma gramática que permite declarar elementos e atributos. XML Schema visa estabelecer regras que os documentos XML devem seguir [MIC 00]. Um documento XML é considerado uma instância de um XML Schema. Existem vários *parsers* que verificam se um determinado documento XML está de acordo com as restrições estabelecidas em seu XML Schema.

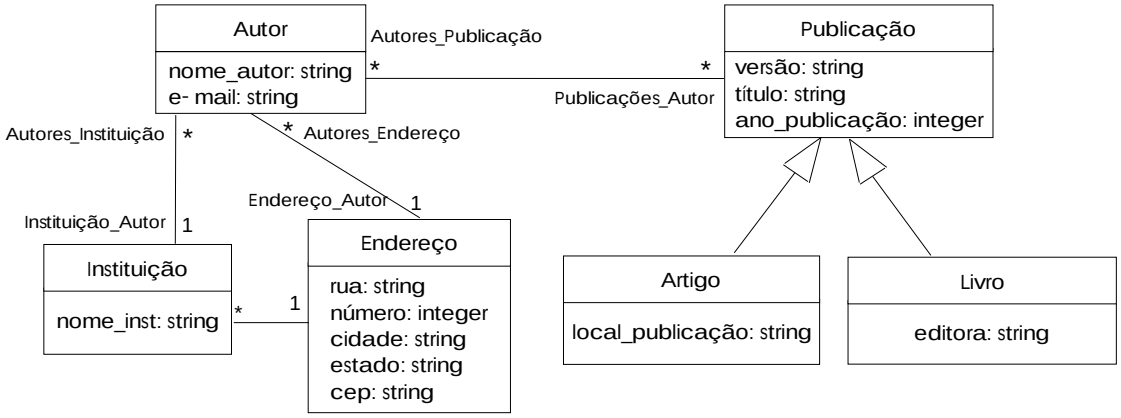


Figura 1.1: Modelo de classes para publicações e autores

Já o propósito de RDF(S) é descrever características, ou metadados, de recursos. Não existem *parsers* que verifiquem se uma descrição de instâncias RDF(S) estão de acordo com seu esquema. Os *parsers* existentes apenas verificam um esquema RDF(S) e fornecem as sentenças formadas pelas descrições dos recursos presentes no esquema.

1.1 Estudo de Caso

Para exemplificar a comparação será utilizada uma pequena aplicação em UML que trata de autores e publicações. O modelo de classes correspondente é apresentado na figura 1.1. Esse modelo será utilizado ao longo deste trabalho e será mapeado para DTD, RDF(S) e XML Schema, de modo a facilitar a comparação entre as três propostas.

A figura 1.2 apresenta um documento XML que possui os conceitos apresentados no modelo de classes da figura 1.1. Esse documento pode ser considerado uma representação das instâncias das classes definidas na figura 1.1, com algumas singularidades. Uma delas é que o modelo de classes da figura 1.1 não possui o conceito de `PUBLICACOES`, o que corresponde à população da classe **Publicação**. No documento XML, esse conceito teve que ser adicionado para permitir a representação de dados sobre várias publicações. Em UML, o conceito de população de uma classe pode ser representado por uma classe agregada, como mostra a figura 1.3.

Outra singularidade encontrada diz respeito à estrutura hierárquica de um documento XML. Foi necessário, portanto, decidir uma hierarquia para representar as instâncias das classes da figura 1.1: serão representadas as publicações com seus autores, ou uma lista de autores com suas publicações? Essa questão é importante, pois em UML um objeto nem sempre é uma hierarquia. Outra ponto importante é determinar se os atributos de uma classe UML serão mapeados para elementos ou atributos XML. Este trabalho adota a opção de elementos e reserva os atributos para representar metadados de uma classe.

Este trabalho está estruturado como segue. O capítulo 2 mostra as principais características de uma DTD. Os principais conceitos de XML Schema são apre-

```
<?xml version="1.0" encoding="US-ASCII"?>
<PUBLICACOES>
  <PUBLICACAO VERSAO="1">
    <TITULO>Título Artigo</TITULO>
    <ANO_PUBLICACAO>2000</ANO_PUBLICACAO>
    <AUTOR>
      <NOME_AUTOR>José</NOME_AUTOR>
      <E-MAIL>jose@instituicaoA.br</E-MAIL>
      <INSTITUICAO>
        <NOME_INST>InstituicaoB</NOME_INST>
        <ENDERECO>
          <RUA>Rua B</RUA>
          <NUMERO>2000</NUMERO>
          <CIDADE>Curitiba</CIDADE>
          <ESTADO>PR</ESTADO>
          <CEP>90000-001</CEP>
        </ENDERECO>
      </INSTITUICAO>
    </AUTOR>
    <AUTOR>
    ...
    </AUTOR>
    <LOCAL_PUBLICACAO>Anais Congresso</LOCAL_PUBLICACAO>
  </PUBLICACAO>
  <PUBLICACAO VERSAO="2">
    <TITULO>Título Livro</TITULO>
    <ANO_PUBLICACAO>1999</ANO_PUBLICACAO>
    <AUTOR>
      <NOME_AUTOR>Maria</NOME_AUTOR>
      <E-MAIL>maria@instituicaoA.br</E-MAIL>
      <ENDERECO>
         <CEP>93320-000</CEP>
      </ENDERECO>
      <INSTITUICAO>
        <NOME_INST>InstituicaoA</NOME_INST>
          <ENDERECO>
          <RUA>Rua A</RUA>
          <NUMERO>1000</NUMERO>
          <CIDADE>Porto Alegre</CIDADE>
          <ESTADO>RS</ESTADO>
          <CEP>90000-000</CEP>
        </ENDERECO>
      </INSTITUICAO>
    </AUTOR>
    <EDITORA>Editora 1</EDITORA>
  </PUBLICACAO>
  <PUBLICACAO>
  ...
  </PUBLICACAO>
</PUBLICACOES>
```

Figura 1.2: Exemplo de documento XML

Figura 1.3: Representação para a população de um tipo em UML

sentados no capítulo 3 e os de RDF(S) são mostrados no capítulo 4. O capítulo 5 apresenta brevemente outras duas propostas de esquema para XML: DSD e DDML. Finalmente, o capítulo 6 faz a comparação das três propostas e apresenta algumas conclusões.

2 DTD

A DTD é a característica mais significativa que XML herdou de SGML [BRA 00]. Sua estrutura está baseada em uma gramática que permite declarar elementos, atributos, entidades e notações. Através de regras estruturais, ela define os elementos que podem ser utilizados em um documento XML, quantas vezes eles podem aparecer e que hierarquia deve ser seguida.

Uma DTD é composta por um conjunto de declarações, as quais podem ser `ELEMENT`, `ATTLIST`, `ENTITY` e `NOTATION`. Todas as declarações iniciam com `<!` e terminam com `>`. Um `ELEMENT` define uma *tag* XML, ou seja, cada declaração `ELEMENT` define um elemento no documento XML. `ATTLIST` define uma lista de atributos pertencentes a um determinado elemento. `ENTITY` define uma entidade e `NOTATION` declara uma notação.

Uma DTD pode ser declarada dentro de um documento XML ou em um arquivo separado. A convenção é utilizar um documento separado com extensão *.dtd*.

No documento XML, a DTD é referenciada através da declaração `<!DOCTYPE ...>`, sendo que o nome que vem a seguir indica o elemento raiz do documento e a localização da DTD, como mostrado abaixo.

```
<? xml version="1.0" encoding="US-ASCII">
<!DOCTYPE PUBLICACOES SYSTEM publicacao.dtd>
<PUBLICACOES>
...
...
</PUBLICACOES>
```

No exemplo acima, entende-se que a DTD está localizada no mesmo diretório do arquivo XML. Pode-se também utilizar um caminho de diretório ou uma URL para informar a localização da DTD.

Dando início ao processo de comparação com UML, uma DTD, dentre as várias que podem ser utilizadas para representar o diagrama de classes UML da figura 1.1, é mostrada na figura 2.1. Ela será referenciada nesse trabalho como `publicacao.dtd`. Analisando esta DTD, pode-ser notar algumas características. As classes foram mapeadas para elementos complexos, com excessão das classes que fazem parte da hierarquia de herança, que foram mapeadas para entidades. Os atributos das classes foram mapeados para elementos. Os tipos dos elementos foram limitados a `PCDATA`. As declarações de elementos, atributos e entidades serão detalhadas nas seções seguintes.

2.1 Declaração de Elementos

Uma declaração de elemento é utilizada para definir um novo elemento e especificar o conteúdo permitido para ele. A declaração é feita do seguinte modo:

```
<!ENTITY % PUBLICACAO "(TITULO, ANO_PUBLICACAO, AUTOR+)">
<!ENTITY % ARTIGO "(%PUBLICACAO;,LOCAL_PUBLICACAO)">
<!ENTITY % LIVRO "(%PUBLICACAO;,EDITORA)">
<!ELEMENT PUBLICACOES (PUBLICACAO+)>
<!ELEMENT PUBLICACAO (%ARTIGO; | %LIVRO;)>
<!ATTLIST PUBLICACAO VERSAO CDATA #REQUIRED>
<!ELEMENT AUTOR (NOME_AUTOR, E-MAIL, ENDERECO?, INSTITUICAO?)>
<!ELEMENT NOME_AUTOR (#PCDATA)>
<!ELEMENT E-MAIL (#PCDATA)>
<!ELEMENT ENDERECO (RUA?, NUMERO?, CIDADE?, ESTADO?, CEP?)>
<!ELEMENT RUA (#PCDATA)>
<!ELEMENT NUMERO (#PCDATA)>
<!ELEMENT CIDADE (#PCDATA)>
<!ELEMENT ESTADO (#PCDATA)>
<!ELEMENT CEP (#PCDATA)>
<!ELEMENT INSTITUICAO (NOME_INST, ENDERECO?)>
<!ELEMENT NOME_INST (#PCDATA)>
<!ELEMENT TITULO (#PCDATA)>
<!ELEMENT ANO_PUBLICACAO (#PCDATA)>
<!ELEMENT LOCAL_PUBLICACAO (#PCDATA)>
<!ELEMENT EDITORA (#PCDATA)>
```

Figura 2.1: DTD para publicações (publicacao.dtd)

```
<!ELEMENT PUBLICACOES ...>
```

A palavra `!ELEMENT` identifica que um novo elemento está sendo declarado, e `PUBLICACOES` é o nome do elemento. Nomes de elementos devem iniciar com uma letra, um *underscore* "_"ou com dois pontos ":". Além disso, são *case sensitive*, ou seja, um nome com letras maiúsculas é diferente de um nome com letras minúsculas. Nomes também não podem possuir espaço e caracteres especiais. Os únicos caracteres permitidos são letras, números, `"."`, `"-"`, `"_"` e `":"`. Não existe restrição quanto ao tamanho dos nomes de elementos.

Depois do nome do elemento declara-se o tipo de conteúdo que ele deve ter. O conteúdo de um elemento pode ser atômico ou complexo [MEL 00b]. Elementos atômicos são aqueles que possuem somente texto ou são vazios. Elementos complexos são aqueles que possuem somente elementos filhos ou texto e elementos filhos misturados. Para cada um desses tipos de conteúdo existe um tipo de declaração correspondente. Elementos que não possuem conteúdo devem ser declarados como `EMPTY`.

```
<!ELEMENT figura EMPTY>
```

No documento XML, elementos vazios podem aparecer de duas formas. A *tag* inicial seguida pela *tag* final (`<figura></figura>`), ou simplesmente a *tag* inicial seguida de "/"(`<figura/>`).

O conteúdo de um elemento que pode conter somente texto deve ser ser declarado como `#PCDATA`.

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

Já elementos que podem conter tanto texto como elementos filhos devem ser declarados como `ANY`.

```
<!ELEMENT paragrafo ANY>
```

O elemento `paragrafo` poderia aparecer no documento XML como mostrado abaixo.

```
<paragrafo> Esta imagem <image/> demonstra ... </paragrafo>
```

A DTD também permite especificar quais são os filhos de um elemento, quantas vezes e em que ordem eles podem aparecer. Para isso, existem alguns caracteres para controle de sequência e outros para controle de cardinalidade [BRA 00].

2.1.1 Controle de Sequência

Quando um elemento possui vários subelementos, eles podem ser organizados de diferentes formas. Sua organização pode ser controlada por dois operadores de conexão lógica: `","` (conector de sequência) e `"|"` (conector de escolha).

A declaração

```
<!ELEMENT AUTOR (NOME_AUTOR, E-MAIL, ENDERECO?, INSTITUICAO?)>
```

estabelece que um elemento `AUTOR` possui quatro elementos filhos, e que esses devem aparecer exatamente na ordem determinada: `NOME_AUTOR` seguido de `E-MAIL`, seguido de `ENDERECO` e por fim `INSTITUICAO`, como mostrado no exemplo abaixo.

```
<AUTOR>
  <NOME_AUTOR>João</NOME_AUTOR>
  <E-MAIL>joao@instituicaoC.br</E-MAIL>
  <ENDERECO>
    <RUA>Rua C</RUA>
    <NUMERO>1500</NUMERO>
    <CIDADE>Porto Alegre</CIDADE>
    <ESTADO>RS</ESTADO>
    <CEP>90300-001</CEP>
  </ENDERECO>
  <INSTITUICAO>
    <NOME_INST>InstituicaoC</NOME_INST>
    <ENDERECO>
      <RUA>Rua B</RUA>
      <NUMERO>200</NUMERO>
      <CIDADE>Porto Algre</CIDADE>
      <ESTADO>RS</ESTADO>
      <CEP>91000-010</CEP>
    </ENDERECO>
  </INSTITUICAO>
</AUTOR>
```

O conector de escolha diz que um elemento pode ter apenas um dos elementos filho declarados na lista. Por exemplo, na declaração `<!ELEMENT PUBLICACAO (ARTIGO | LIVRO)>` é estabelecido que o elemento `PUBLICACAO` pode conter o elemento `ARTIGO` ou o elemento `LIVRO`.

2.1.2 Controle de Cardinalidade

Na seção anterior, foram utilizados os caracteres "*" e "+" nas declarações. Esses caracteres, juntamente com "?", determinam o número de vezes que um sub-elemento pode aparecer dentro de um elemento. Eles devem ser colocados logo após o nome do elemento que está sendo restringido. Um elemento que não possui nenhum desses caracteres é considerado obrigatório.

O caracter "?" indica que o elemento é opcional. A declaração

```
<!ELEMENT AUTOR (NOME_AUTOR, E-MAIL, ENDERECO?, INSTITUICAO?)>
```

indica que os elementos ENDERECO e INSTITUICAO são opcionais. Já o elemento AUTOR na declaração

```
<!ELEMENT PUBLICACAO (TITULO, ANO_PUBLICACAO, AUTOR+)>
```

pode aparecer várias vezes seguidas. Isso é determinado pelo caracter "+".

O documento abaixo segue as restrições determinadas pela declaração mostrada acima.

```
<PUBLICACAO>
  <TITULO> Título Publicação </TITULO>
  <ANO_PUBLICACAO> 1999 </ANO_PUBLICACAO>
  <AUTOR> João </AUTOR>
  <AUTOR> Maria </AUTOR>
  <AUTOR> José </AUTOR>
</PUBLICACAO>
```

É importante ressaltar que as ocorrências do elemento AUTOR devem ser consecutivas, pois o conector de sequência ","exige isso.

Para expressar que um elemento é opcional e que, quando aparece, pode aparecer várias vezes, utiliza-se o caracter "*".

2.2 Declarações de Atributos

Atributos são declarados separadamente dos elementos, em uma lista de declaração de atributos, como mostrado abaixo.

```
<!ATTLIST PUBLICACAO
      VERSAO CDATA #REQUIRED
      NUMERO_PAGINAS CDATA #IMPLIED>
```

Nesse exemplo, PUBLICACAO indica a que elemento os atributos pertencem. Depois aparece nome, tipo e obrigatoriedade do atributo. Por exemplo, o atributo VERSAO é do tipo CDATA e é obrigatório.

Atributos podem ser de tipos variados. Os tipos possíveis são:

- CDATA: texto simples.
- NMTOKEN: uma palavra ou *token* que possui as mesmas restrições de formação de um nome de elemento, exceto que o primeiro caracter pode ser um número (ao contrário do nome de um elemento).

- NMTOKENS: vários NMTOKENS separados por espaço.
- ENTITY: indica que o valor do atributo é uma referência a uma entidade.
- ENTITIES: indica que o valor do atributo é uma referência a várias entidades.
- ID: indica que o atributo deve ter um valor único.
- IDREF: uma referência a um atributo ID. O valor do atributo deve ser igual ao valor de um outro atributo do tipo ID. Se não existir nenhum atributo com aquele valor, um erro será sinalizado pelo *parser* na validação do documento XML.
- IDREFS: uma referência a vários atributos ID. Os valores devem ser separados por espaço.
- NOTATION: indica que o valor do atributo deve ser uma das notações definidas anteriormente. Uma notação permite que um elemento contenha dados não XML, desde que esses dados estejam de acordo com um formato identificado, declarado através de <!NOTATION ...>. Pode-se também especificar uma aplicação para processar esses dados, como mostra o exemplo abaixo.

```
<!NOTATION gif SYSTEM "C:\APPS\SHOW_GIF.EXE">
<!NOTATION jpeg SYSTEM "C:\APPS\SHOW_JPEG.EXE">
<!ELEMENT figura EMPTY>
<!ATTLIST figura
      tipo NOTATION (jpeg | gif) #REQUIRED
      nome CDATA #REQUIRED>
```

- Grupo: um grupo limita o valor de uma atributo a um conjunto de valores. Os valores possíveis são enumerados um após o outro, separados por "|".

```
<!ATTLIST paragrafo
          formatacao (esquerda | direita | centralizada) >
```

Além disso, um atributo pode ser obrigatório (#REQUIRED) ou opcional (#IMPLIED). Pode ainda ter um valor *default*, o qual deve aparecer entre aspas depois do tipo do atributo, como no exemplo abaixo:

```
<!ATTLIST paragrafo
          formatacao (esquerda | direita | centralizada) "esquerda">
```

Se a declaração do valor *default* aparecer precedida de #FIXED, então esse é o único valor permitido para o atributo. Por exemplo:

```
<!ATTLIST endereco pais CDATA #FIXED "Brasil">
```

Entidade Pré-definida	Nome do Caracter	Caracter representado
&	e-comercial	&
'	apóstrofe	'
>	maior que	>
<	menor que	<
&quote;	aspas	"

Tabela 2.1: Exemplos de Entidades pré-definidas

2.3 Entidades

Uma entidade é utilizada para reaproveitamento de declarações. O uso de entidades pode diminuir o montante de trabalho necessário para o projeto de uma DTD e diminuir os erros provenientes desse trabalho.

Existem quatro tipos de entidades: entidades pré-definidas, entidades gerais, entidades parâmetro e entidades externas [McG 98].

2.3.1 Entidades Pré-Definidas

XML fornece um conjunto padrão de entidades que não precisam ser definidas em uma DTD. Essas entidades são utilizadas para contornar o problema de incluir um caracter especial em um documento XML, como por exemplo os caracteres <, >, &, entre outros [SIM 99]. Esses caracteres não podem ser inseridos diretamente no documento XML porque são interpretados pelo *parser* como caracteres de marcação. Algumas entidades pré-definidas são mostradas na tabela 2.1.

2.3.2 Entidades Gerais

Entidades gerais são usadas para substituir porções de texto, principalmente quando o texto a ser substituído possui caracteres de marcação, como mostra o exemplo abaixo.

```
<!ENTITY COMENTARIO "<P>Esse documento descreve dados sobre publicações
        do autor &AUTORDOC;</P>">
<!ENTITY AUTORDOC "Vanessa">
<!ELEMENT DOCUMENTO (P)+>
<!ELEMENT P (#PCDATA)>
```

As entidades `COMENTARIO` e `AUTORDOC` definidas acima podem ser utilizadas no documento XML ou na DTD através de referências a seus nomes, as quais devem iniciar com `"&"` e terminar com `";"` [McG 98].

```
&AUTORDOC;
&COMENTARIO;
```

2.3.3 Entidades Parâmetro

Entidades parâmetro podem ser vistas como macros de texto que podem ser utilizadas internamente em uma DTD [SAH 00]. Desse modo, um grupo comum de

elementos pode ser armazenado em uma entidade para que não seja necessário declará-lo várias vezes.

```
<!ENTITY % PUBLICACAO "(TITULO, ANO_PUBLICACAO, AUTOR+)">
```

O conteúdo da entidade deve sempre aparecer entre aspas. Antes do nome da entidade deve aparecer o símbolo "%". Feito isso, esta declaração pode ser reaproveitada nas declarações de outros elementos, como mostrado a seguir:

```
<!ENTITY % PUBLICACAO "(TITULO, ANO_PUBLICACAO, AUTOR+)">
<!ENTITY % ARTIGO "(%PUBLICACAO;,LOCAL_PUBLICACAO)">
<!ENTITY % LIVRO "(%PUBLICACAO;,EDITORA)">
<!ELEMENT PUBLICACAO (%ARTIGO; | %LIVRO;)>
```

A utilização da entidade parâmetro deve seguir a seguinte regra: "%" antes do nome da entidade e ";" depois. A declaração mostrada acima equivale a:

```
<!ELEMENT PUBLICACAO
          ((TITULO, ANO_PUBLICACAO, AUTOR+, LOCAL_PUBLICACAO) |
           (TITULO, ANO_PUBLICACAO, AUTOR+, EDITORA))
```

2.3.4 Entidades Externas

Entidades externas fazem referência a entidades que foram definidas em outros locais. Uma entidade externa possui um dos dois identificadores PUBLIC ou SYSTEM, dependendo do tipo de entidade que está sendo referenciada.

Entidades referenciadas com SYSTEM indicam que a entidade está armazenada em um arquivo que pode ser acessado a partir do sistema de arquivos local. Já uma entidade PUBLIC indica uma entidade pública. Mais detalhes sobre entidades externas podem ser encontrados em [BRA 00].

3 XML Schema

XML Schema é um Candidato a Recomendação da W3C para dar estrutura a documentos XML[FAL 00]. A especificação de XML Schema assume que pelo menos dois documentos XML são utilizados: um documento **instância** e pelo menos um documento **esquema**. O documento instância contém a informação propriamente dita, e o documento esquema descreve a estrutura e tipo do documento instância. A distinção entre instância e esquema é semelhante à distinção entre objeto e classe em linguagens de programação orientadas a objeto. Uma classe descreve um objeto tal como um esquema descreve um documento instância [BOX 00].

Como XML Schema descreve um documento XML instância, e esse deve sempre ser uma hierarquia, um diagrama de classes UML pode originar vários XML Schema correspondentes. Cada forma diferente de aninhamento entre as classes dá origem a um novo esquema XML. Um XML Schema, dentre os vários que podem ser utilizados para representar o diagrama de classes da figura 1.1, é mostrado abaixo. Ele será referenciado nesse trabalho como `publicacao.xsd`.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.inf.ufrgs.br/~vanessa/Publicacao"
        elementFormDefault="qualified"
        xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">

<element name="PUBLICACOES" type="pub:tPublicacoes"/>
<element name="ANO_PUBLICACAO" type="integer"/>

<complexType name="tPublicacoes">
  <sequence>
    <element name="PUBLICACAO" type="pub:tPublicacao"
             minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="tPublicacao">
  <sequence>
    <element name="TITULO" type="string"/>
    <element ref="pub:ANO_PUBLICACAO"/>
    <element name="AUTOR" type="pub:tAutor"
             minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="VERSAO" type="string"/>
</complexType>

<complexType name="tArtigo">
  <complexContent>
    <extension base="pub:tPublicacao">
      <sequence>
        <element name="LOCAL_PUBLICACAO" type="string"/>
      </sequence>
```

```
    </extension>
  </complexContent>
</complexType>

<complexType name="tLivro">
  <complexContent>
    <extension base="pub:tPublicacao">
      <sequence>
        <element name="EDITORA" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="tAutor">
  <sequence>
    <element name="NOME_AUTOR" type="string"/>
    <element name="E-MAIL" type="string" />
    <element name="ENDERECO" type="pub:tEndereco"
             minOccurs="0" maxOccurs="1"/>
    <element name="INSTITUICAO" type="pub:tInstituicao"
             minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>

<complexType name="tInstituicao">
  <sequence>
    <element name="NOME_INST" type="string"/>
    <element name="ENDERECO" type="pub:tEndereco"
             minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>

<complexType name="tEndereco">
  <sequence>
    <element name="RUA" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="NUMERO" type="integer"
             minOccurs="0" maxOccurs="1"/>
    <element name="CIDADE" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="ESTADO" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="CEP" type="pub:tCep"
             minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>

<simpleType name="tCep">
  <restriction base="string">
    <pattern value="\d{5}-\d{3}"/>
  </restriction>
</simpleType>
</schema>
```

3.1 Conceitos Básicos

Um XML Schema sempre inicia com `<schema>` e termina com `</schema>`. Todas as declarações de elementos, atributos e tipos devem ser colocadas entre essas duas *tags*. Isto acontece porque um XML Schema deve ser um documento XML válido, e para isso deve possuir apenas um elemento raiz, que nesse caso é `<schema>`. Além disso, o elemento `schema` pode conter definições de *namespaces* e outras opções *default* [VLI 00].

Em XML Schema, o modelo de conteúdo de um elemento pode ser especificado a partir da declaração de um tipo. Um tipo pode ser **Simples** ou **Complexo**. Um tipo simples pode ser atribuído a um atributo ou a um elemento simples, que possui somente texto e não possui elementos filhos. Já um tipo Complexo é utilizado para dizer quais são os subelementos permitidos para um determinado elemento. Tipos simples são declarados com `simpleType`, e tipos complexos com `complexType`. Um `simpleType` pode ser um dos tipos básicos definidos em XML Schema, tais como *string*, `data`, `float`, `double`, `timeDurations`, entre outros. A tabela 3.1 mostra uma relação dos tipos simples disponibilizados por XML Schema. Os exemplos estão separados por vírgula [FAL 00].

Um tipo simples também pode ser definido como uma lista ou como uma união de tipos. Uma lista é definida com `list`, como mostrado abaixo. Um tipo `list` é tratado como uma lista de *tokens* separados por um espaço em branco [JEL 00].

```
<simpleType name="tListaInteiro">
   <list itemType="integer"/>
</simpleType>
```

Na instância XML, um elemento do tipo `tListaInteiro` pode apresentar vários valores do tipo inteiro, como mostrado abaixo.

```
<listaInt>20003 15037 95977 95945</listaInt>
```

O tipo `union` faz a união de vários tipos simples.

```
<simpleType name="tUniao">
  <union memberTypes="pub:tCep pub:tListaInteiro"/>
</simpleType>
```

Quando um tipo simples é definido utilizando `union`, o valor léxico do elemento na instância XML determina qual tipo base foi utilizado [JEL 00]. No exemplo acima, o tipo `tUniao` foi definido como uma união dos tipos `tCep` e `tListaInteiro`, portanto, um elemento do tipo `tUniao` pode ser um Cep ou uma lista de inteiros. Os elementos abaixo são válidos de acordo com a definição do tipo `tUniao`.

```
<ceps>93320-005</ceps>
<ceps>95630 95977 95945</ceps>
<ceps>90000-240</ceps>
```

simpleType	**Exemplo**
string	tabela
byte	-1, 126
unsignedByte	0, 126
binary	62696E617279
integer	-126789, -1, 0, 1, 126789
positiveInteger	1, 126789
negativeInteger	-126789, -1
nonNegativeInteger	0, 1, 126789
nonPositiveInteger	-126789, -1, 0
int	-1, 126789675
unsignedInt	0, 126789675
long	-1, 12678967543233
unsignedLong	0, 12678967543233
short	-1, 12678
unsignedShort	0, 12678
decimal	-1.23, 0, 123.4, 1000.00
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
boolean	true, false
time	13:20:00.000, 13:20:00.000-05:00
timeInstant	1999-05-31T13:20:00.000-05:00
timePeriod	1999-05-31T13:20
timeDuration	P1Y2M3DT10H30M12.3S
date	1999-05-31
month	1999-05
year	1999
century	19
recurringDay	---31
recurringDate	-05-31
recurringDuration	-05-31T13:20:00
Name	AUTOR
QName	pub:tEndereco
NCName	tEndereco
uriReference	http://www.xyz.com
language	en-GB, en-US, fr
ID	
IDREF	
IDREFS	
ENTITY	
ENTITIES	
NOTATION	
NMTOKEN	EUA, Brasil
NMTOKENS	EUA Inglaterra, Brasil Canadá México

Tabela 3.1: Tipos Simples de XML Schema

Um `complexType` define restrições para o modelo de conteúdo de um determinado elemento, o que é feito através dos atributos para especificação de cardinalidade `minOccurs` e `maxOccurs`, e dos delimitadores de grupos de elementos `sequence`, `all` e `choice`.

A declaração `element` liga um tipo a um nome de elemento. Elementos podem ser declarados dentro de um tipo complexo ou abaixo de `schema`. Nesse último caso, são considerados elementos **globais**.

O atributo `minOccurs` especifica o número mínimo de vezes que um subelemento pode aparecer, e `maxOccurs`, o número máximo. Os valores desses atributos devem ser inteiros positivos, mas o atributo `maxOccurs` também permite o valor `unbounded` [PIM 00].

Na esquema `publicacao.xsd`, a declaração

```
<element name="AUTOR" type="pub:tAutor"
         minOccurs="1" maxOccurs="unbounded"/>
```

especifica que o elemento `AUTOR` pode aparecer, no mínimo, uma vez, e que não há um limite máximo. A declaração dos atributos `minOccurs` e `maxOccurs` não é obrigatória. Quando omitidos, são assumidos os valores 1 (um) para `minOccurs` e 1 (um) para `maxOccurs` respectivamente. Isso faz com que o elemento em questão seja obrigatório.

Pode-se especificar o conteúdo de um elemento através de um elemento `sequence`, `choice` ou `all`. O grupo `sequence` indica que os subelementos devem aparecer na instância XML na mesma ordem em que foram declarados no esquema. O grupo `choice` diz que somente um dos elementos declarados no grupo pode aparecer na instância, e o grupo `all` estabelece que todos os elementos do grupo podem aparecer uma ou nenhuma vez, e que eles podem aparecer em qualquer ordem.

Além desses tipos de conteúdo para um elemento, existem ainda mais dois. Um elemento pode ser vazio, ou seja, não ter conteúdo algum. Para isso, basta declarar um `complexType` sem nenhum elemento, e declarar um `element` daquele tipo, como mostrado abaixo.

```
<complexType name="TFigura">
</complexType>

<element name="FIGURA" type="pub:tFigura"/>
```

Um elemento pode também ter um conteúdo misto, que mistura texto e subelementos. Isso pode ser expressado através do atributo `mixed`.

É importante ressaltar que o modelo *mixed* de XML Schema é diferente do modelo *mixed* de XML 1.0. No modelo *mixed* de XML Schema, a ordem e o número de elementos filhos que aparecem na instância deve ser a mesma declarada no esquema. Em XML 1.0, um elemento declarado como *mixed* pode ter texto e subelementos aparecendo em qualquer ordem, e sem restrições de cardinalidade [FAL 00]. Portanto, o exemplo mostrado abaixo exige que o elemento `REFERENCIA` apareça sempre após o elemento `CITACAO`, e que cada um apareça somente uma vez em cada `PARAGRAFO`.

```
<element name="PARAGRAFO" type="pub:tParagrafo"/>

<complexType name="tParagrafo" mixed="true">
  <sequence>
    <element name="CITACAO" type="string"/>
    <element name="REFERENCIA" type="string/>
  <sequence>
</complexType>
```

Em XML Schema, atributos são declarados com `attribute`. Um atributo pode ser declarado como um tipo `simpleType` apenas, `complexType`s não são permitidos para atributos. Além disso, pode-se especificar se um atributo é obrigatório ou opcional. Isto é feito através do atributo `use`. Se o atributo é obrigatório, então declara-se `use=required`. Um atributo também pode ser opcional (`use=optional`) ou fixo (`use=fixed`). No caso de ser fixo, deve-se dizer o valor *default* do atributo utilizando `value`.

A declaração

```
<attribute name="VERSAO" type="string" use=required/>
```

indica que o atributo `VERSAO` é do tipo `string` e é obrigatório. Atributos podem ser declarados abaixo de `schema`, dentro de um tipo complexo ou dentro de um grupo de atributos.

Um grupo de atributos é declarado com `attributeGroup`. Ele define atributos comuns que podem ser utilizados em vários tipos complexos através de referência. Maiores detalhes sobre referências serão fornecidos na seção 3.3.

```
<attributeGroup name="GrupoAtributos">
  <attribute name="VERSAO" type="string"/>
  <attribute name="DATA" type="date"/>
</attributeGroup>
```

3.2 Derivação de Tipos

XML Schema possui um mecanismo de derivação de tipos, ou seja, permite a criação de novos tipos a partir de outros já existentes. Isso pode ser feito de duas maneiras: por restrição ou por extensão.

3.2.1 Derivação por Restrição

Tipos simples só podem ser derivados por restrição, o que é feito aplicando **facetas** a um tipo básico, ou através do uso de uma linguagem de expressões regulares [FAL 00]. No esquema `publicacao.xsd`, o tipo `tCep` foi definido como uma restrição do tipo `string`. O elemento `pattern` que aparece nesse exemplo indica que o valor de um elemento do tipo `tCep` deve possuir 5 (cinco) dígitos e um traço ('-') seguido de mais 3 (três dígitos. Essa restrição é feita através de uma expressão regular.

```
<simpleType name="tCep">
  <restriction base="string">
    <pattern value="\d{5}-\d{3}"/>
  </restriction>
</simpleType>
```

Um exemplo do uso de facetas é mostrado abaixo. Nesse caso, o tipo `integer` está sendo restrito para que seu conjunto de valores válidos inclua apenas números maiores que 0 (zero). Isso foi feito através da faceta `minInclusive`.

```
<simpleType name="tNumero">
  <restriction base="integer">
    <minExclusive value="0"/>
  </restriction>
</simpleType>

<complexType name="tEndereco">
  <sequence>
    <element name="RUA" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="NUMERO" type="pub:tNumero"
             minOccurs="0" maxOccurs="1"/>
    <element name="CIDADE" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="ESTADO" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="CEP" type="pub:tCep"
             minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>
```

Nem sempre é necessário dar um nome a um tipo. Se ele for utilizado somente na declaração de um elemento, ele pode ser declarado dentro do elemento em questão. Isto vale não só para tipos simples, mas também para tipos complexos e atributos. O exemplo abaixo é equivalente ao exemplo mostrado acima.

```
<complexType name="tEndereco">
  <sequence>
    <element name="RUA" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="NUMERO" minOccurs="0" maxOccurs="1"/>
      <simpleType name="tNumero">
        <restriction base="integer">
          <minExclusive value="0"/>
        </restriction>
      </simpleType>
    </element>
    <element name="CIDADE" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="ESTADO" type="string"
             minOccurs="0" maxOccurs="1"/>
    <element name="CEP" type="art:tCep"
             minOccurs="0" maxOccurs="1"/>
  </sequence>
</complexType>
```

Todo tipo de dados possui um conjunto de facetas que caracterizam suas propriedades, as quais podem ser restringidas para criar um novo tipo derivado do original. Por exemplo, pode-se limitar o tamanho de uma *string* ou o *encoding* de um tipo binário (por exemplo, se é `hex` ou `base64`) [JEL 00].

A tabela 3.2 mostra as facetas definidas em XML Schema que podem ser utilizadas para restringir um tipo simples. [BIR 00] define quais facetas podem ser aplicadas a quais tipos simples.

Faceta
length
mimLength
maxLength
pattern
enumeration
maxInclusive
maxExclusive
minInclusive
minExclusive
precision
scale
encoding
period
duration

Tabela 3.2: Facetas definidas em XML Schema

Tipos complexos podem ser derivados por restrição e por extensão. A derivação por restrição permite, por exemplo, restringir a cardinalidade de um subelemento, como mostrado abaixo.

```
<complexType name="tAutorRestrito">
  <complexContent>
    <restriction base="tAutor">
      <sequence>
        <element name="NOME_AUTOR" type="string"/>
        <element name="E-MAIL" type="string" />
        <element name="ENDERECO" type="pub:tEndereco"
                 minOccurs="1" maxOccurs="1"/>
        <element name="INSTITUICAO" type="pub:tInstituicao"
                 minOccurs="0" maxOccurs="1"/>
      </sequence>
    </restriction>
  </complexContent>
</complexType>
```

Esse exemplo mostra uma restrição feita ao elemento `ENDERECO` do tipo `tAutor`. De opcional, ele passou a ser obrigatório. Note que para fazer a restrição de um elemento, todos os outros tiveram que ser redeclarados, mesmo os que não sofreram restrições.

3.2.2 Derivação por Extensão

A derivação por extensão permite adicionar características novas a um tipo. Isso é semelhante ao conceito de herança de UML, já que as características antigas não precisam ser redeclaradas, elas são **herdadas**. O esquema `publicacao.xsd` mostra dois exemplos de derivação por extensão: os tipos `tArtigo` e `tLivro` são derivados de `tPublicacao`. Os elementos de `tPublicacao` não foram declarados novamente, apenas os novos elementos foram acrescentados.

A figura 1.2 utiliza o elemento PUBLICACAO de modos diferentes. Primeiro, com elementos do modelo de conteúdo definidos em tArtigo e depois com elementos definidos em tLivro, embora o elemento PUBLICACAO tenha sido declarado como sendo do tipo tPublicacao. XML Schema possui um mecanismo de *casting* semelhante ao de UML, que permite que elementos de subclasses sejam utilizados para substituir elementos da superclasse. Através desse conceito, é permitido utilizar o modelo de conteúdo de ARTIGO ou LIVRO nos locais onde PUBLICACAO é permitido, como mostrado abaixo [BOX 00].

```
<?xml version="1.0" encoding="US-ASCII"?>
<pub:PUBLICACOES
     xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao"
     xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
     xsi:schemaLocation="http://www.inf.ufrgs.br/~vanessa/Publicacao
                         publicacao.xsd">
  <pub:PUBLICACAO xsi:type="pub:tArtigo" VERSAO="1">
    <pub:TITULO>Título Artigo</pub:TITULO>
    <pub:ANO_PUBLICACAO>2000</pub:ANO_PUBLICACAO>
    <pub:AUTOR>
    ...
    </pub:AUTOR>
    <pub:LOCAL_PUBLICACAO>Anais Congresso</pub:LOCAL_PUBLICACAO>
  </pub:PUBLICACAO>
</pub:PUBLICACOES>
```

3.3 Referência

Definições de tipos podem ser reutilizadas através de uma referência a seu nome. No esquema publicacao.xsd, o tipo complexo tEndereco é utilizado como parte do tipo complexo tAutor e tInstituicao.

Além disso, uma declaração de atributo, elemento ou grupo pode ser referenciada, permitindo a reutilização de declarações. Isto é feito através do atributo ref. Ao invés de declarar um elemento com name e type, referencia-se um elemento que já foi declarado anteriormente. O efeito obtido é o mesmo de que se a declaração do elemento referenciado tivesse sido feita novamente. No esquema publicacao.xsd, o elemento ANO_PUBLICACAO é referenciado na declaração do tipo complexo tPublicacao.

```
<element name="ANO_PUBLICACAO" type="integer"/>

<complexType name="tPublicacao">
  <sequence>
    <element name="TITULO" type="string"/>
    <element ref="pub:ANO_PUBLICACAO"/>
    <element name="AUTOR" type="pub:tAutor"
             minOccurs="1" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="VERSAO" type="string"/>
</complexType>
```

Referências são usadas para evitar redeclarações de elementos ou estruturas comuns. Por exemplo, pode-se definir um grupo de atributos comuns a vários

tipos e referenciar esse grupo dentro de vários `complexType`s. A única restrição para o uso das referências é que o elemento referenciado seja global, ou seja, que tenha sido declarado abaixo de `<schema>`, e não dentro de um `complexType`.

3.4 Grupos de Substituição e Tipos Abstratos

XML Schema permite declarar tipos e elementos abstratos. Esses tipos e elementos são ditos abstratos porque não podem ser utilizados em uma instância XML, do mesmo modo que uma classe abstrata em UML não pode possuir instâncias. Um tipo abstrato é declarado através do atributo `abstract`.

```
<complexType name="tPublicacao" abstract="true"/>
```

Um elemento abstrato é declarado da mesma forma.

Para que um tipo abstrato seja utilizado, é necessário declarar um tipo não abstrato derivado dele. No esquema `publicacao.xsd`, os tipos `tArtigo` e `tLivro` são derivados do tipo `tPublicacao`, portanto, podem ser utilizados para substituí-lo (através do uso do atributo `xsi:type`).

Já um elemento abstrato deve ser substituído por um elemento de seu grupo de substituição. Ao elemento a ser substituído dá-se o nome de elemento **exemplar**. Esse elemento deve ser global. Os elementos de um grupo de substituição devem ter o mesmo tipo do elemento exemplar, ou ser de um tipo derivado daquele. Um exemplo de declaração de um grupo de substituição é mostrado abaixo. Nesse caso, `ANO` poderá ser utilizado nos locais onde a utilização de `ANO_PUBLICACAO` seja permitida. É importante ressaltar que grupos de substituição não precisam necessariamente ser utilizados apenas para elementos abstratos. Elementos comuns também podem possuir grupos de substituição.

```
<element name="ANO" type="integer"
substitutionGroup="ANO_PUBLICACAO"/>
```

Elementos e tipos abstratos são usados para definir interfaces, do mesmo modo que as classes abstratas são utilizadas em UML.

3.5 População, Unicidade, Chaves e Referência

XML Schema permite indicar que o valor de um elemento ou atributo deve ser único em um certo escopo [FAL 00]. Para indicar que o valor de um elemento ou atributo particular deve ser único, utiliza-se o elemento `unique`, seleciona-se um conjunto de elementos com `selector` e depois indica-se que um elemento ou atributo deve ser único dentro do escopo selecionado através de `field`. Os elementos `selector` e `field` contêm expressões XPath [CLA 99]. Supondo-se que o tipo complexo `tAutor` tivesse um atributo `COD_AUTOR` e que ele devesse ser único, sua declaração seria como segue.

```
<unique name="UnicidadeCodAutor">
  <selector>PUBLICACOES/PUBLICACAO/AUTOR</selector>
  <field>@COD_AUTOR</field>
</unique>
```

O conceito de chave em XML Schema também assegura que um determinado valor deve ser único. A única diferença é que esse valor pode ser referenciado em um outro lugar, permitindo desse modo que o esquema expresse restrições de integridade. Esses dois conceitos são declarados através dos elementos `key` e `keyref`, respectivamente, e sua sintaxe é semelhante à declaração de unicidade, embora se utilizem os elementos `key` e `keyref` ao invés de `unique`.

```
<key name="ChaveAutor">
  <selector>PUBLICACOES/PUBLICACAO/AUTOR</selector>
  <field>@COD_AUTOR</field>
</key>
```

Em um diagrama de classes UML, estas restrições não possuem conceitos equivalentes, já que ele não representa instâncias, e sim classes. Só faz sentido falar em unicidade, chaves ou referência quando se trata da população de uma determinada classe. O padrão UML define uma linguagem para expressar tais restrições, a OCL (*Object Constraint Language*) [OMG 00]. No entanto, este trabalho considera apenas o diagrama de classes de UML.

3.6 Valores Nulos

XML permite que elementos sejam omitidos em um documento. Isto pode ser feito para indicar que o valor é desconhecido ou não se aplica àquela situação. Apesar disso, em alguns casos é necessário indicar explicitamente que o valor de um elemento é nulo (*null*). XML Schema possui um mecanismo que permite expressar valores nulos.

Na verdade, o valor *null* não aparece explicitamente no documento instância. Ao invés disso, utiliza-se um atributo `null` para fazer essa indicação. No esquema XML, o elemento deve ser declarado com `nullable="true"`.

```
<element name="DATA" type="date" nullable="true"/>
```

No documento instância XML, o elemento `DATA` pode ou não apresentar um valor. Caso seja nulo, ele apresentará um atributo `null`.

```
<DATA null="true"></DATA>
```

3.7 Namespaces

Um esquema pode ser visto como um conjunto de declarações de tipos e elementos cujos nomes pertencem a um *namespace*. O *namespace* deve ser um nome único, e por isso é comum utilizar o formato de uma URL para definí-lo.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.inf.ufrgs.br/~vanessa/Publicacao"
        elementFormDefault="qualified"
        xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">
```

```
<pub:PUBLICACOES
   xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao"
   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
   xsi:schemaLocation="http://www.inf.ufrgs.br/~vanessa/Publicacao
        http://www.inf.ufrgs.br/~vanessa/publicacao.xsd">
<!-- elementos do documento -->
</pub:PUBLICACOES>
```

Figura 3.1: Instância XML que utiliza `schemaLocation` para indicar a localização do esquema a ser utilizado

A declaração acima associa um prefixo `pub` ao *namespace* `http://www.inf.ufrgs.br/~vanessa/Publicacao`. Esse prefixo aparece no esquema `publicacao.xsd` quando um tipo definido nele é utilizado para declarar um elemento ou atributo.

É necessário declarar o namespace onde se encontram as definições de XML Schema. Isso é feito através da declaração

```
xmlns="http://www.w3.org/2000/10/XMLSchema"
```

O atributo `targetNamespace` indica o *namespace* que está sendo construído ou redefinido.

O uso de namespaces aumenta a flexibilidade de XML Schema, permitindo a reutilização de definições feitas em outros esquemas através de `import` e `include`. Maiores detalhes sobre o uso de namespaces em XML Schema podem ser encontrados em [BRA 99].

3.7.1 Schema Location

O atributo `schemaLocation` é utilizado em três circunstâncias:

1. No documento instância, o atributo `schemaLocation` é utilizado para indicar a localização do esquema. Como um exemplo, a figura 3.1 faz a declaração de um documento instância que segue o esquema `publicacao.xsd`.

2. Em um esquema, o elemento `include` possui um atributo `schemaLocation` obrigatório e contém uma referência URI que deve identificar um documento esquema. Nesse caso as declarações dos dois esquemas (o que está sendo definido e o que está sendo incluído) são mescladas. Mais detalhes sobre `include` serão dados posteriormente.

3. Também em um esquema, o elemento `import` possui dois atributos opcionais `namespace` e `schemaLocation`. Se presente, o atributo `schemaLocation` indica a localização do arquivo esquema.

Um esquema não precisa obrigatoriamente declarar um *namespace*. Para isso existe o atributo `noNamespaceSchemaLocation`, o qual é utilizado para indicar a localização do documento esquema [FAL 00].

3.7.2 Importando Tipos

Um esquema pode utilizar tipos que foram definidos em outros esquemas, mesmo que eles estejam em *namespaces* diferentes.

Importando Tipos de Namespaces Comuns

Se dois esquemas foram definidos com o mesmo namespace, então as definições feitas em um podem ser utilizadas em outro através da declaração `include`.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.inf.ufrgs.br/~vanessa/Publicacao"
        elementFormDefault="qualified"
        xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">

<include schemaLocation=
         "http://www.inf.ufrgs.br/~vanessa/catalogo.xsd"/>
```

A declaração acima inclui no *namespace* `http://www.inf.ufrgs.br/~vanessa/Publicacao` as declarações contidas no esquema `catalogo.xsd`. Os tipos declarados no esquema `catalogo.xsd` são utilizados normalmente no esquema `publicacao.xsd`, como se tivessem sido definidos localmente.

Importando Tipos de Namespaces Distintos

Quando o esquema a ser importado está em um *namespace* diferente, utiliza-se `import`. O elemento `import` indica de qual *namespace* as definições estão sendo importadas.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.inf.ufrgs.br/~vanessa/Publicacao"
        elementFormDefault="qualified"
        xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">

<import namespace="http://Congressos.com.br"
        schemaLocation="http://Congressos.com.br"
         "http://http://Congressos.com.br/Congresso.xsd"/>
```

3.7.3 Redefinindo Tipos

XML Schema fornece um mecanismo de redefinição de tipos pertencentes a outros esquemas. Porém, esse mecanismo possui uma restrição: apenas tipos pertencentes ao mesmo *namespace* podem ser redefinidos.

O exemplo abaixo mostra a redefinição do tipo `tCatalogo`, definido no esquema `catalogo.xsd`.

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://www.inf.ufrgs.br/~vanessa/Publicacao"
        elementFormDefault="qualified"
        xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">

<redefine schemaLocation=
         "http://www.inf.ufrgs.br/~vanessa/catalogo.xsd"/>
```

```
  <!-- Redefinição do tipo tCatalogo -->
  <complexType name="tCatalogo">
    <complexContent>
      <extension base="tCatalogo">
        <sequence>
          <element name="preco" type="float"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

</redefine>
```

No exemplo acima, o tipo `tCatalogo` foi redefinido por extensão. Como a base da extensão foi o próprio tipo `tCatalogo`, o efeito produzido foi a inclusão do elemento `preco` ao tipo `tCatalogo`.

O elemento `redefine` atua do mesmo modo que `include`, já que inclui todos os tipos definidos no esquema `catalogo.xsd` no esquema `publicacao.xsd`. A única diferença é que `redefine` permite que os tipos definidos em `catalogo.xsd` sejam alterados.

4 RDF

O propósito de RDF Schema é um pouco diferente do de XML Schema. RDF foi concebido para descrever metadados sobre **recursos**, os quais podem ser, por exemplo, documentos XML. A especificação de RDF é dividida em duas partes principais. A primeira parte, RDF [LAS 99], define como descrever recursos em termos de suas propriedades e valores; a segunda parte, RDF Schema [BRI 00], define propriedades específicas que podem ser utilizadas para definir esquemas. Essas duas definições juntas costumam ser referidas como RDF(S) [STA 00b].

RDF provê interoperabilidade entre aplicações que trocam informações na WEB [LAS 99], e pode ser utilizado em aplicações como Localização de Recursos *(Search Engines)*, Catalogação *(digital libraries)*, Agentes Inteligentes (troca de conhecimento), para expressar Políticas de Privacidade de um *Web Site*, entre outras.

Os principais objetivos de RDF são

- Tornar possível especificar semântica para dados baseados em XML de uma maneira padronizável e interoperável
- Definir um mecanismo para descrever recursos que não faça nenhum préjulgamento sobre um domínio de aplicação particular, nem defina a semântica de nenhum domínio de aplicação *a priori*.
- Definir um mecanismo que seja independente de domínio, mas que permita descrever informações sobre qualquer domínio.

4.1 Modelo Básico

RDF é um modelo para representar propriedades e valores de propriedades. Propriedades em RDF podem ser pensadas como atributos de recursos e assim sendo correspondem a pares atributo-valor tradicionais. Propriedades RDF também representam *Relacionamentos* entre recursos. Desse modo, RDF é capaz de representar um diagrama Entidade Relacionamento [LAS 99].

O modelo RDF consiste de três tipos básicos de objetos:

- **Recursos:** Tudo o que é descrito por expressões RDF é considerado um recurso. Um recurso pode ser uma página HTML, uma coleção de páginas HTML, um elemento XML, etc. Um recurso também pode ser um objeto que não é diretamente acessível através da Web, como por exemplo um livro impresso. Os nomes dos recursos são sempre *URIs*.
- **Propriedades:** Uma propriedade é um aspecto específico, característica, atributo, ou relação utilizada para descrever um recurso. Cada propriedade tem seu significado específico, define os valores permitidos, os tipos de recursos que pode descrever e o relacionamento com outras propriedades.

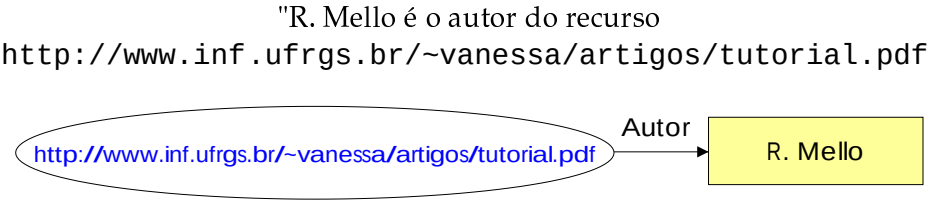


Figura 4.1: Grafo RDF para uma sentença

- **Sentenças:** Um recurso específico junto com uma propriedade e o valor dessa propriedade para aquele recurso é uma *Sentença* RDF. Essas três partes individuais da sentença são chamadas, respectivamente, **sujeito**, **predicado** e **objeto**. [LAS 99] também utiliza o termo **tripla** para definir uma sentença, já que ela é sempre formada por três elementos.

```
Recurso ⇒ Sujeito
Propriedade ⇒ Predicado
Valor da Propriedade ⇒ Objeto
```

Uma sentença RDF pode ser representada por grafos dirigidos rotulados, onde nodos ovais representam recursos, nodos retangulares representam literais e arcos representam propriedades. Arcos são setas que iniciam no sujeito e apontam para o objeto da sentença. A figura 4.1 apresenta um exemplo de grafo RDF para a sentença "R. Mello é o autor do recurso `http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf`".

Essa sentença pode ser estruturada do seguinte modo:

```
http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf ⇒
     Recurso (Sujeito)
autor ⇒ Propriedade (Predicado)
R. Mello ⇒ Valor da propriedade (Objeto)
```

As propriedades podem ter valores simples, também chamados de Literais, ou valores complexos. A figura 4.2 mostra um exemplo de uma propriedade estruturada. Nesse caso, a estrutura da sentença é a seguinte:

```
http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf ⇒ Recurso
autor ⇒ Propriedade Estruturada
Entidade sem nome ⇒ Valor da propriedade autor (Recurso)
Nome ⇒ Propriedade
R. Mello ⇒ Valor da propriedade Name
Email ⇒ Propriedade
ronaldo@inf.ufrgs.br ⇒ Valor da propriedade Email
```

4.2 Sintaxe

Um modelo RDF, além de possuir uma representação gráfica, também pode ser descrito em XML. A sintaxe utilizada para fazer essa descrição será discutida nessa seção.

"O indivíduo cujo nome é Ronaldo Mello, email <ronaldo@inf.ufrgs.br> é o autor de `http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf`"

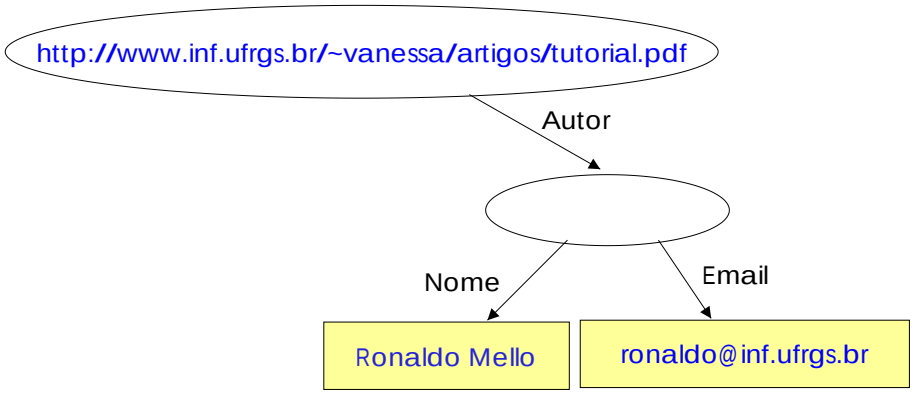


Figura 4.2: Grafo RDF com propriedade estruturada (complexa)

O elemento `RDF` é utilizado para marcar os limites de uma instância de um Modelo RDF em um documento XML. Ele é opcional se o conteúdo puder ser identificado através do contexto da aplicação.

4.2.1 Description, about e ID

A sintaxe de RDF foi projetada para permitir que várias sentenças sobre um mesmo recurso sejam agrupadas em um elemento `Description`. Esse elemento contém a identificação do recurso a ser descrito e uma lista de propriedades que se aplicam a esse recurso. A identificação do recurso a ser descrito pode ser feita através do atributo `about` ou através do atributo `ID`. Um elemento `Description` pode apresentar um atributo `about` ou um atributo `ID`, mas nunca ambos ao mesmo tempo [CHA 00]. A diferença entre eles é que o valor de `about` é interpretado como uma referência URI: se as sentenças descritas em `Description` referem-se a um recurso que ainda não possui uma URI, então é fornecido um identificador através do atributo `ID`. Desse modo, o atributo `ID` sinaliza a criação de um novo recurso e o atributo `about` refere-se a um recurso já existente.

As propriedades de um recurso são especificadas dentro do elemento `Description`. Para cada nova propriedade é adicionado um elemento abaixo de `Description`. O nome do elemento é o nome da propriedade, sendo que podem haver propriedades com o mesmo nome. Caso o valor da propriedade sendo descrita seja um literal, o valor é informado entre as *tags* do elemento que representa a propriedade. Caso contrário, o atributo `resource` é utilizado para especificar que algum outro recurso é o valor dessa propriedade.

A seguir é mostrada uma representação em XML para a sentença da figura 4.1.

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">
  <Description
       about="http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf">
    <pub:Autor>R. Mello</pub:Autor>
```

```
  </Description>
</RDF>
```

4.2.2 Containers

RDF também define três tipos de objeto Container: `Bag`, `Sequence` e `Alternative`. A tabela 4.1 mostra as propriedades de cada um desses tipos de objeto.

	Ordenação	Duplicação	Propriedade Multivalorada
`Bag`	Não	Sim	Sim
`Sequence`	Sim	Sim	Sim
`Alternative`	Não	Não	Não

Tabela 4.1: Tipos de Objeto Container de RDF

Para representar uma coleção de recursos, RDF utiliza um recurso adicional que identifica uma coleção específica. Esse recurso deve ser declarado como sendo uma instância de um dos tipos de objeto container definidos anteriormente (`Bag`, `Sequence` ou `Alternative`). A Figura 4.3 mostra um exemplo de definição de uma propriedade container. Sua representação em XML é mostrada abaixo.

Os autores de `http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf` são Mello, Dorneles, Kade, Braganholo e Heuser.

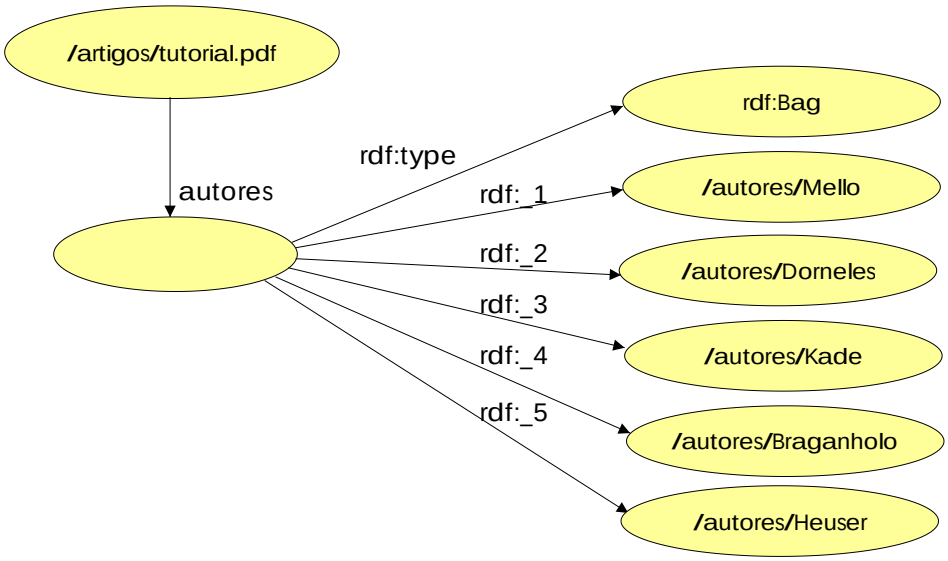


Figura 4.3: Exemplo de container

```
<rdf:RDF>
  <rdf:Description
       about="http://www.inf.ufrgs.br/~vanessa/artigos/tutorial.pdf">
    <pub:autores>
      <rdf:Bag>
```

```
        <rdf:li resource="http://www.inf.ufrgs.br/autores/Mello"/>
        <rdf:li resource="http://www.inf.ufrgs.br/autores/Dorneles"/>
        <rdf:li resource="http://www.inf.ufrgs.br/autores/Kade"/>
        <rdf:li resource="http://www.inf.ufrgs.br/autores/Braganholo"/>
        <rdf:li resource="http://www.inf.ufrgs.br/autores/Heuser"/>
      </rdf:Bag>
    </pub:autores>
  </rdf:Description>
</rdf:RDF>
```

A Figura 4.4 mostra como fazer referência a um container definido anteriormente, ou seja, como definir propriedades de um container. Esse exemplo expressa que R. Mello é o autor do *Bag* "artigos", mas não diz nada sobre cada artigo individualmente. Caso seja necessário descrever cada ítem do container, utiliza-se o atributo `aboutEach`. A figura 4.5 expressa que cada artigo foi escrito por R. Mello.

```
<rdf:Bag ID="artigos">
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/tut3sbbd.pdf" />
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/a000139.pdf" />
</rdf:Bag>

<rdf:Description about="#artigos">
  <pub:Autor>R. Mello</pub:Autor>
</rdf:Description>
```

Figura 4.4: Referenciando um container

```
<rdf:Bag ID="artigos">
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/tut3sbbd.pdf" />
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/a000139.pdf" />
</rdf:Bag>

<rdf:Description aboutEach="#artigos">
  <pub:Autor>R. Mello</pub:Autor>
</rdf:Description>
```

Figura 4.5: Referenciando cada membro de um container

O uso do atributo `aboutEach` não possui uma representação gráfica específica, ou seja, é possível expressar referência a membros de um container utilizando a sintaxe XML de RDF, mas não é possível fazer essa mesma representação graficamente. Caso seja necessário utilizar a representação gráfica, o uso do atributo `aboutEach` equivale a fazer uma sentença para cada membro do container separadamente.

Outro modo de referenciar cada membro de um container é através do atributo `aboutEachPrefix`. Esse atributo permite que se faça a referência sem ter que listar cada membro do container, desde que o URI de cada membro inicie com um padrão de valor. A figura 4.6 mostra um exemplo do uso do atributo `aboutEachPrefix`.

```
<rdf:Description aboutEachPrefix="http://www.inf.ufrgs.br/~ronaldo">
  <pub:Autor>R. Mello</pub:Autor>
</rdf:Description>
```

é equivalente a

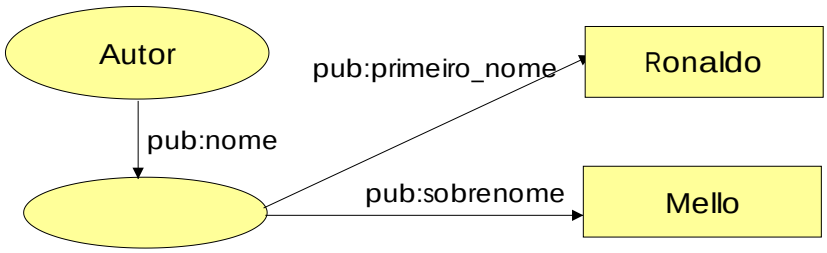
```
<rdf:Bag ID="artigos">
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/tut3sbbd.pdf" />
  <rdf:li resource="http://www.inf.ufrgs.br/~ronaldo/a000139.pdf" />
</rdf:Bag

<rdf:Description aboutEach="#artigos">
  <pub:Autor>R. Mello</pub:Artigo>
</rdf:Description>
```

Figura 4.6: Exemplo da utilização do atributo `aboutEachPrefix`

4.2.3 Relações

Uma sentença especifica uma relação entre dois recursos, portanto, RDF só representa relações binárias. Para representar relações maiores, acrescenta-se um recurso intermediário, como mostrado na figura 4.7.




```
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao">
  <rdf:Description about="Autor">
    <pub:nome rdf:parseType="Resource">
       <pub:primeiro_nome>Ronaldo</pub:primeiro_nome>
       <pub:sobrenome>Mello</pub:sobrenome>
    </pub:nome>
  </rdf:Description>
</RDF>
```

Figura 4.7: Relação ternária

4.3 RDF Schema

RDF define um modelo simples para descrever relacionamentos entre recursos em termos de propriedades e valores. Propriedades RDF podem ser vistas como atributos de recursos. Assim sendo, correspondem a pares atributo valor tradici-

onais. Propriedades RDF também representam relacionamentos entre recursos. Desse modo, o modelo de dados RDF pode representar um diagrama Entidade Relacionamento. Apesar disso, o modelo de dados RDF não fornece um mecanismo para declarar essas propriedades, nem fornece mecanismos para definir os relacionamentos entre essas propriedades e outros recursos. Este é o papel de RDF Schema [BRI 00].

RDF Schema é um sistema de classes extensível e genérico que pode ser utilizado como base para esquemas de um domínio específico. Esses esquemas podem ser compartilhados e estendidos através de refinamento de subclasses. Além disso, definições de metadados podem ser reutilizadas através do compartilhamento de esquemas.

4.3.1 Sistema de Tipos

RDF Schema fornece um Sistema de Tipos (*Type System*) básico para ser utilizado em modelos RDF. Seu vocabulário está definido em um namespace chamado `rdfs`.

```
rdfs=http://www.w3.org/2000/01/rdf-schema#
rdf=http://www.w3.org/1999/02/22-rdf-sysntax-ns#
```

Dando continuidade ao estudo de caso, um esquema para as publicações e autores pode ser definido em RDF(S) como mostrado abaixo. Ele será referenciado ao longo deste texto como `publicacao.rdf`.

```
<?xml version="1.0" ?>
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:pub="http://www.inf.ufrgs.br/~vanessa/Publicacao#">

<!-- Definições das classes -->

 <rdf:Description ID="Publicacao">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf
     rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>

 <rdf:Description ID="Artigo">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#Publicacao"/>
 </rdf:Description>

 <rdf:Description ID="Livro">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#Publicacao"/>
 </rdf:Description>

 <rdf:Description ID="Autor">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf
     rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>
```

```
 <rdf:Description ID="Instituicao">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf
     rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>

 <rdf:Description ID="Endereco">
   <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf
     rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>

<!-- Propriedades da Classe Autor -->

 <rdf:Description ID="nome_autor">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Autor"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>

 <rdf:Description ID="e-mail">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Autor"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>

 <rdf:Description ID="instituicao_autor">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Autor"/>
   <rdfs:range rdf:resource="#Instituicao"/>
 </rdf:Description>

 <rdf:Description ID="endereco_autor">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Autor"/>
   <rdfs:range rdf:resource="#Endereco"/>
 </rdf:Description>

 <rdf:Description ID="publicacoes_autor">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Autor"/>
   <rdfs:range rdf:resource="#Publicacao"/>
 </rdf:Description>

<!-- Propriedades da Classe Publicacao -->

 <rdf:Description ID="titulo">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Artigo"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>
```

```
 <rdf:Description ID="ano_publicacao">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Artigo"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer"/>
 </rdf:Description>

 <rdf:Description ID="autores_publicacao">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Publicacao"/>
   <rdfs:range rdf:resource="#Autor"/>
 </rdf:Description>

<!-- Propriedades da Classe Artigo -->

 <rdf:Description ID="local_publicacao">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Artigo"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>

<!-- Propriedades da Classe Livro -->

 <rdf:Description ID="editora">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Livro"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>

<!-- Propriedades da Classe Instituição -->

 <rdf:Description ID="nome_inst">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Instituicao"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>

 <rdf:Description ID="endereco_instituicao">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Instituicao"/>
   <rdfs:range rdf:resource="#Endereco"/>
 </rdf:Description>

<!-- Propriedades da Classe Endereço -->

 <rdf:Description ID="rua">
   <rdf:type
     resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
   <rdfs:domain rdf:resource="#Endereco"/>
   <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
 </rdf:Description>
```

```
<rdf:Description ID="numero">
  <rdf:type
    resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
  <rdfs:domain rdf:resource="#Endereco"/>
  <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#integer"/>
</rdf:Description>

<rdf:Description ID="cidade">
  <rdf:type
    resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
  <rdfs:domain rdf:resource="#Endereco"/>
  <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
</rdf:Description>

<rdf:Description ID="estado">
  <rdf:type
    resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
  <rdfs:domain rdf:resource="#Endereco"/>
  <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
</rdf:Description>

<rdf:Description ID="cep">
  <rdf:type
    resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Property"/>
  <rdfs:domain rdf:resource="#Endereco"/>
  <rdfs:range rdf:resource="http://www.w3.org/TR/xmlschema-2/#string"/>
</rdf:Description>

</rdf:RDF>
```

Como tudo em RDF é considerado um recurso, RDF Schema estabelece que eles podem ser organizados em classes. Um recurso pode ser instância de uma ou mais classes. A propriedade `rdf:type` é utilizada para indicar as classes das quais um recurso é instância.

As classes podem estar organizadas em uma hierarquia. Por exemplo, uma classe **ArtigoTecnico** pode ser considerada subclasse de **Artigo**, a qual é subclasse de **Publicacao**. Isso significa que qualquer recurso do tipo `rdf:type RecursoTecnico` é também considerado como sendo do tipo `rdf:type Publicacao`. Tal relacionamento entre classes é denotado através da propriedade `subClassOf`.

O sistema de tipos de RDF Schema é semelhante ao sistema de tipos de UML, com algumas pequenas diferenças. Um delas é que ao invés de definir classes em termos das propriedades que suas instâncias devem ter, um RDF Schema define propriedades em termos de classes de recursos aos quais elas se aplicam. Este é o papel de `rdfs:domain` e `rdfs:range`. Por exemplo, pode-se definir que a propriedade `editora` possui um domínio `Livro` e um *range* `string`, enquanto que em UML seria definida uma classe `Livro` com um atributo chamado `editora` do tipo `string`.

A especificação de RDF Schema não define nenhum tipo específico de dados, mas permite que eles sejam usados como valor da propriedade `rdfs:range`.

A figura 4.8 [BRI 00] mostra os conceitos de classe, subclasse e recurso. Uma classe é representada por um retângulo arredondado, um recurso é representado por um ponto grande. As setas são desenhadas de um recurso para a classe que

os define. Uma subclasse é representada por um retângulo arredondado dentro de outro (a superclasse).

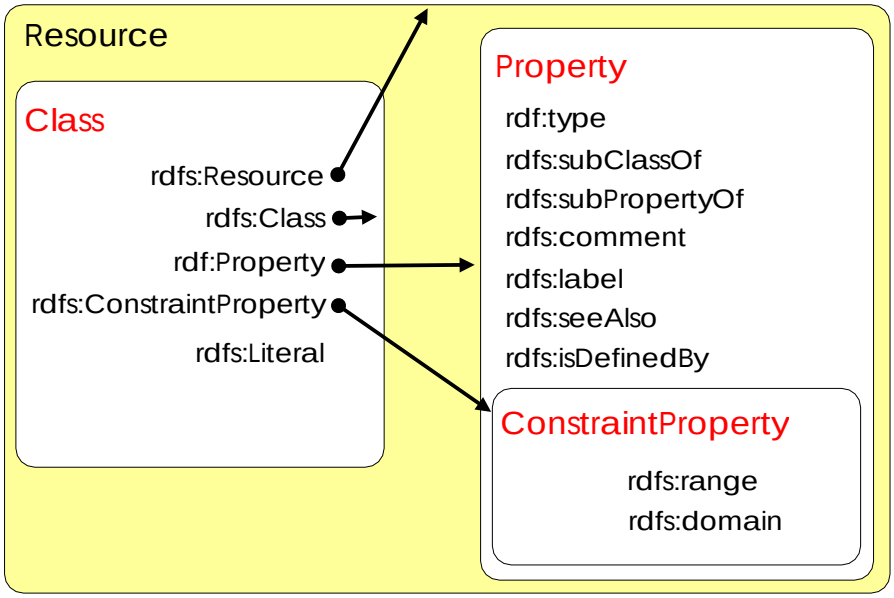


Figura 4.8: Classes e Recursos como Conjuntos e Elementos

Classes

Os seguintes recursos são classes definidas como parte do vocabulário de RDF Schema.

- `rdfs:Resource`

  Todas as coisas que estão sendo descritas por expressões RDF são chamadas **Recursos**, e são consideradas instâncias da classe `rdfs:Resource`.

- `rdf:Property`

  Representa o subconjunto dos recursos RDF que são propriedades.

- `rdfs:Class`

  Corresponde ao conceito genérico de *Tipo* ou *Categoria*, semelhante à noção de Classe em UML.

  Quando um esquema define uma nova classe, o recurso que representa aquela classe deve ter uma propriedade `rdf:type` cujo valor é o recurso `rdfs:Class`.

Propriedades

Todo modelo RDF que utiliza o mecanismo de esquema também inclui as propriedades mostradas a seguir. Elas são instâncias da classe `rdf:Property` e fornecem mecanismos para expressar relacionamentos entre classes e suas instâncias ou superclasses.

- `rdf:type`

  Indica que um recurso é um membro de uma classe, e portanto possui todas as características de um membro de uma classe. Quando um recurso possui uma propriedade `rdf:type` cujo valor é uma classe específica, diz-se que o recurso é uma **instância** da classe especificada.

  Uma instância da classe `Artigo` pode ser representada em RDF(S) como mostrado abaixo [STA 00a].

```
<?xml version="1.0" encoding="US-ASCII"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pub="http://www.inf.ufrgs.br/~vanessa/publicacao.rdf#"
  xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">

 <rdf:Description
   ID="http://www.inf.ufrgs.br/~vanessa/Publicacao/Artigo1.html">
   <rdf:type rdf:resource=
        "www.inf.ufrgs.br/~vanessa/publicacao.rdf#Publicacao"/>
   <rdf:type rdf:resource=
        "www.inf.ufrgs.br/~vanessa/publicacao.rdf#Artigo"/>
   <pub:titulo rdf:resource="Titulo Artigo"/>
   <pub:ano_publicacao rdf:resource="2000"/>
   <pub:local_publicacao rdf:resource="Anais Congresso"/>
   <pub:autor rdf:resource=
        "www.inf.ufrgs.br/~vanessa/Publicacao/Autor1.html"/>
   <pub:autor rdf:resource=
        "www.inf.ufrgs.br/~vanessa/Publicacao/Autor2.html"/>
 </rdf:Description>

</rdf:RDF>
```

  Apesar da instância acima mostrar um artigo com dois autores, essa restrição de cardinalidade para o relacionamento de Publicação com Autor não foi feita no esquema RDF(S) `publicacao.rdf`. Isso deve-se ao fato de RDF(S) não possuir um mecanismo de restrição de cardinalidades como em UML.

- `rdf:subClassOf`

  Essa propriedade especifica uma relação subconjunto/superconjunto entre classes.

  A propriedade `rdfs:subClassOf` é transitiva. Se a classe A é subclasse de B, e B é subclasse de C, então A é também implicitamente subclasse de C. Somente instâncias de `rdfs:Class` podem ter a propriedade `rdfs:subClassOf`. O valor dessa propriedade é sempre do tipo `rdf:type rdfs:Class`. Uma classe pode ser subclasse de mais de uma classe. Porém existe uma restrição que evita grafos de herança cíclicos. Uma classe não pode ser subclasse de si mesma, e nem de uma de suas subclasses. O fato de uma classe poder ser subclasse de várias classes permite que RDF(S) expresse herança múltipla de modo similar a UML.

- `rdf:subPropertyOf`

  A propriedade `rdfs:subPropertyOf` é uma instância de `rdf:Property` e é utilizada para especificar que uma propriedade é especialização de outra. Se uma propriedade P2 é `subPropertyOf` de outra propriedade mais geral P1, e se um recurso A tem uma propriedade P2 com um valor B, isto implica que o recurso A também tem uma propriedade P1 com valor B [BRI 00].

- `rdfs:seeAlso` e `rdf:isDefinedBy`

  Um dado recurso pode estar descrito em mais de um local na Internet. A propriedade `seeAlso` pode ser usada para apontar descrições alternativas de um recurso. Sua sub propriedade `isDefinedBy` aponta para a descrição original ou autoritária do recurso.

- `rdfs:label` e `rdfs:comment`

  São propriedades utilizadas para fins de documentação, ou seja, para descrever os recursos de uma forma mais compreensível.

  `rdfs:label` é usado para dar um nome compreensível para um recurso.

  `rdfs:comment` é usado para dar uma descrição mais detalhada a um recurso.

Restrições

RDF Schema pode descrever limitações nos tipos de valores que são válidos para alguma propriedade, ou nas classes para as quais faz sentido atribuir certas propriedades.

- `rdfs:ConstraintResource` e `rdfs:ConstraintProperty`

  São superclasse de `rdfs:domain` e de `rdfs:range`. São utilizados para informar que existem restrições a respeito de algum recurso.

- `rdfs:range`

  Uma restrição `range` diz que o valor de uma propriedade deve ser um recurso de uma determinada classe. Por exemplo, uma restrição de `range` aplicada à propriedade `autor` expressa que o valor da propriedade `autor` deve ser um recurso da classe `Pessoa`.

- `rdfs:domain` Uma restrição `domain` diz que uma propriedade pode ser utilizada em recursos de uma determinada classe. Por exemplo, a propriedade `autor` só pode se originar de um recurso que é uma instância da classe `Publicação`.

5 Outros Modelos

Além dos modelos citados nesse trabalho, existem várias outras propostas de esquemas para XML. Esse capítulo apresenta duas dessas propostas: DSD e DDML. DSD é uma proposta da AT&T e Brics, e DDML é uma nota da W3C. Ambas serão apresentadas brevemente por não se tratarem de modelos tão populares quanto a DTD, XML Schema e RDF.

5.1 DSD

Um *Document Structure Description* (DSD) é uma especificação de uma classe de documentos XML. Um DSD define uma gramática para documentos XML, atributos *default*, conteúdo de elementos e documentação da classe. Sua sintaxe é descrita em XML [KLA 99].

Em DSD um **documento aplicação** é um documento XML que tem o propósito de estar de acordo com um esquema DSD. O processador DSD identifica se o documento aplicação está ou não de acordo com seu esquema DSD [KLA 00]. Um processador DSD pode ser encontrado em `http//www.brics.dk/DSD/implementation.html`

Um documento aplicação é associado com um DSD através da instrução de processamento

```
<?dsd URI="uri"?>
```

onde **uri** indica a localização do esquema DSD.

Um esquema DSD consiste de um conjunto de definições, cada qual associada com um ID que possibilita reuso e redefinições [KLA 00].

5.1.1 Restrições a elementos

A definição de elemento é o principal tipo de definição em DSD. Ela consiste de um nome de elemento e uma restrição.

```
<ElementDef ID="TITULO" Name="TITULO"
            Defaultable="no">
  <Content>
    <StringType/>
  </Content>
</ElementDef>
```

5.1.2 Declarações de Atributos

Uma declaração de atributo consiste de um nome de atributo e um tipo *string* (vide seção 5.1.3).

```
<AttributeDecl Name="VERSAO">
  <StringType/>
</AttributeDecl>
```

5.1.3 Tipos *string*

Um tipo *string* é um conjunto de strings definido por uma expressão regular. Além das expressões regulares, alguns operadores podem ser utilizados na definição de um tipo *string*, os quais são mostrados na tabela 5.1 [KLA 99]:

Operador
Sequence
ZeroOrMore
OneOrMore
Union
Optional
Intersection
Complement
Repeat
Empty
String
CharSet
CharRange
AnyChar

Tabela 5.1: Operadores aplicáveis à construção de um tipo *string*

Pode-se utilizar os operadores da tabela 5.1 para definir o tipo `tCep` como mostrado abaixo.

```
<StringTypeDef ID="tCep">
  <Sequence>
    <Repeat Value="5">
      <CharRange Start="0" End="9"/>
    </Repeat>
    <String Value="-"/>
    <Repeat Value="3">
      <CharSet Value="0123456789"/>
    </Repeat>
  </Sequence>
</StringTypeDef>
```

5.1.4 Expressões de Conteúdo

O conteúdo de um elemento pode ser visto como uma sequência de elementos filhos ou texto. As expressões de conteúdo de DSD são construídas através de expressões atômicas e operadores de expressão de conteúdo. Uma expressão atômica é uma descrição de elemento ou um tipo *string*. Descrições de elemento são utilizadas para adicionar restrições aos elementos filhos, e tipos *string* especificam restrições a um elemento do tipo texto. [KLA 00].

Operador
Sequence
Optional
ZeroOrMore
OneOrMore
Union
AnyElement
Intersection
Empty
If

Tabela 5.2: Operadores aplicáveis à restrição do conteúdo de um elemento

Os operadores de expressão de conteúdo são mostrados na tabela 5.2.

DSD permite também utilizar expressões booleanas. Mais detalhes podem ser encontrados em [KLA 99].

Um esquema DSD para o documento da figura 1.2 é mostrado abaixo. O atributo `IDRef` do elemento `DSD` define o elemento raiz do documento XML.

```
<?dsd URI="http://www.brics.dk/DSD/dsd.dsd"?>

<DSD IDRef="PUBLICACOES" DSDVersion="1.0">

  <ElementDef ID="PUBLICACOES" Name="PUBLICACOES">
    <OneOrMore>
      <Element IDRef="PUBLICACAO"/>
    </OneOrMore>
  </ElementDef>

  <ElementDef ID="PUBLICACAO" Name="PUBLICACAO">
    <Sequence>
      <Element IDRef="TITULO"/>
      <Element IDRef="ANO_PUBLICACAO"/>
      <OneOrMore>
        <Element IDRef="AUTOR"/>
      </OneOrMore>
      <Optional>
         <Element IDRef="LOCAL_PUBLICACAO"/>
      </Optional>
      <Optional>
         <Element IDRef="EDITORA"/>
      </Optional>
    </Sequence>
    <AttributeDecl Name="VERSAO">
      <StringType/>
    </AttributeDecl>
  </ElementDef>

  <ElementDef ID="TITULO" Name="TITULO">
    <StringType/>
  </ElementDef>

  <ElementDef ID="ANO_PUBLICACAO" Name="ANO_PUBLICACAO">
```

```
  <StringType/>
</ElementDef>

<ElementDef ID="LOCAL_PUBLICACAO" Name="LOCAL_PUBLICACAO">
  <StringType/>
</ElementDef>

<ElementDef ID="EDITORA" Name="EDITORA">
  <StringType/>
</ElementDef>

<ElementDef ID="AUTOR" Name="AUTOR">
  <Sequence>
    <Element IDRef="NOME_AUTOR"/>
    <Element IDRef="E-MAIL"/>
    <Optional>
      <Element IDRef="ENDERECO"/>
    </Optional>
    <Optional>
      <Element IDRef="INSTITUICAO"/>
    </Optional>
  </Sequence>
</ElementDef>

<ElementDef ID="NOME_AUTOR" Name="NOME_AUTOR">
  <StringType/>
</ElementDef>

<ElementDef ID="E-MAIL" Name="E-MAIL">
  <StringType/>
</ElementDef>

<ElementDef ID="INSTITUICAO" Name="INSTITUICAO">
  <Sequence>
    <Element IDRef="NOME_INST"/>
    <Optional>
      <Element IDRef="ENDERECO"/>
    </Optional>
  </Sequence>
</ElementDef>

<ElementDef ID="NOME_INST" Name="NOME_INST">
  <StringType/>
</ElementDef>

<ElementDef ID="ENDERECO" Name="ENDERECO">
  <Sequence>
    <Optional>
      <Element IDRef="RUA"/>
    </Optional>
    <Optional>
      <Element IDRef="NUMERO"/>
    </Optional>
    <Optional>
      <Element IDRef="CIDADE"/>
    </Optional>
    <Optional>
      <Element IDRef="ESTADO"/>
```

```
      </Optional>
      <Optional>
        <Element IDRef="CEP"/>
      </Optional>
    </Sequence>
  </ElementDef>

  <ElementDef ID="RUA" Name="RUA">
    <StringType/>
  </ElementDef>

  <ElementDef ID="NUMERO" Name="NUMERO">
    <StringType IDRef="tNumero"/>
  </ElementDef>

  <StringTypeDef ID="tNumero">
    <OneOrMore>
      <Sequence>
        <CharRange Start="0" End="9"/>
      </Sequence>
    </OneOrMore>
  </StringTypeDef>

  <ElementDef ID="CIDADE" Name="CIDADE">
    <StringType/>
  </ElementDef>

  <ElementDef ID="ESTADO" Name="ESTADO">
    <StringType/>
  </ElementDef>

  <ElementDef ID="CEP" Name="CEP">
    <StringType IDRef="tCep"/>
  </ElementDef>

  <StringTypeDef ID="tCep">
    <Sequence>
      <Repeat Value="5">
        <CharRange Start="0" End="9"/>
      </Repeat>
      <String Value="-"/>
      <Repeat Value="3">
        <CharSet Value="0123456789"/>
      </Repeat>
    </Sequence>
  </StringTypeDef>

</DSD>
```

5.2 DDML

Document Definition Markup Language (DDML) [BOU 99] é um W3C Note, conhecido anteriormente como *XSchema*.

Um documento DDML utiliza sintaxe XML e contém um único elemento `DocumentDef` no qual são aninhadas informações que descrevem o esquema.

5.2.1 Declaração de Elementos

A declaração de um elemento é feita com `ElementDecl`.

```
<ElementDecl Name="PUBLICACAO">
   <!-- Conteúdo do elemento -->
</ElementDecl>
```

O modelo de conteúdo de um elemento deve ser especificado dentro da declaração do elemento. Os modelos de conteúdo permitidos são: `Ref`, `Choice`, `Seq`, `Empty`, `Any`, `PCData` ou `Mixed`. Todos devem estar precedidos de um elemento `Model`.

Um elemento vazio pode ser declarado como mostrado abaixo.

```
<ElementDecl Name="FIGURA">
  <Model>
    <Empty/>
  </Model>
</ElementDecl>
```

A cardinalidade de um elemento pode ser expressada através do atributo `Frequency`. Os valores permitidos são `Required`, `Optional`, `ZeroOrMore` e `OneOrMore`.

```
<ElementDecl Name="ENDERECO">
  <Model>
    <Seq>
       <Ref Element="RUA" Frequency="Optional"/>
       <Ref Element="NUMERO" Frequency="Optional"/>
       <Ref Element="CIDADE" Frequency="Optional"/>
       <Ref Element="ESTADO" Frequency="Optional"/>
       <Ref Element="CEP" Frequency="Optional"/>
    </Seq>
  </Model>
</ElementDecl>
```

Os elementos referenciados através de `Ref` devem ter sido declarados com o mesmo nome utilizado na referência.

5.2.2 Declaração de Atributos

Atributos são declarados através de elementos `AttDef`. O nome, tipo e valor *default* de um atributo são definidos através de atributos do elemento `AttDef` [BOU 99].

```
<ElementDecl Name="PUBLICACAO">
  <!-- Outras declarações -->
  <AttGroup>
    <AttDef Name="VERSAO" Type="CData"/>
  </AttGroup>
</ElementDecl>
```

```
<ElementDecl Name="FIGURA">
  <Model>
    <Empty/>
  </Model>
  <AttGroup>
    <AttDef Name="status" Type="Enumerated">
      <Enumeration>
        <EnumerationValue Value="jpg"/>
        <EnumerationValue Value="gif"/>
      </Enumeration>
    </AttDef>
  </AttGroup>
</ElementDecl>
```

Além de elementos e atributos, DDML permite declarar entidades e notações. Mais detalhes podem ser encontrados em [BOU 99].

Um esquema DDML para o documento XML da figura 1.2 é mostrado abaixo. Esta descrição foi validada através da DTD de DDML fornecida por [BOU 99].

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

<DocumentDef FileExtension="ddml" prefix="">

<ElementDecl Name="PUBLICACOES">
  <Model>
    <Ref Element="PUBLICACAO" Frequency="OneOrMore"/>
  </Model>
</ElementDecl>

<ElementDecl Name="PUBLICACAO">
  <Model>
    <Seq>
      <Ref Element="TITULO"/>
      <Ref Element="ANO_PUBLICACAO"/>
      <Ref Element="AUTOR"/>
      <Choice>
         <Ref Element="LOCAL_PUBLICACAO"/>
         <Ref Element="EDITORA"/>
      </Choice>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="VERSAO" Type="CData"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="TITULO">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="ANO_PUBLICACAO">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>
```

```
<ElementDecl Name="AUTOR">
  <Model>
    <Seq>
      <Ref Element="NOME_AUTOR"/>
      <Ref Element="E-MAIL"/>
      <Ref Element="ENDERECO" Frequency="Optional"/>
      <Ref Element="INSTITUICAO" Frequency="Optional"/>
    </Seq>
  </Model>
</ElementDecl>

<ElementDecl Name="LOCAL_PUBLICACAO">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="EDITORA">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="NOME_AUTOR">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="E-MAIL">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="ENDERECO">
  <Model>
    <Seq>
       <Ref Element="RUA" Frequency="Optional"/>
       <Ref Element="NUMERO" Frequency="Optional"/>
       <Ref Element="CIDADE" Frequency="Optional"/>
       <Ref Element="ESTADO" Frequency="Optional"/>
       <Ref Element="CEP" Frequency="Optional"/>
    </Seq>
  </Model>
</ElementDecl>

<ElementDecl Name="INSTITUICAO">
  <Model>
    <Seq>
       <Ref Element="NOME_INST"/>
       <Ref Element="ENDERECO" Frequency="Optional"/>
    </Seq>
  </Model>
</ElementDecl>

<ElementDecl Name="NOME_INST">
```

```
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="RUA">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="NUMERO">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="CIDADE">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="ESTADO">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

<ElementDecl Name="CEP">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>

</DocumentDef>
```

6 Comparação e Conclusão

Um diagrama de classes em UML descreve os tipos de objetos em um sistema e os vários tipos de relacionamento estático que existem entre elas [FOW 00]. Essa seção apresenta os mapeamentos desses conceitos para DTD, XML Schema e RDF(S). A tabela 6.1 faz um mapeamento dos conceitos de UML para as estruturas disponíveis nas especificações de DTD, XML Schema e RDF Schema, respectivamente.

Talvez o mapeamento mais claro seja o do conceito de classe. Em RDF, existe um conceito semelhante (`Class`) e, em XML Schema existe o conceito de tipo complexo (`complexType`) que pode ser utilizado para representar uma classe e seus atributos como mostrado no esquema `publicacao.xsd`. Em uma DTD, uma classe pode ser representada por um elemento que possui um modelo de conteúdo complexo. Conseqüentemente, os atributos podem ser representados em XML Schema através de elementos de um tipo complexo, em DTD através de elementos e em RDF por propriedades (`Property`) de classes. Os atributos (`attribute`) em XML Schema e DTD (`ATTLIST`) também poderiam ser utilizados para representar atributos de uma classe. Segundo [PIM 00], os atributos dos elementos XML são, em geral, reservados para armazenar dados sobre dados, ou metadados. Portanto, os atributos em XML Schema e DTD podem ser utilizados para representar metadados de uma classe UML. Nesse ponto, existe um diferença entre XML e UML: UML não separa dados de metadados em uma classe.

Os tipos dos atributos das classes podem ser representados em XML Schema com um `simpleType`, exceto quando o atributo representa um relacionamento. Nesse caso seu tipo deve ser um `complexType`. Em RDF, o elemento `range` é utilizado para representar o tipo de uma propriedade. Nesse ponto, a DTD perde em relação aos outros modelos, pois fornece apenas o tipo `#PCDATA` para expressar o tipo de um elemento.

UML fornece três tipos de visibilidade de atributos para uma classe: *public*, *protected* e *private*. Em DTD, RDF(S) e XML Schema, todos os atributos são públicos por definição.

Os relacionamentos em RDF são mapeados de forma muito mais clara que em XML Schema e DTD, já que o propósito do atributo `range` é justamente fazer o relacionamento de duas classes. Na verdade, UML utiliza o termo **Associação**, ao invés de Relacionamento. Em XML Schema, os relacionamentos são representados por aninhamento de tipos complexos. No esquema da figura 1.1, as classes Autor e Instituição possuem um relacionamento 1 para n. No XML Schema correspondente (esquema `publicacao.xsd`), foram definidos dois tipos complexos, um `tAutor` e outro `tInstituicao`. O tipo `tAutor` contém um elemento `INSTITUICAO` do tipo `tInstituicao` que implementa esse relacionamento. O mais lógico seria fazer justamente o contrário, ou seja, definir que o tipo complexo `tInstituicao` possui um elemento `AUTOR` do tipo `tAutor`, o qual pode

aparecer várias vezes (devido à cardinalidade do relacionamento). No entanto, decidiu-se representar o relacionamento da primeira forma, porque o elemento principal do esquema é `PUBLICACAO`, o qual está relacionado com `AUTOR`, que por sua vez relaciona-se com `INSTITUICAO`. Como XML exige uma hierarquia de elementos, a representação deve começar pelo elemento de maior interesse, nesse caso, `PUBLICACAO`. É claro que essa decisão depende da aplicação a ser modelada, mas essencialmente um relacionamento em XML Schema será sempre representado por aninhamento de tipos complexos. O mesmo ocorre com a DTD.

UML também apresenta outros tipos de relacionamentos entre classes, dentre os quais destacam-se dois: Agregação e Composição. Uma agregação pode ser representada da mesma forma que um relacionamento comum, tanto em XML Schema quanto em RDF(S) e DTD. Já uma composição pode ser representada em RDF(S) por uma propriedade do tipo `bag` ou `sequence` [CHA 98]. Em XML Schema e DTD, não há um mecanismo para diferenciar uma composição de uma agregação.

O conceito de herança (Generalização) é bastante claro em RDF e XML Schema, mas deixa a desejar em DTD. Em RDF(S), a propriedade `subClassOf` é utilizada para fazer a especialização de classes. Em XML Schema, a derivação de tipos complexos por extensão pode ser utilizada para implementar esse conceito, já que ela não exige que as propriedades herdadas sejam redeclaradas. Entretanto, XML Schema não possui um mecanismo de derivação de tipos que seja capaz de expressar herança múltipla, pois na derivação de tipos só é possível utilizar um tipo "base". Ao contrário, RDF(S) permite expressar herança múltipla através de `subClassOf`, já que esse mecanismo permite que uma classe seja subclasse de várias classes. XML Schema também não permite que se faça derivação por restrição e extensão ao mesmo tempo. Em UML, é possível que uma classe herde conceitos de outra e ao mesmo tempo redefina (restrinja) alguns dos atributos herdados. Em XML Schema, essa herança tem que ser feita em passos distintos [KLE 00]. Em uma DTD, é possível simular herança através do uso de entidades parâmetro, como mostra a figura 2.1. As classes que fazem parte da hierarquia de herança são definidas como entidades parâmetro para que seu conteúdo não tenha que ser repetido a cada nova definição de classe. É importante ressaltar que a abordagem utilizada nesse trabalho não é a única forma possível de mapear herança em DTDs. [MEL 00a] apresenta outras possibilidades.

A cardinalidade dos relacionamentos possui um mapeamento bem claro em XML Schema: `minOccurs` para a cardinalidade mínima e `maxOccurs` para a cardinalidade máxima. Por outro lado, RDF(S) não fornece um mecanismo para restrição de cardinalidade. Esse conceito pode ser adicionado utilizando o mecanismo de extensão de RDF(S). Um exemplo pode ser encontrado em [BRO 00]. Já em uma DTD, a cardinalidade pode ser expressada através dos símbolos `?`,`*` e `+`. Não existem símbolos especiais para determinar cardinalidade mínima e máxima, pois cada símbolo por si só consegue expressar as duas coisas. Por exemplo, o símbolo `?` expressa cardinalidade mínima 0 e máxima 1.

O conceito de *namespace* de XML Schema e RDF(S) possui um conceito semelhante em UML. Trata-se do conceito de pacote. Através desse conceito, é possível que um diagrama de classes utilize definições feitas em outro diagrama [FOW 00]. Um pacote pode ser pensado como um conjunto de definições

de tipos, do mesmo modo que um *namespace* em XML. Em DTD, o conceito de pacote pode ser mapeado para o conceito de entidade externa.

Já em RDF(S), existe um conceito que não possui equivalente nem em UML nem em XML Schema e nem em DTD. Trata-se do conceito de `rdf:subPropertyOf`. Uma sub-propriedade define uma hierarquia de propriedades, semelhante ao conceito de subclasse, só que para propriedades ao invés de classes.

Analisando os esquemas `publicacao.dtd`, `publicacao.xsd` e `publicacao.rdf`, nota-se que apesar de RDF permitir uma representação direta para os relacionamentos entre as classes, sua sintaxe é bem mais extensa que a de XML Schema e DTD. Além disso, RDF não pode ser utilizada para validar um documento, ou seja, não existem *parsers* que verificam se um determinado documento segue todas as regras definidas em um esquema RDF. Os *parsers* existentes apenas lêem a representação XML de um modelo RDF e identificam as sentenças formadas pela descrição de cada recurso [SWI 00]. Um *parser* RDF(S) *on-line* pode ser encontrado em `http://jigsaw.w3.org:8000/description`.

XML Schema e DTD também possuem algumas restrições quanto ao seu poder de representação de um modelo de classes UML. A principal delas é o fato das classes terem que ser representadas através de uma hierarquia. Desse modo, um modelo de classes não possui uma representação única em XML Schema ou DTD: tudo depende da classe escolhida como raiz da hierarquia. Raízes diferentes resultam em esquemas diferentes.

O que fica claro com tudo isso é que o propósito dos modelos apresentados nesse trabalho são bem diferentes. O propósito de RDF é representar metadados, e não dados propriamente ditos. RDF vem sendo utilizado para descrever páginas HTML, de modo que um mecanismo de busca possa interpretar os metadados dessas páginas e retornar melhores resultados para uma determinada pesquisa. Já XML Schema e DTDs definem um vocabulário XML que pode ser usado para descrever documentos XML.

Tabela 6.1: Comparação entre UML, DTD, XML Schema e RDF

Conceito UML	**DTD**	**XML Schema**	**RDF**
Classe	Elemento complexo	`complexType`	`Class`
Atributo	`!ELEMENT`	`element`	`Property`
Tipos de Atributos	`#PCDATA`	`simpleType`	Não Definido (utiliza os de XML Schema)
Domínio	-	Restrição de `simpleType`	`range`
Visibilidade de atributos	todos os atributos são públicos	todos os atributos são públicos	todos os atributos são públicos
Associação	Elemento aninhado	`complexType` aninhado	`Property` + `range`
Composição	-	-	`Property` + `range`
Agregação	Elemento aninhado	`complexType` aninhado	`Property` + `range`
Generalização	Uso de Entidade Parâmetro	Derivação de `complexType` por Extensão	`subClassOf`
-	-	-	`subPropertyOf`
Herança Múltipla	Uso de Entidades Parâmetro com conector de seqüência	-	`subClassOf`
Cardinalidade Mínima	símbolos `?`,`*` e `+`	`minOccurs`	-
Cardinalidade Máxima	símbolos `?`,`*` e `+`	`maxOccurs`	-
Classe Abstrata	-	Tipo Abstrato	-
Instância	Documento XML	Documento XML	Documento XML com descrições sobre recursos
Pacote	Entidade Externa	*Namespace*	*Namespace*
-	-	`unique`	-
-	-	`key`	-
-	-	`keyref`	-

Referências Bibliográficas

[ABI 97] ABITEBOUL, S. Querying Semistructured Data. In: Proceedings of International Conference on Database Theory, 1997. **Proceedings...** [S.l.: s.n.].

[BIR 00] BIRON, P. V. ; MALHOTRA, A. **XML Schema Part 2:** Datatypes. W3C Candidate Recomendation, 2000. URL `http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/(Nov.2000)`.

[BOU 99] BOURRET, R.; COWAN, J. ; MACHERIUS, Ingo and, L. S. S. **Document Definition Markup Language (DDML) Specification, Version 1.0**. W3C Note, Jan. 1999. URL `http://www.w3.org/TR/NOTE-ddml(Dez.2000)`.

[BOX 00] BOX, D.; SKONNARD, A. ; LAM, J. **Essencial XML:** Beyond Markup. Addison Wesley, Jul. 2000.

[BRA 99] BRAY, T.; HOLLANDER, D. ; LAYMAN, A. **Namespaces in XML**. W3C Recomendation, 1999. URL `http://www.w3c.org/TR/1999/REC-xml-names-19990114/(Aug.2000)`.

[BRA 00] BRADLEY, N. **The XML Companion**. Addison-Wesley, 2nd ed., 2000.

[BRI 00] BRICKLEY, D. ; GUHA, R. V. **Resource Description Framework (RDF) Schema Specification 1.0**. W3C Proposed Recomendation, Mar. 2000. URL `http://www.w3c.org/TR/2000/CR-rdf-schema-0-20000327/(Aug.2000)`.

[BRO 00] BROEKSTRA, J. et al. **Adding Formal Semantics to the Web**, Set. 2000. URL `http://www.ontoknowledge.org/oil/papers/extending-rdfs.html(Oct.2000)`.

[CHA 98] CHANG, W. **A Discussion of the Relationship Between RDF-Schema and UML**. W3C Note, Ago. 1998. URL `http://www.w3.org/TR/1998/NOTE-rdf-uml-19980804/`.

[CHA 00] CHAMPIN, P.-A. **RDF Tutorial**, Jun. 2000. URL `http://www710.univ-lyon1.fr/~champin/rdf-tutorial/(Aug.2000)`.

[CLA 99] CLARK, James DEROSE, S. **XML Path Language**. W3C Recomendation, Nov. 1999. URL `http://www.w3.org/TR/xpath.html(Sep.2000)`.

[DOR 00] DORNELES, C. F. **Extraçao de Dados Semi-Estruturados com Base em uma Ontologia**. Ppgc-ufrgs: Dissertação de mestrado, UFRGS, Porto Alegre, RS, Fev. 2000.

[FAL 00] FALLSIDE, D. C. **XML Schema Part 0:** Primer. W3C Candidate Recomendation, 2000. URL `http://www.w3.org/TR/xmlschema-0/(Nov.2000)`.

[FOW 00] FOWLER, M. ; SCOTT, K. **UML Essencial**. Bookman, 2000.

[JEL 00] JELLIFFE, R. **W3C XML Schema Datatypes Reference**, Nov. 2000. URL `http://www.xml.com/pub/a/2000/11/29/schemas/dataref.html(Dez.2000)`.

[KLA 99] KLARLUND, N.; MÜLLER, A. ; SCHWARTZBACH, M. Document Structure Description 1.0, Out. 1999. URL `http//www.brics.dk/DSD/specification.html(Aug.2000)`.

[KLA 00] KLARLUND, N.; MÜLLER, A. ; SCHWARTZBACH, M. DSD: A Schema Language for XML, Jun. 2000. URL `http//www.brics.dk/DSD(Aug.2000)`.

[KLE 00] KLEIN, M.; HARMELEN, F. v. ; HORROCKS, I. The Relation Between Ontologies and XML Schemata. In: Procedings of the Workshop on Applications of Ontology and Problem-Solving Methods, Ago. 2000. **ECAI-00, 14th European Conference on Artificial Intelligence**. Berlin: [s.n.]. Disponível por HTTP em: `http://www.cs.vu.nl/~mcaklein/papers/oil-xmls.pdf(Nov.2000)`.

[LAS 99] LASSILA, O. ; SWICK, R. R. **Resource Description Framework (RDF) Model and Sintax Specification**. W3C Recomendation, 1999. URL `http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222/(Nov.2000)`.

[McG 98] McGRATH, S. **XML By Example**. Prentice Hall PTR, 1st ed., 1998.

[MEL 00a] MELLO, R. **Uma Camada de Mediação para Integração de Fontes XML com o Suporte de Ontologias**. Ph.D. thesis, PPCG/UFRGS, Porto Alegre, RS, 2000. (em andamento).

[MEL 00b] MELLO, R. et al. Dados Semi-Estruturados. In: SBBD 2000 - XV Simpósio Brasileiro de Banco de Dados, João Pessoa, PB, 2000. **Anais...** [S.l.: s.n.].

[MIC 00] MICROSOFT. **XML Schema Developers Guide**. Msdn on-line WEB Workshop, 2000. URL `http://msdn.microsoft.com/xml/xmlguide/schema-overview.asp(Mai.2000)`.

[OMG 00] OMG. **OMG Unified Modeling Language Specification**, Mar. 2000. URL `http://www.omg.org/technology/documents/formal/unified_modeli%ng_language.htm(Dez.2000)`.

[PIM 00] PIMENTEL, M. d. G. C.; TEIXEIRA, C. A. C. ; SANTANCHÊ, A. XML: Explorando suas Aplicações na Web. In: XIX Jornada de Atualização em Informática, Curitiba, PR, 2000. **Anais...** [S.l.: s.n.].

[SAH 00] SAHUGUET, A. Everything you ever wanted to know about DTDs, but were afraid to ask. In: WebDB-2000, 2000. **Procedings...** [S.l.: s.n.]. Disponível por HTTP em: `http://db.cis.upenn.edu/DL/(Apr.2000)`.

[SIM 99] SIMPSON, J. **Just XML**. Upper Saddle River: Prentice Hall, 1st ed., 1999.

[STA 00a] STAAB, S. et al. **Exemplo de instância representada em RDF(S)**, 2000. URL `http://ontoserver.aifb.uni-karlsruhe.de/schema/ka2example.rdf(Nov.2000)`.

[STA 00b] STAAB, S. et al. An Extensible Approach for Modeling Ontologies in RDF(S). In: ECDL 2000 Workshop on the Semantic Web, 2000. **Proceedings...** [S.l.: s.n.]. Disponível por HTTP em: `http://www.aifb.uni-karlsruhe.de/WBS(Sep.2000)`.

[SWI 00] SWICK, R. et al. **Resource Description Framework (RDF)**. W3C, 2000. URL `http://www.w3c.org/RDF/(Nov.2000)`.

[VLI 00] VLIST, E. v. d. **Using XML Schemas**, Nov. 2000. URL `http://www.xml.com/pub/a/2000/11/29/schemas/part1.html(Dez.2000)`.