Transforming Valid XML Documents into RDF via RDF Schema

Pham Thi Thu Thuy, Young-Koo Lee*, Sungyoung Lee, and Byeong-Soo Jeong
Department of Computer Engineering, Kyung Hee University, Korea
{tttpham, sylee}@oslab.khu.ac.kr, {yklee, jeong}@khu.ac.kr

Abstract

Interpreting the XML data in a current web into sources that can be used by the Semantic Web has received great attention in recent years. In this paper, we propose a procedure for transforming valid XML documents into RDF by using vocabularies of RDF schema. The first objective here is to obtain classes and properties from labels in XML document exactly by accessing the XML DTD. After that, we can interpret XML data as RDF triples by using some vocabularies of RDF schema. The main advantage of our approach is that it ensures the integrity of the structure and meaning of the original XML documents while transforming them into RDF. This procedure can be used for any kind of valid XML documents.

1. Introduction

Most of the web sites today are designed for human reading, not for computer understanding. Computers essentially play a role in parsing web pages for displaying and processing jobs. They have no reliable way to draw the semantics from a page [2]. The Semantic Web will improve the meaningful content of the web pages. It is not completely a new generation of web, but an expansion of the current one. The meaning in the Semantic Web is mostly represented by Resource Description Framework (RDF). RDF encrypts these meanings in the sets of triples that build meaningful webs about related things. These are recognized by the Universal Resource Identifiers (URIs) which tie meanings to a unique definition so that users can easily find them and their relationships on the web [2].

However, a considerable amount of resources is available in eXtensible Markup Language (XML) rather than in RDF. The main success of XML is its flexibility. Users can define their own tags to describe elements in the XML document. Moreover, they can also predefine the structure of XML documents by writing a Document Type Definition (DTD). The XML document, obeying the XML syntaxes, is called well-formed XML document. If a well-formed XML document is created based on the construction in a DTD, it is called a valid XML document. Usually, DTD is used as a standard mechanism to exchange information on the web. For example, in the electronic commerce, when the associates are unanimous in a common DTD, they will produce valid XML documents and carry out their communication. This provides us a large number of valid XML documents. Alternatively, users can draw DTD from a well-formed XML document by following its construction and labels. Otherwise, there is a tool helping to draw DTD from XML documents, such as DTDMaker [3].

Although XML plays an important role in structuring the document, it has disadvantages when coming to the semantic interoperability. XML mainly focuses on the grammar but there is no way to describe the semantics of the document. Moreover, because XML enables users to define their own tags, an object can be described in different ways. For instance, we label something as <price>@12.00</price> and another organization labels the same field as <cost>$12.00</cost>. In this case, a machine cannot differentiate between two meanings unless Semantic Web technologies such as RDF are added [4]. Furthermore, in the Semantic Web, the operability requires not only the structured data but also the semantic content [5]. Therefore, we cannot directly use XML data for the Semantic Web, and need another language to interpret this data.

Though, the general purpose language for representing information in the Semantic Web is RDF, it cannot describe classes and properties in structured documents. Instead, they are depicted by the RDF Vocabulary Description Language 1.0: RDF schema, shortly, RDF schema [21]. Therefore, our procedure interprets valid XML documents as RDF model and uses vocabularies of the RDF schema. Our main contribution is a set of rules that derive classes and properties from XML DTD and interpret XML data as RDF statements by using RDF schema vocabularies.

* Corresponding author

The remainder of the paper is organized as follows. In section 2, we briefly introduce the related work. Section 3 describes the role of XML, RDF and RDF schema in representing knowledge on the web. This will be followed by the algorithm of the procedure and the corresponding example in section 4. Finally, section 5 concludes this paper.

2. Related work

The transforming XML into RDF is not new. Sergey Melnik [6] was one of the pioneers proposing an algorithm to extract RDF triples from XML documents. This algorithm creates a version of RDF that can process arbitrary XML document. However, it mainly focuses on how to handle all XML elements but does not concern about exploiting domain's information. Therefore, the issues follow the structures in XML but bear little meaning and do not fit well into RDF model.

Another approach is presented in the C-Web project [7]. This method uses XPath, an XML query language, to map information in XML documents to domain specific ontologies. This proposal exploits more specific meaning and structure of the XML documents. However, beside reference to XML document and its DTD, it requires referring to another resource, the specification of rules, which is not a requirement in our approach.

In another paper, Michel Klein introduces a procedure to interpret XML statements as RDF data via RDF schema specification [8]. This approach is close to our method. However, it does not transform all XML elements. Instead it concentrates on translating some pieces of information in the XML document. Moreover, elements in XML document are decided to be classes or properties depending upon user's opinion. Therefore, the results of this approach could be different among users' point of view. Our method bases on XML DTD, it can transform every label in XML statements, which is defined in DTD, into RDF. Therefore, the results follow the data structure, and maintain the meaning of the original XML documents.

There are several other approaches giving new XML syntax for RDF. These approaches use XML to define a language to represent the triples. For example, there are the "strawman unstriped syntax" of Tim Berners-Lee [9], and Jonathan Borden's syntax [10]. Similarity, authors in SIMILE project use XSLT to convert XML to RDF/XML [11]. Our method does not provide a new XML syntax for RDF, but uses DTD to extract RDF statements from that XML document.

3. Knowledge representation

There are three essential requirements for arbitrary language used for data interchange on the web:

1) Language should have the ability to describe any form of data to satisfy all the potential need.

2) The represented data should be easily accessed by other organizations and its supported software, such as parsers or query APIs, should be reusable (syntactic operability).

3) It should have definitions for mappings between terms in the data (semantic interoperability) [5].

3.1. Using XML

XML is competent to describe any data by allowing users to create their own tags and decide structures for the document, thus satisfying the first condition. It also meets the second requirement because XML parsers can parse any XML data and they are reusable. However, XML does not ensure the semantics of data. It mainly concentrates on document's grammar and does not provide the relationship between data [1]. For example, we need to exchange a piece of simple information, a description (a name) of the product. It is depicted in the form of a model in figure 1.

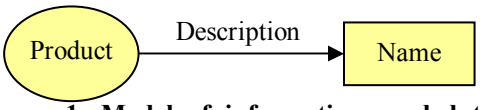

Figure 1. Model of information needed to be exchanged, the relationship *Description* between *Product* and *Name.*

From this model, a DTD or an XML schema can be created. In this example we use DTD. Since DTD just defines structures for the XML document, there are several DTDs and the corresponding XML expressions created for this model (Table 1). Because it is impossible to recognize the relationship between *Product* and *Name* from DTD, it is hard to rebuild the model from this DTD. This is not a big problem if the communication is one-to-one between two parties since they can agree in a DTD before exchanging information. However, the communication on the web enables multiple partners and exchanged information often changes during time. Every change in information requires changes in DTD structure and corresponding XML document, which is costly. Moreover, since information can be described in different ways by a DTD, every communication can choose difference kinds of DTD. Therefore, it is difficult to change the structure for all of them [5]. What we need is a common description of a resource.

Table 1. Possibility DTDs and XML encodings

Encoding DTD	Example XML data
<!ELEMENT Product (Desc)> <!ATTLIST Prod_id ID #REQUIRED> <!ELEMENT Desc (Name)> <!ATTLIST Name id ID #IMPLIED>	<Prod_id="X"> <Desc> <Name id="Y"> </Desc></Product>
<!ELEMENT Description (Prod, Name)> <!ELEMENT Prod (#CDATA)> <!ELEMENT Name (#CDATA)>	<Description> <Prod>X</Prod> <Name>Y</Name> </Description>
<!ELEMENT Product (id, Description)> <!ELEMENT id (#CDATA)> <!ELEMENT Description (Name)> <!ELEMENT Name (id)>	<Product><id>X</id> <Description> <Name><id> Y </id> </Name> </Description> </Product>

3.2. Using RDF

RDF satisfies all the requirement of representing knowledge. It identifies items by using URIs and describes resources in terms of subject, predicate, and object. This enables RDF to represent any kind of data, satisfying the first condition. The fact that RDF can be passed by various independent and reusable parsers ensures the second requirement[1]. For the last requirement, RDF surpasses XML. With the demonstrations like natural language, RDF can easily describe the model (defining objects and their relationships) of information, so it does not need to translate the model into DTD and then the DTD into XML. As RDF descriptions are independent of XML, a change in XML syntax does not require a change in RDF model [5].

However, RDF only provides simple descriptions about resources and their values, so to depict classes of resources or specific properties of these resources RDF schema is used. It is mentioned in the second last paragraph of section 1.

4. Procedure description

Our procedure has two main steps. The first one presents the strategy to derive classes and properties from XML DTD. The second uses this strategy to scan the XML document and produce RDF statements.

4.1. Extraction of classes and properties

In this stage, we create the collection of classes and properties from the given DTD as an input. This collection will be used to model data in the next step. The general idea of this step is as follows:

- Element being declared by <!DOCTYPE> is the root-class of document.
- For each sub-element (elements in brackets or following the first element), we decide whether they are subclasses or properties of the class.
- For data type definition of every element, we can predict the format of data in XML.

A DTD is made up of three main building blocks: ELEMENT, ATTLIST, and ENTITY. ELEMENT is the main building block of XML documents. In the DTD, XML elements are declared with an ELEMENT. An element definition has the following syntax:

<!ELEMENT element-name (element-content)>

element-content may be EMPTY, or data type, or sequences of children. Because ELEMENT is used to describe elements of a document and each element can contain children elements [12], the function of these elements is like a class in a structure program, therefore, we will treat *element-name* as a class-name in our procedure. If the *element-content* contains sequences of children, our procedure considers these children as subclass of the *element-name*.

ATTLIST provides extra information about elements. Its function is to describe the property of a class, so we consider it as a class property. Following is a general syntax of an ATTLIST element:

<!ATTLIST element-name attribute-name attribute-type default-value>

element-name is the name of element (class) for which we declare an attribute. *Attribute-name* is a name of the attribute we want to declare, in our procedure it is a name of the property. *Attribute-type* is a data type and *default-value* specifies default value of the attribute.

Finally, ENTITY is used to define a shortcut for a common text in XML. Its syntax is as follows:

<!ENTITY name definition>

In this case, *name* is the name of ENTITY and *definition* is its definition. For example, `<!ENTITY today "July 22, 2007">`. In XML document, it is referred between "&" and ";", such as `<DATE>&today;</DATE>`. Because of the function in the DTD, our procedure handles *name* as a variable and *definition* as its value. When our procedure meets this variable in the document, its value will be called.

Besides these there are some declarations in DTD, such as CDATA, PCDATA, #REQUIRED, #IMPLIED, etc. Their purpose is to declare the data type or the displaying conditions of elements or attributes in the document [12]. Our procedure is not concerned about these declarations because when it finds the values of a class or property, it takes whole values without parsing them, unless these values are declared as an ENTITY.

[1] http://infolab.stanford.edu/~melnik/rdf/api.html

4.2. XML transformation

After deriving classes and properties from a DTD, we continue to examine the valid XML document. The result is RDF triples to interpret these XML data. The URI of the XML document will be the subject of the first statement. The algorithm starts traversing from the beginning of the XML document and finishes when it meets the close tag of root element. The comments are skipped during the transformation process.

For every tag in the XML document, we verify whether it is a name of a class or a property. If it does not match with any class or property in our database, we skip it and continue to the next tag. Furthermore, all texts describing between quotation marks in tags or between open and close tags are values of a class or a property. Based on this, our procedure decides what RDF statements should be created.

1. If the tag matches with a class, following three cases should be considered:

a) If this is the root-class, create the first statements:

URI of document	rdfs: Resource	root-class
Root-class	rdf: hasClass	class-name

These statements are used once in our procedure. Since in an XML document, there is only one root-class and all other classes are its children, when we meet the root-class we use *rdfs: Resource* to connect the resource of XML document (URI) to the root-class of the document. *rdf: hasClass* is defined to connect two classes. In this case it describes that *root-lass* has a class-child, *class-name*.

b) This class can be a child of root-class or another class. If the previous statement is unfinished (statement with only two elements: subject and predicate are filled, the object is empty), we complete this statement by supplementing the parent class of considered class in the object and add one more statement to describe this class.

		parent-class-name
parent-class-name	rdf:hasClass	Class-name

c) Create the new statement (simple case of b):

parent-class-name	rdf: hasClass	Class-name

It means when we find out a class, we have to specify its parent.

2. If the element matches with a property, we verify the class this property describes and predict the value of this property. However, because our RDF statements are sometimes unfinished, we consider two cases:

a) If the previous statement is unfinished, complete it with the name of class this property belongs to, and create new unfinished statement:

		Class-name
Class-name	rdf: Property	property-name
property-name	rdf: value	

rdf: Property used to describe an attribute is a property of a class (class-name), and *rdf: value* is declared for the value of this property.

b) It is a simple case of a, we also describe which class this property depicts and create an unfinished statement:

Class-name	rdf: Property	property-name
property-name	rdf: value	

3. If it does not match a class or property, we check whether it is a value of a class/property or not. It is a value if it is placed between quotation marks or between the open and the close tag. Furthermore, we have to verify whether it is a declaration of an ENTITY or not. If it is a description of ENTITY, we replace this value by its definition. Therefore, we only consider that this value belongs to a class or property.

a) If pervious statement is unfinished, it is surely a value of a property. Because in previous statements, only statements describe for a property is always unfinished statements, we add this value to this empty column:

		value

b) Else, so this value is belong to a class. We describe which class has this value by following statement:

Class-name	rdf: value	value

4.3. Example

In order to illustrate for our procedure, we use sample files at http://www.vervet.com/. This website supports free download of the XML editor, XMLPro. After installing this software, we have several data samples. We choose files describing product, because these kinds of files are so popular on the web as well as in the electronic business. A DTD file, “Catalog.dtd”, defines the structure for the XML file as following:

```
<!DOCTYPE Catalog [
<!ELEMENT Catalog (Product+)>
<!ELEMENT Product (Specifications+,
Options?, Price+, Notes?)>
<!ELEMENT Specifications (#PCDATA)>
<!ELEMENT Options (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Notes (#PCDATA)>
<!ATTLIST Product Name CDATA #IMPLIED>
<!ATTLIST Category (HandTool|Table|Shop-
Professional)"HandTool">
<!ATTLIST Partnum CDATA #IMPLIED>
<!ATTLIST Plant (Pittsburgh|Milwaukee|
Chicago)"Chicago"><!ATTLIST Inventory
(InStock|Backordered| Discont)"InStock">
```

```
<!ATTLIST Specifications Weight CDATA
#IMPLIED><!ATTLIST Power CDATA #IMPLIED>
<!ATTLIST Options Finish (Metal|Polished|
Matte) "Matte">
<!ATTLIST Options Adapter (Included|
Optional|NotApplicable)"Included">
<!ATTLIST Options Case (HardShell| Soft|
NotApplicable) "HardShell">
<!ATTLIST Price MSRP CDATA #IMPLIED>
<!ATTLIST Price Wholesale CDATA #IMPLIED>
<!ATTLIST Price Street CDATA #IMPLIED>
<!ATTLIST Price Shipping CDATA #IMPLIED>
]>
```

Using rules in section 4.1, we derive classes and their corresponding properties as below:

Root class: Catalog, Subclass: Product (Properties: name, category, partnum, plant, inventory)

Subclass of Product: Specification, Options, Price and Notes.

Properties of Specification: weight, power

Properties of Options: finish, adapter, case

Properties of Price: MSRP, wholesale, street, shipping

After having the set of classes and properties from the previous step, we scan the XML file, "Catalog.xml", to produce RDF statements by using algorithm in section 4.2. Because the file is quite long with four products but they are the same in structure, we just pick the first product to analyze. Following is XML file with the first product:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Catalog SYSTEM "catalog.dtd">
<Catalog>
<Product Name="Speed Drill Pro" Partnum=
"123XYZ" Plant="Pittsburgh" Inventory=
"Backordered" Category="Shop-Professional">
 <Specifications Weight="8lbs."
Power="120v"/><Options Adapter="Included"
Case="HardShell"/>
 <Price MSRP="$149.95" Wholesale="$99.95"
Street="$129.95" Shipping="$15.00"/>
<Notes>Professional Version of the top
selling from the consumer line.</Notes>
</Product>...</Catalog>
```

The above XML document is interpreted to RDF triples in the table 2.

Table 2: RDF statements from the XML data

Subject	Predicate	Object
http://www.vervet.com	rdfs: Resource	Catalog
Catalog	rdf:hasClass	Product
Product	rdf: Property	Name
Name	rdf: value	"Speed Drill Pro"
Product	rdf: Property	Partnum
Partnum	rdf: value	"123XYZ"
Product	rdf: Property	Plant
Plant	rdf: value	"Pittsburgh"
Product	rdf: Property	Inventory
Inventory	rdf: value	"Backordered"
Category	rdf: value	"Shop-Professional"
Product	rdf: hasClass	Specifications
Specifications	rdf: Property	Weight
Weight	rdf: value	"8lbs"
Specifications	rdf: Property	Power
Power	rdf: value	"120v"
Product	rdf: hasClass	Options
Options	rdf: Property	Adapter
Adapter	rdf: value	"Included"
Options	rdf: Property	Case
Case	rdf: value	"HardShell"
Product	rdf: hasClass	Price
Price	rdfs:domain	MSRP
MSRP	rdf:value	"$149.95"
Price	rdfs:domain	Wholesale
Wholesale	rdf:value	"$99.95"
Price	rdfs:domain	Street
Street	rdf:value	"$129.95"
Price	rdfs:domain	Shipping
Shipping	rdf:value	"$15.00"
Product	rdf: hasClass	Notes
Notes	rdf: value	"Professional Version of the top selling from the consumer line."

These above RDF statements keep the structure as well as the relationship of every element in XML. Moreover, they represent the meaning of the data as well as the relationship between data. For example, *Name* is a property of *Product* and its value is *Speed Drill Pro*. The graph description of the above RDF triples is presented in the figure 2.

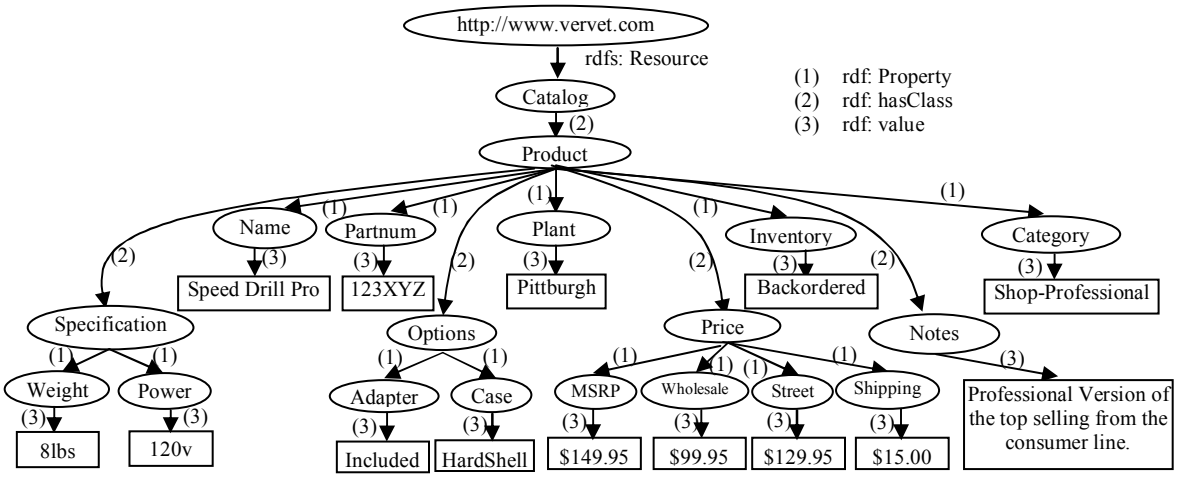

Figure 2. The result RDF statements are presented in the graph

5. Conclusions

In this paper we have proposed a procedure to transform valid XML documents into RDF statements by using RDF schema vocabularies. Our proposed method enables lots of XML data available in Internet to be used in the next generation of web, the Semantic Web. We choose valid XML documents to translate because most of the XML documents, used in the current web, are in valid forms. Moreover, based on their DTDs, we can anticipate the structure of the XML documents as well as the relationship of elements in these documents. Furthermore, DTDs helps us to decide which labels in XML documents are important and what is their role as class or property. Based on their role and relationship, our procedure interprets XML data into an RDF model.

Our procedure outperforms the existing methods due to the following three reasons. Firstly, it transforms all the elements of an XML document into RDF retaining the original structure and the meaning of the document. Secondly, elements in XML are clarified in classes or properties based on their definition in DTD, making the result independent from the users' point of view. Finally, languages used in our procedure do their jobs as their original functions. DTD is used for defining XML structure, XML for describing data, RDF for providing triple statements about data, and RDF schema for supporting vocabularies to describe the relationship among data. If this procedure is executed, a large amount of the XML data will be interpreted into RDF statements which are useful for the Semantic Web.

Acknowledgment

This work is supported by Ubiquitous Computing and Network (UCN) grants Z1500-0603-0004, the Ministry of Information and Communication (MIC) 21th Century Frontier R&D Program in Korea.

Preferences

[1] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks, "The Semantic Web: The roles of XML and RDF", *IEEE*, 2000.

[2] Tim Berners-Lee, James Handler, and Ora Lassila, "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", *Scientific American*, 2001.

[3] Michael Erdmann, Rudi Studer, "How to structure and access XML documents with Ontologies", April 2000.

[4] Tim Berners-Lee, "Semantic Web – A guide to the future XML web services and knowledge management", *Weaving the Web*, Harper San Francisco, 1999.

[5] Stefan Decker, Frank van harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michel Klein, and Sergey Melnik, "The Semantic Web – on the respective roles of XML and RDF", 2000.

[6] Sergey Melnik, "Bridging the gap between RDF and XML", Dec 1999.

[7] B.Amann, I.Fundulaki, M.Scholl, C.Beeri, and A-M.Vercoustre, "Mapping XML fragments to community Web ontologies", *Fourth International Workshop on the Web and Databases* (WebDDB'2001).

[8] Michel Klein, "Interpreting XML via an RDF Schema", *Database and Expert Systems Applications*, 2002.

[9] Tim Berners_Lee , "A strawman unstriped syntax for RDF in XML", W3C, March 2007.

[10] Jonathan Boden, "Simplified XML syntax for RDF", June 2001, available at: http://www.openhealth.org/RDF/RDFSurfaceSyntax.html

[11] Mark H.Butler, John Gilbert, Andy Seaborne, and Kevin Smarthers, "Data conversion, extraction and record linkage using XML and RDF tools in Project SMILE", 2004.

[12] Refsnes Data, "Introduction to DTD", 1999-2007, at: http://www.w3schools.com/dtd/dtd_intro.asp

[13] Michel Klein, Dieter Fensel, Frank van Harmelen, and Ian Horrocks, "The relation between ontologies and XML Schemas", 2001.

[14] Ivan Herman, "Semantic Web", *W3C*, available at: http://www.w3.org/2001/sw/

[15] Peter Patel-Schneider, and Jérôme Siméon, "The Yin/Yang Web: XML syntax and RDF Semantics", *11th International WWW conference,* Hawaii, 2002.

[16] T.Bray, J. Paoli, and C.M. Sperberg-McQueen, "eXtensible Markup Language (XML) 1.0", W3C Recommendation, Feb.1998; available at http://www.w3.org/TR/REC-xml

[17] Dan Brickley, R.V. Guha, and Brian McBride, "RDF Vocabulary description language 1.0: RDF schema", *W3C*, 10 Feb 2004, available at http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[18] Tim Berners_Lee, "Why RDF model is different from the XML model", Sep 1998, available at: http://www.w3.org/DesignIssues/RDF-XML.html.

[19] Tim Furche, François Bry, and Oliver Bolzer, "XML Perspectives on RDF querying: Towards integrated access to data and metadata on the Web", 2005.

[20] Pronab Ganguly, Fethi A. Rabhi and Pradeep K.Ray, "Bridging Semantic Gap", *Third Asian Pacific conference on Pattern languages of Program*, 2002.

[21] Frank Manola, Eric Miller, "RDF Primer", *W3C Recommendation,* February 2004, available at: http://www.w3c.org/TR/REC-rdf-syntax/

[22] Peter Patel-Schneider and Jérôme Siméon, "The Yin/Yang Web: XML syntax and RDF Semantics", *The Eleventh International World Wide Web conference,* Honolulu, Hawaii, May 2002.