Describing Web Resources in RDF

Bojan Furlan

A Semantic Web Primer,
G. Antoniou, F. van Harmelen

Drawbacks of XML

- XML is a universal metalanguage for defining markup
- It provides a uniform framework for interchange of data and metadata between applications
- However, XML does not provide any means of talking about the semantics (meaning) of data
- E.g., there is no intended meaning associated with the nesting of tags
  - It is up to each application to interpret the nesting.

Nesting of Tags in XML

David Billington is a lecturer of Discrete Maths

```
<course name="Discrete Maths">
        <lecturer>David Billington</lecturer>
</course>
<lecturer name="David Billington">
        <teaches>Discrete Maths</teaches>
</lecturer>
```

Opposite nesting, same information!

Basic Ideas of RDF

- Basic building block: **object-attribute-value** triple
  - It is called a statement
  - Sentence about Billington is such a statement
- RDF has been given a syntax in XML
  - This syntax inherits the benefits of XML
  - Other syntactic representations of RDF possible

Basic Ideas of RDF (2)

- The fundamental concepts of RDF are:
  - resources
  - properties
  - statements

Resources

- We can think of a resource as an object, a “thing” we want to talk about
  - E.g. authors, books, publishers, places, people, hotels
- Every resource has a URI, a Universal Resource Identifier

Properties

- Properties are a special kind of resources
- They describe relations between resources
  - E.g. “written by”, “age”, “title”, etc.
- Properties are also identified by URIs
- Advantages of using URIs:
  - A global, worldwide, unique naming scheme
  - Reduces the homonym problem of distributed data representation

Statements

- Statements assert the properties of resources
- A statement is an object-attribute-value triple
  - It consists of a resource, a property, and a value
- Values can be resources or **literals**
  - Literals are atomic values (strings)

Three Views of a Statement

- A triple
- A piece of a graph
- A piece of XML code

Thus an RDF document can be viewed as:

- A set of triples
- A graph (semantic net)
- An XML document

Statements as Triples

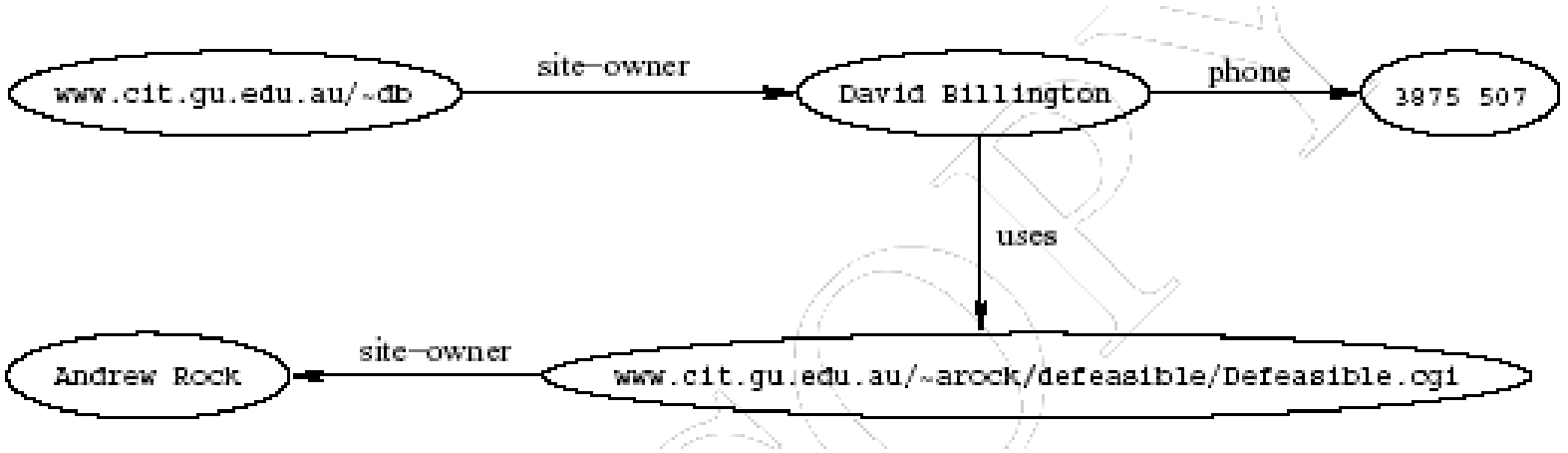
(http://www.cit.gu.edu.au/~db,
http://www.mydomain.org/site-owner,
#David Billington)

- The triple (x,P,y) can be considered as a logical formula P(x,y)
  - Binary predicate P relates object x to object y
  - RDF offers only binary predicates (properties)

Statements as Directed Graph

- A directed graph with labeled nodes and arcs
  - **from** the resource (the **subject** of the statement)
  - **to** the value (the **object** of the statement)
- Known in AI as a *semantic net*
- The value of a statement may be a resource
  - It may be linked to other resources

A Set of Triples as a Semantic Net



Statements in XML Syntax

- Graphs are a powerful tool for human understanding **but**
- The Semantic Web vision requires machine-accessible and machine-processable representations
- There is a 3rd representation based on XML
  - But XML is not a part of the RDF data model
  - E.g. serialisation of XML is irrelevant for RDF

Statements in XML (2)

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">

      <rdf:Description
       rdf:about="http://www.cit.gu.edu.au/~db">
          <mydomain:site-owner
            rdf:resource="#David Billington"/>
      </rdf:Description>
</rdf:RDF>
```

Statements in XML (3)

- An RDF document is represented by an XML element with the tag **rdf:RDF**
- The content of this element is a number of descriptions, which use **rdf:Description** tags.
- Every description makes a statement about a resource, identified in 3 ways:
  - an **about** attribute, referencing an existing resource
  - an **ID** attribute, creating a new resource
  - without a name, creating an anonymous resource

Statements in XML (4)

- The **rdf:Description** element makes a statement about the resource **http://www.cit.gu.edu.au/~db**
- Within the description
  - the property is used as a tag
  - the content is the value of the property

Reification

- In RDF it is possible to make statements about statements
  - **Grigoris believes that David Billington is the creator of http://www.cit.gu.edu.au/~db**
- Such statements can be used to describe belief or trust in other statements
- The solution is to assign a unique identifier to each statement
  - It can be used to refer to the statement

Reification (2)

- Introduce an auxiliary object (e.g. **belief1**)
- relate it to each of the 3 parts of the original statement through the properties **subject**, **predicate** and **object**
- In the preceding example
  - subject of **belief1** is **David Billington**
  - predicate of **belief1** is creator
  - object of **belief1** is **http://www.cit.gu.edu.au/~db**

Data Types

- Data types are used in programming languages to allow interpretation
- In RDF, typed literals are used, if necessary

(#David Billington,

http://www.mydomain.org/age,

“27”^^http://www.w3.org/2001/XMLSchema#integer)

Data Types (2)

- ^^-notation indicates the type of a literal
- In practice, the most widely used data typing scheme will be the one by XML Schema
  - But the use of **any** externally defined data typing scheme is allowed in RDF documents
- XML Schema predefines a large range of data types
  - E.g. Booleans, integers, floating-point numbers, times, dates, etc.

Basic Ideas of RDF Schema

- RDF is a universal language that lets users describe resources in their own vocabularies
  - RDF does not assume, nor does it define semantics of any particular application domain
- The user can do so in RDF Schema using:
  - Classes and Properties
  - Class Hierarchies and Inheritance
  - Property Hierarchies

Classes and their Instances

- We must distinguish between
  - Concrete “things” (individual objects) in the domain: Discrete Maths, David Billington etc.
  - Sets of individuals sharing properties called classes: lecturers, students, courses etc.
- Individual objects that belong to a class are referred to as instances of that class
- The relationship between instances and classes in RDF is through **rdf:type**

Why Classes are Useful

- Impose restrictions on what can be stated in an RDF document using the schema
  - As in programming languages
    - User.LogIn(); and not String.LogIn();
  - Disallow nonsense from being stated

Nonsensical Statements disallowed through the Use of Classes

- Discrete Maths is taught by Concrete Maths
  - We want courses to be taught by lecturers only
  - Restriction on values of the property “is taught by” (**range restriction**)
- Room MZH5760 is taught by David Billington
  - Only courses can be taught
  - This imposes a restriction on the objects to which the property can be applied (**domain restriction**)

Class Hierarchies

- Classes can be organised in hierarchies
  - A is a subclass of B if every instance of A is also an instance of B
  - Then B is a superclass of A
- A subclass graph need not be a tree
- A class may have multiple superclasses

Class Hierarchy Example

Inheritance in Class Hierarchies

- Range restriction: **Courses must be taught by academic staff members only**
- Michael Maher is a professor
- He inherits the ability to teach from the class of academic staff members
- This is done in RDF Schema by fixing the semantics of “is a subclass of”
  - It is not up to an application (RDF processing software) to interpret “is a subclass of”

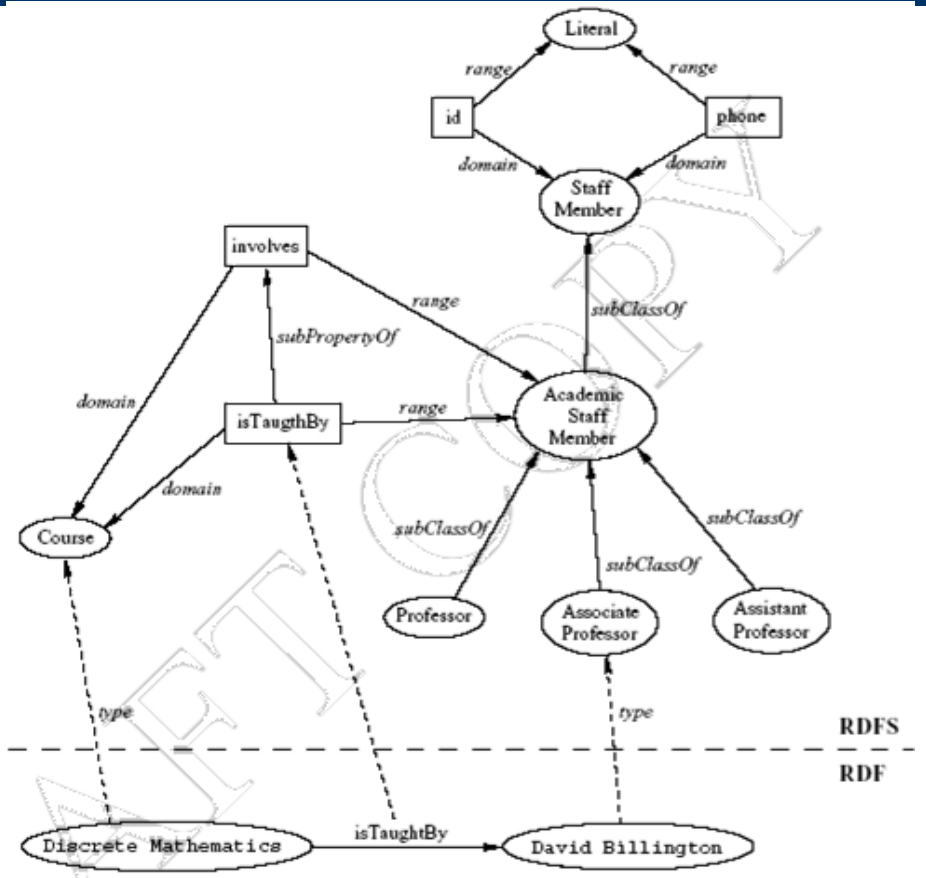
Property Hierarchies

- Hierarchical relationships for properties
  - E.g., “is taught by” is a subproperty of “involves”
  - If a course C is taught by an academic staff member A, then C also involves A
- The converse is not necessarily true
  - E.g., A may be the teacher of the course C, or
  - a tutor who marks student homework but does not teach C
- P is a subproperty of Q, if Q(x,y) is true whenever P(x,y) is true

RDF Layer vs RDF Schema Layer

- Discrete Mathematics is taught by David Billington
- The schema is itself written in a formal language, RDF Schema, that can express its ingredients:
  - subClassOf, Class, Property, subPropertyOf, Resource, etc.

RDF Layer vs RDF Schema Layer (2)



Why an RDF Query Language? Different XML Representations

- XML at a lower level of abstraction than RDF
- There are various ways of syntactically representing an RDF statement in XML
- Thus we would require several XPath queries, e.g.
  - **//uni:lecturer/uni:title** if **uni:title** element
  - **//uni:lecturer/@uni:title** if **uni:title** attribute
  - Both XML representations equivalent!

SPARQL Basic Queries

- SPARQL is based on matching graph patterns
- The simplest graph pattern is the triple pattern :
  - like an RDF triple, but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions
- Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern

Examples

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
    ?c rdf:type rdfs:Class .
}
```

- Retrieves all triple patterns, where:

-the property is rdf:type

-the object is rdfs:Class

- Which means that it retrieves all classes

Examples (2)

- Get all instances of a particular class (e.g. course) :

(declaration of rdf, rdfs prefixes omitted for brevity)

```
PREFIX uni: <http://www.mydomain.org/uni-ns#>
SELECT ?i
WHERE
{
    ?i rdf:type uni:course .
}
```

Using select-from-where

- As in SQL, SPARQL queries have a SELECT-FROM-WHERE structure:
  - **SELECT** specifies the projection: the number and order of retrieved data
  - **FROM** is used to specify the source being queried (optional)
  - **WHERE** imposes constraints on possible solutions in the form of graph pattern templates and boolean constraints
- Retrieve all phone numbers of staff members:

```
SELECT ?x ?y
WHERE
{ ?x uni:phone ?y .}
```

- Here **?x** and **?y** are variables, and **?x uni:phone ?y** represents a resource-property-value triple pattern

Implicit Join

- Retrieve all lecturers and their phone numbers:

```
SELECT ?x ?y
WHERE
{ ?x rdf:type uni:Lecturer ;
        uni:phone ?y . }
```

- Implicit join: We restrict the second pattern only to those triples, the resource of which is in the variable **?x**
  - Here we use a syntax shorcut as well: a semicolon indicates that the following triple shares its subject with the previous one

Implicit join (2)

- The previous query is equivalent to writing:

```
SELECT ?x ?y
WHERE
{
   ?x rdf:type uni:Lecturer .
   ?x uni:phone ?y .
}
```

Explicit Join

- Retrieve the name of all courses taught by the lecturer with ID 949352

```
SELECT ?n
WHERE
{
   ?x rdf:type uni:Course ;
         uni:isTaughtBy :949352 .
   ?c uni:courseName ?n .
   FILTER (?c = ?x) .
}
```

Optional Patterns

```
<uni:lecturer rdf:about="949352">
    <uni:name>Grigoris Antoniou</uni:name>
</uni:lecturer>
<uni:professor rdf:about="94318">
    <uni:name>David Billington</uni:name>
    <uni:email>david@work.example.org</uni:email>
</uni:professor>
```

- For one lecturer it only lists the name
- For the other it also lists the email address

Optional Patterns (2)

- All lecturers and their email addresses:

```
SELECT ?name ?email
WHERE
{  ?x rdf:type uni:Lecturer ;
        uni:name ?name ;
        uni:email ?email .
}
```

Optional Patterns (3)

- **The result of the previous query would be:**

?name	?email
David Billington	david@work.example.org

- **Grigoris Antoniou is listed as a lecturer, but he has no e-mail address**

Optional Patterns (4)

- As a solution we can adapt the query to use an optional pattern:

```
SELECT ?name ?email
WHERE
{  ?x rdf:type uni:Lecturer ;
        uni:name ?name .
        OPTIONAL { x? uni:email ?email }
}
```

Optional Patterns (5)

- The meaning is roughly “give us the names of lecturers, and if known also their e-mail address”
- The result looks like this:

?name	?email
Grigoris Antoniou	
David Billington	david@work.example.org

SPARQL Tutorials and examples

- **Sparqlette -** A SPARQL demo query service
  - http://www.wasab.dk/morten/2005/04/sparqlette/
- **SPARQL Tutorial**
  - http://jena.sourceforge.net/ARQ/Tutorial/
- **Anatomy of a simple SPARQL query**
  - https://www.ibm.com/developerworks/xml/library/j-sparql/