XML Schema, RDF(S) e UML: uma Comparação

Vanessa de Paula Braganholo
Carlos A. Heuser
Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática*
Porto Alegre - RS - Brasil
e-mail: {vanessa,heuser}@inf.ufrgs.br

Resumo

XML Schema, RDF Schema e UML são três propostas que podem ser utilizadas para modelagem de dados, assim como para vários outros propósitos. Entre eles podemos citar a descrição de documentos XML (XML Schema), descrição de metadados (RDF(S)) e análise e projeto de aplicações Orientadas a Objeto (UML). Este trabalho baseia-se em um estudo de caso que apresenta a modelagem de uma aplicação UML e faz o mapeamento desta aplicação para XML Schema e RDF Schema. Apesar da similaridade de seus nomes, RDF Schema possui um papel diferente do de XML Schema. XML Schema 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 [3]. Este trabalho tem como objetivo comparar estas três propostas, de modo que uma pessoa que conheça um dos modelos possa compreender os outros mais facilmente.

1 Introdução

XML (*eXtensible Markup Language*) vem se estabelecendo como um padrão para representação de dados semi-estruturados e vem sendo utilizada para descrever dados que serão manipulados por aplicações ou trocados através da Web.

Um diagrama de classes UML (*Unified Modeling Language*) é um padrão para modelagem de dados definido em [12]. Um documento XML pode representar instâncias de classes definidas em um modelo UML. Por conseqüência, pode-se definir modelos XML que representem estes dados, da mesma forma que um diagrama de classes UML o faz.

O objetivo deste trabalho é apresentar duas propostas de esquemas para dados XML definidas pela W3C (*World Wide Web Consortium*): XML Schema [8] e RDF(S) [2], e compará-las com o diagrama de classes de 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, 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. Um diagrama de classes UML descreve classes de objetos, seus relacionamentos e atributos. XML Schema visa estabelecer regras que os documentos XML devem seguir. 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. 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.

Este trabalho está estruturado como segue. A seção 2 apresenta um estudo de caso que será utilizado para fazer a comparação das duas propostas. Os principais conceitos de XML Schema são apresentados

*Este trabalho tem financiamento parcial de CNPq, IBM e Solectron

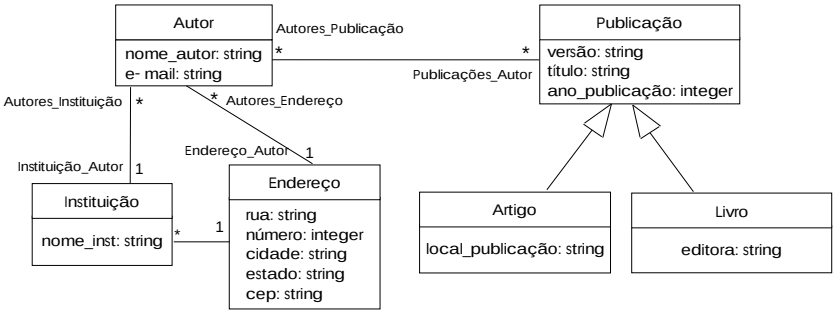

Figura 1: Modelo de classes para publicações e autores

na seção 3 e os de RDF(S) são mostrados na seção 4. A seção 5 faz a comparação das duas propostas e apresenta algumas conclusões.

2 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. Esse modelo será utilizado ao longo deste trabalho e será mapeado para RDF(S) e XML Schema, de modo a facilitar a comparação entre as três propostas.

A figura 2 apresenta um documento XML que possui os conceitos apresentados no modelo de classes da figura 1. Esse documento pode ser considerado uma representação das instâncias das classes definidas na figura 1, com algumas singularidades. Uma delas é que o modelo de classes da figura 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 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: 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. Outro 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.

3 XML Schema

XML Schema é um Candidato a Recomendação da W3C para dar estrutura a documentos XML. 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 assim como um esquema descreve um documento instância [1].

Como XML Schema descreve um documento XML instância, e este 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, é mostrado na figura 4.

```
<?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 2: Exemplo de documento XML

Figura 3: Representação para a população de um tipo em UML

3.1 Conceitos Básicos

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

Um `simpleType` pode ser um dos tipos básicos definidos em XML Schema, tais como *string*, *integer*, *date*, entre outros. Uma relação completa dos tipos simples definidos por XML Schema pode ser encontrada em [9].

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`. O atributo `minOccurs` especifica o número mínimo de vezes que um subelemento pode aparecer, e `maxOccurs`, o número máximo. Os valores destes atributos devem ser inteiros positivos, mas o atributo `maxOccurs` também permite o valor `unbounded` [13]. Na figura 4, 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.

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. 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 [8].

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>
```

Definições de tipo podem ser reutilizadas através de uma referência a seu nome. Na figura 4, o tipo complexo `tEndereco` é utilizado como parte do tipo complexo `tAutor` e `tInstituicao`. Além disso, elementos globais podem ser referenciados em outras declarações. A figura 4 mostra uma referência ao elemento global `ANO_PUBLICACAO` na declaração do tipo complexo `tPublicacao`.

Em XML Schema, atributos são declarados com `attribute`. Como ressaltado anteriormente, atributos em XML costumam ser utilizados para representar metadados. Na figura 4, o tipo complexo `tPublicacao` possui um atributo `VERSAO` que corresponde ao atributo **versão** do diagrama de classes da figura 1.

```
<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>
```

Figura 4: XML Schema para o documento da figura 2

Este atributo é um metadado da classe Publicação, mas em UML ele foi representado junto com os outros atributos da classe. UML não permite separar metadados de dados, ao contrário de XML.

Do mesmo modo que uma declaração `element` liga um tipo simples ou complexo a um nome de elemento em um contexto, uma declaração de atributo liga um tipo simples a um nome de atributo em um contexto [1].

3.2 Derivação de Tipos e Tipos Abstratos

Em XML Schema, tipos podem ser derivados por restrição ou por extensão. A figura 4 mostra dois exemplos de derivação de tipos por extensão: os tipos `tArtigo` e `tLivro` são derivados de `tPublicacao`. Nesse caso, as definições dos tipos `tArtigo` e `tLivro` apenas adicionam elementos ao tipo `tPublicacao`, simulando assim o conceito de herança de UML.

A figura 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 deste conceito, é permitido utilizar o modelo de conteúdo de `ARTIGO` ou `LIVRO` nos locais onde `PUBLICACAO` é permitido, como mostrado abaixo [1].

```
<?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>
```

Em derivações por restrição, troca-se a palavra `extension` por `restriction` e, ao invés de adicionar elementos ou atributos ao tipo, faz-se restrições de cardinalidade a elementos ou se utilizam tipos mais restritos para um determinado elemento. Por exemplo, pode-se dizer que um elemento do tipo *integer* deve agora ser do tipo *positiveInteger*. Ao contrário da derivação por extensão, na derivação por restrição é necessário repetir todo o modelo de conteúdo do tipo que está sendo restringido. Pode-se pensar na derivação de tipos simples por restrição como a definição de um domínio em UML.

Na figura 4, 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. Além das expressões regulares, existem outras formas de restringir um tipo simples. Pode-se definir valores máximos e mínimos para um determinado tipo, um conjunto de valores fixos, entre outros.

Além da derivação de tipos, 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"/>
```

Para que um tipo abstrato seja utilizado, é necessário declarar um tipo não abstrato derivado dele. 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. Neste 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"/>
```

3.3 População, Unicidade, Chaves e Referências

XML Schema permite indicar que o valor de um elemento ou atributo deve ser único em um certo escopo [8]. 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 [7]. 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="UnicidadeCodigoAutor">
  <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`.

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*) [12]. No entanto, este trabalho considera apenas o diagrama de classes de UML.

3.4 Namespaces

A declaração abaixo descreve um *namespace* para o esquema definido na figura 4, e associa a ele um prefixo `pub`. Esse prefixo aparece quando um tipo definido neste esquema é utilizado para declarar um elemento ou atributo.

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

O atributo `schemaLocation` pode ser utilizado na instância XML para ligar um namespace à localização física do arquivo que contém o esquema.

```
<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/Esquemas/Publicacao/Publicacao.xsd">
<!-- elementos do documento -->
</pub:PUBLICACOES>
```

O uso de *namespaces* aumenta a flexibilidade de XML Schema, permitindo a reutilização de definições feitas em outros esquemas. Pode-se também redefinir tipos declarados em um determinado *namespace*. Maiores detalhes sobre o uso de *namespaces* em XML podem ser encontrados em [6].

Em UML, o conceito espelho de *namespace* é o conceito de pacotes.

Além das características apresentadas neste trabalho, XML Schema permite, ainda, expressar valores nulos, importar e exportar tipos através do uso de *namespaces*, entre outros. Mais informações sobre estas características podem ser encontradas em [8].

4 RDF Schema

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 [11], define como descrever recursos em termos de suas propriedades e valores; a segunda parte, RDF Schema [2], define propriedades específicas que podem ser utilizadas para definir esquemas. Essas duas definições juntas costumam ser referidas como RDF(S) [15].

RDF utiliza o conceito de **sentença**. Uma sentença é um par propriedade-valor e um recurso ao qual esta propriedade se aplica. Sentenças são 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 [4]. 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.

Também são definidos três tipos de objeto Container: `Bag`, `Sequence` e `Alternative`. A tabela 1 mostra as propriedades de cada um desses tipos de objeto.

Tabela 1: Tipos de Objeto Container de RDF

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

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.

O vocabulário de RDF está definido em dois *namespaces*: `rdf` e `rdfs`. Declarações RDF devem ser precedidas de um dos dois prefixos, dependendo de qual das duas especificações definiu a propriedade em questão.

```
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 na figura 5. Por questões de espaço, são mostradas apenas as definições das classes Autor, Publicação e Artigo. Uma descrição completa do modelo RDF para as classes da figura 1 pode ser encontrada em `www.inf.ufrgs.br/~vanessa/artigos/classes.rdf`.

Como tudo em RDF é considerado um recurso, RDF Schema estabelece que esses recursos 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 de subclasses. Isso significa que qualquer recurso de um tipo que é subclasse de outro, é também considerado como sendo do tipo da superclasse. Tal relacionamento entre classes é denotado através da propriedade `subClassOf`.

O sistema de tipos de RDF Schema é semelhante ao sistema de tipos do modelo de classes de UML, com algumas pequenas diferenças. Uma delas é que, ao invés de definir classes em termos das propriedades que suas instâncias devem ter, um RDF Schema define propriedades em termos das 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 em UML seria definida uma classe `Livro` com um atributo chamado `editora` do tipo `string` [2].

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

```
<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="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>
<!-- 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>
</rdf:RDF>
```

Figura 5: Esquema RDF(S) para publicações e autores

```
<?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/Publicacao.rdfs#"
     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/Publicacao.rdfs#Publicacao"/>
   <rdf:type rdf:resource="www.inf.ufrgs.br/~vanessa/Publicacao/Publicacao.rdfs#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>
```

Figura 6: Instância da classe Artigo representada em RDF

4.1 Classes

RDF Schema considera que os recursos `rdfs:Resource`, `rdf:Property` e `rdfs:Class` são classes.

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

A classe `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 seja o recurso `rdfs:Class`.

4.2 Propriedades

Todo modelo RDF que utiliza o mecanismo de esquema também inclui as propriedades `rdf:type`, `rdf:subClassOf` e `rdf:subPropertyOf`. 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 que são esperadas de um membro da classe em questão. 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 mostra a figura 6 [14].

Apesar da instância acima mostrar um artigo com dois autores, esta restrição de cardinalidade para o relacionamento de Publicação com Autor não foi feita no esquema RDF(S) da figura 5. Isso deve-se ao fato de que RDF(S) não possui um mecanismo de restrição de cardinalidades como em UML.

A propriedade `rdf:subClassOf` é transitiva e especifica uma relação subconjunto/superconjunto entre classes. 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 desta propriedade é sempre do tipo `rdf:type rdfs:Class` [4]. 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.

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, isso implica que o recurso A também tem uma propriedade P1 com valor B [2].

4.3 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. As restrições `rdfs:range` e `rdfs:domain` são utilizadas para este propósito.

Uma restrição `range` diz que o valor de uma propriedade deve ser um recurso de uma determinada classe. Por exemplo, uma restrição `range` aplicada à propriedade `endereco_autor` expressa que o valor da propriedade `endereco_autor` deve ser um recurso da classe `Endereco`. Já uma restrição `domain` diz que uma propriedade pode ser utilizada em recursos de uma determinada classe. Por exemplo, a propriedade `endereco_autor` só pode se originar de um recurso que é uma instância da classe `Autor`. O nome da propriedade que implementa o relacionamento entre duas classes pode ser o nome do papel atribuído ao relacionamento no diagrama de classes UML.

5 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 [10]. Esta seção apresenta os mapeamentos desses conceitos para XML Schema e RDF(S). A tabela 2 faz um resumo do mapeamento dos conceitos de UML para as estruturas disponíveis nas especificações de XML Schema e RDF Schema, respectivamente.

Tabela 2: Resumo da comparação entre UML, XML Schema e RDF

Conceito UML	**XML Schema**	**RDF**
Classe	`complexType`	`Class`
Atributo	`element`	`Property`
Tipos de Atributos	`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
Associação	`complexType` aninhado	`Property` + `range`
Composição	-	`Property` (`bag` ou `sequence`)
Agregação	`complexType` aninhado	`Property` + `range`
Generalização	Derivação de `complexType` por Extensão	`subClassOf`
-	-	`subPropertyOf`
Herança Múltipla	-	`subClassOf`
Cardinalidade Mínima	`minOccurs`	-
Cardinalidade Máxima	`maxOccurs`	-
Classe Abstrata	Tipo Abstrato	-
Instância	Documento XML	Documento XML com descrições sobre recursos
Pacote	*Namespace*	*Namespace*
-	`unique`, `key` e `keyref`	-

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 na figura 4. Conseqüentemente, os atributos podem ser representados em XML Schema através de elementos de um tipo complexo e, em RDF por propriedades (`Property`) de classes. Os atributos (`attribute`) em XML Schema também poderiam ser utilizados para representar atributos de uma classe. [13] afirma que os atributos dos elementos XML são, em geral, reservados para armazenar dados sobre dados, ou metadados. Portanto, os atributos em XML Schema podem ser utilizados para representar metadados de uma classe UML. Neste 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. Neste caso seu tipo deve ser um `complexType`. Em RDF, o elemento `range` é utilizado para representar o tipo de uma propriedade.

UML fornece três tipos de visibilidade de atributos para uma classe: *public*, *protected* e *private*. Em 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, 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, as classes Autor e Instituição possuem um relacionamento 1 para n. No XML Schema correspondente (figura 4), 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, neste 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.

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). Já uma composição pode ser representada em RDF por uma propriedade do tipo `bag` ou `sequence` [5]. Em XML Schema, não há um mecanismo para diferenciar uma composição de uma agregação.

O conceito de herança (Generalização) também é bastante claro nas duas propostas. 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 este mecanismo permite que uma classe seja subclasse de várias classes.

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 [3].

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 [10]. Um pacote pode ser pensado como um conjunto de definições de tipos, do mesmo modo que um *namespace* em XML.

Já em RDF(S), existe um conceito que não possui equivalente nem em UML nem em XML Schema. 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 a figura 4 e o esquema RDF exemplo apresentado na figura 5, 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. 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 [16]. Um *parser* RDF(S) *on-line* pode ser encontrado em `http://jigsaw.w3.org:8000/description`.

XML Schema também possui 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. Deste modo, um modelo de classes não possui uma representação única em XML Schema: 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 neste artigo são bem dife-

rentes. 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 destas páginas e retornar melhores resultados para uma determinada pesquisa. Já XML Schema define um vocabulário XML que pode ser usado para descrever documentos XML.

Referências

[1] Don Box, Aaron Skonnard, and John Lam. *Essencial XML*. Addison Wesley, July 2000.

[2] Dan Brickley and R. V. Guha. Resource description framework (rdf) schema specification 1.0. W3C Proposed Recomendation, March 2000. http://www.w3c.org/TR/2000/CR-rdf-schema-0-20000327/.

[3] Jeen Broekstra, Michel Klein, Stefan Decker, Dieter Fensel, and Ian Horrocks. Adding formal semantics to the web. http://www.ontoknowledge.org/oil/papers/extending-rdfs.html, Sep 2000.

[4] Pierre-Antoine Champin. Rdf tutorial. http://www710.univ-lyon1.fr/ champin/rdf-tutorial/, June 2000.

[5] W. Chang. A discussion of the relationship between rdf-schema and uml. W3C Note, aug 1998. http://www.w3.org/TR/1998/NOTE-rdf-uml-19980804/.

[6] W3C Consortium. Namespaces in xml. W3C Recomendation, 1999. http://www.w3c.org/TR/1999/REC-xml-names-19990114/.

[7] W3C Consortium. Xml path language. W3C Recomendation, November 1999. http://www.w3.org/TR/xpath.html.

[8] W3C Consortium. Xml schema part 0: Primer. W3C Candidate Recomendation, 2000. http://www.w3.org/TR/xmlschema-0/.

[9] W3C Consortium. Xml schema part 2: Datatypes. W3C Candidate Recomendation, 2000. http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/.

[10] Martin Fowler and Kendall Scott. *UML Essencial*. Bookman, 2000.

[11] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and sintax specification. W3C Recomendation, 1999. http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222/.

[12] Object Management Group OMG. Omg unified modeling language specification. http://www.omg.org/technology/documents/formal/unified_modeling_language.htm, March 2000.

[13] Maria da Graça Campos Pimentel, Cesar Augusto Camillo Teixeira, and André Santanchè. Xml: Explorando suas aplicações na web. In *XIX Jornada de Atualização em Informática*, Curitiba, 2000.

[14] Steffen Staab, Michael Erdmann, Alexander Maedche, and Stefan Decker. Exemplo de instância representada em rdf(s). http://ontoserver.aifb.uni-karlsruhe.de/schema/ka2example.rdf.

[15] Steffen Staab, Michael Erdmann, Alexander Maedche, and Stefan Decker. An extensible approach for modeling ontologies in rdf(s). In *ECDL 2000 Workshop on the Semantic Web*, 2000.

[16] Ralph Swick, Eric Miller, Bob Schloss, David Singer, and Dan Brickley. Resource description framework (rdf). W3C, 2000. http://www.w3c.org/RDF/.