By Michel Klein
Vrije Universiteit

XML, RDF, and Relatives

Languages for representing data and knowledge are an important aspect of the Semantic Web. And there are a lot of languages around! Most languages are based on XML or use XML as syntax; some have connections to RDF or RDF Schemas. This tutorial will briefly introduce XML, XML Schemas, RDF, and RDF Schemas.

Let's start with XML

XML (eXtensible Markup Language) is a specification for computer-readable documents. *Markup* means that certain sequences of characters in the document contain information indicating the role of the document's content. The markup describes the document's data layout and logical structure and makes the information self-describing, in a sense. It takes the form of words between pointy brackets, called *tags*—for example, <name> or <h1>. In this aspect, XML looks very much like the well-known language HTML.

However, *extensible* indicates an important difference and a main characteristic of XML. XML is actually a *metalanguage*: a mechanism for representing other languages in a standardized way. In other words, XML only provides a data format for structured documents, without specifying an actual vocabulary. This makes XML universally applicable: you can define customized markup languages for unlimited types of documents. This has already occurred on a massive scale. Besides many proprietary languages—ranging from electronic order forms to application file formats—a number of standard languages are defined in XML (called *XML applications*). For example, XHTML is a redefinition of HTML 4.0 in XML.

Let's take a more detailed look at XML. The main markup entities in XML are *elements*. They consist normally of an opening tag and a closing tag—for example, <person> and </person>. Elements might contain other elements or text. If an element has no content, it can be abbreviated as <person/>. Elements should be properly nested: a child element's opening and closing tags must be within its parent's opening and closing tags. Every XML document must have exactly one root element. Elements can carry *attributes* with values, encoded as additional "word = value" pairs inside an element tag—for example, <person name="John">. Here is a piece of XML:

```
<?xml version="1.0"?>
<employees>
    List of persons in company:
    <person name="John">
        <phone>47782</phone>
        On leave for 2001.
    </person>
</employees>
```

XML does not imply a specific interpretation of the data. Of course, on account of the tag's names, the meaning of the previous piece of XML seems obvious to human users, but it is not formally specified! The only legitimate interpretation is that XML code contains named entities with subentities and values; that is, every XML document forms an ordered, labeled tree. This generality is both XML's strength and its weakness. You can encode all kinds of data structures in an unambiguous syntax, but XML does not specify the data's use and semantics. The parties that use XML for their data exchange must agree beforehand on the vocabulary, its use, and its meaning.

Enter DTDs and XML Schemas

Such an agreement can be partly specified by *Document Type Definitions* and XML Schemas. Although DTDs and XML Schemas do not specify the data's meaning, they do specify the names of elements and attributes (the vocabulary) and their use in documents. Both are mechanisms with which you can specify the structure of XML documents. You can then validate specific documents against the structure prescription specified by a DTD or an XML Schema.

DTDs provide only a simple structure prescription: they specify the allowed nesting of elements, the elements' possible attributes, and the locations where normal text is allowed. For example, a DTD might prescribe that every person element must have a name attribute and may have a child element called phone whose content must be text. A DTD's syntax looks a bit awkward, but it is actually quite simple.

XML Schemas are a proposed successor to DTDs. The XML Schema definition is still a candidate recommendation from the W3C (World Wide Web Consortium), which means that, although it is considered stable, it might still undergo small revisions. XML Schemas have several advantages over DTDs. First, the XML Schema mechanism provides a richer grammar for prescribing the structure of elements. For example, you can specify the exact number of allowed occurrences of child elements, you can specify default values, and you can put elements in a *choice* group, which means that exactly one of the elements in that group is allowed at a specific location. Second, it provides data typing. In the example in the previous paragraph, you could prescribe the phone element's content as five digits, possibly preceded by another five digits between brackets. A third advantage is that the XML Schema definition provides inclusion and derivation mechanisms. This lets you reuse common element definitions and adapt existing definitions to new practices.

A final difference from DTDs is that XML Schema prescriptions use XML as their encoding syntax. (XML is a metalanguage, remember?) This simplifies tool development, because both the structure prescription and the prescribed documents use the same syntax. The XML Schema specification's developers exploited this feature by using an XML Schema document to define the class of XML Schema documents. After all, because an XML Schema prescription is an XML application, it must obey rules for its structure, which can be defined by another XML Schema prescription. However, this recursive definition can be a bit confusing.

RDF represents data about data

XML provides a syntax to encode data; the resource description framework is a mechanism to tell something about data. As its name indicates, it is not a language but a model for representing data about "things on the Web." This type of data about data is called *metadata*. The "things" are *resources* in RDF vocabulary.

RDF's basic data model is simple: besides resources, it contains *properties* and *statements*. A property is a specific aspect, characteristic, attribute, or relation that describes a resource. A statement consists of a specific resource with a named property plus that property's value for that resource. This value can be another resource or a *literal* value: free text, basically. Altogether, an RDF description is a list of triples: an object (a resource), an attribute (a property), and a value (a resource or free text). For example, Table 1 shows the three triples necessary to state that a specific Web page was created by something with a name "John" and a phone number "47782."

Table 1. An RDF description consisting of three triples indicating that a specific Web page was created by something with a name John and a phone number "47782."

Object	Attribute	Value
http://www.w3.org/	created_by	#anonymous_resource1
#anonymous_resource1	name	"John"
#anonymous_resource1	phone	"47782"

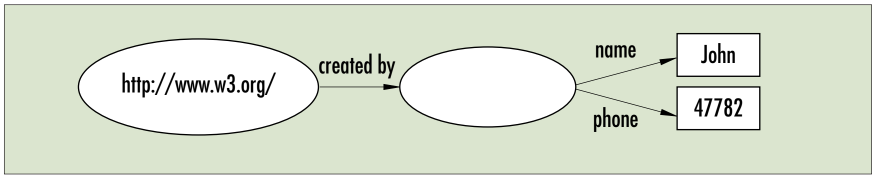

Figure 1. A directed labeled graph for the triples in Table 1.

You can easily depict an RDF model as a directed labeled graph. To do this, you draw an oval for every resource and an arrow for every property, and you represent literal values as boxes with values. Figure 1 shows such a graph for the triples in Table 1.

These example notations reveal that RDF is ignorant about syntax; it only provides a model for representing metadata. The triple list is one possible representation, as is the labeled graph, and other syntactic representations are possible. Of course, XML would be an obvious candidate for an alternative representation. The specification of the data model includes such an XML-based encoding for RDF.

As with XML, an RDF model does not define (a priori) the semantics of any application domain or make assumptions about a particular application domain. It just provides a domain-neutral mechanism to describe metadata. Defining domain-specific properties and their semantics requires additional facilities.

Defining an RDF vocabulary: RDF Schema

Basically, RDF Schema is a simple type system for RDF. It provides a mechanism to define domain-specific properties and classes of resources to which you can apply those properties.

The basic modeling primitives in RDF Schema are *class* definitions and *subclass-of* statements (which together allow the definition of class hierarchies), *property* definitions and *subproperty-of* statements (to build property hierarchies), *domain* and *range* statements (to restrict the possible combinations of properties and classes), and *type* statements (to declare a resource as an instance of a specific class). With these primitives you can build a schema for a specific domain. In the example I've been using throughout this tutorial, you could define a schema that declares two classes of resources, Person and WebPage, and two properties, name and phone, both with the domain Person and range Literal. You could use this schema to define the resource http://www.w3.org/ as an instance of WebPage and the anonymous resource as an instance of Person. Together, this would give some interpretation and validation possibilities to the RDF data.

RDF Schema is quite simple compared to full-fledged knowledge representation languages. Also, it still does not provide exact semantics. However, this omission is partly intentional; the W3C foresees and advocates further extensions to RDF Schema.

Because the RDF Schema specification is also a kind of metadata, you can use RDF to encode it. This is exactly what occurs in the RDF Schema specification document. Moreover, the specification provides an RDF Schema document that defines the properties and classes that the RDF Schema specification introduced. As with the XML Schema specification, such a recursive definition of RDF Schema looks somewhat confusing.

XML and RDF are different formalisms with their own purposes, and their roles in the realization of the Semantic Web vision will be different. XML aims to provide an easy-to-use syntax for Web data. With it, you can encode all kinds of data that is exchanged between computers, using XML Schemas to prescribe the data structure. This makes XML a fundamental language for the Semantic Web, in the sense that many techniques will probably use XML as their underlying syntax.

XML does not provide any interpretation of the data beforehand, so it does not contribute much to the "semantic" aspect of the Semantic Web. RDF provides a standard model to describe facts about Web resources, which gives some interpretation to the data. RDF Schema extends those interpretation possibilities somewhat more. However, to realize the Semantic Web vision, it will be necessary to express even more semantics of data, so further extensions are needed. There are already some initial steps in this direction—for example, the DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer) language, which adds new modeling primitives and formal semantics to RDF Schema.

The "Further Reading" sidebar contains pointers to more detailed explanations of XML and RDF and lists the URLs of the official homepages of XML, RDF, and the Semantic Web Activity at the W3C. Through those pages, you can find many projects and applications related to these topics. ∎

Further Reading

- The pages at www.w3.org/XML and www.w3.org/RDF contain pointers to the official definitions of the languages that I covered in this minitutorial.
- XML.com (www.xml.com) contains technical introductions to both XML and XML Schemas.
- Pierre-Antoine Champin provides comprehensive tutorial on RDF and RDF Schema at www710.univ-lyon1.fr/~champin/rdf-tutorial.
- Robin Cover maintains a comprehensive online reference for XML and related techniques at www.oasis-open.org/cover.
- The vision of the Semantic Web is sketched at www.w3.org/2001/sw/Activity.
- The DAML+OIL extension to RDF Schema lives at www.daml.org/2001/03/daml+oil-index.

Michel Klein is a PhD student at the Information Management Group of the Vrije Universiteit in Amsterdam. His research interests include ontology modeling, maintenance, and integration, and representation and interoperability issues of semistructured data. Contact him at the Faculty of Sciences, Division of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, Netherlands; michel.klein@cs.vu.nl; www.cs.vu.nl/~mcaklein.