3 Describing Web Resources in RDF

3.1 Introduction

XML is a universal metalanguage for defining markup. It provides a uniform framework, and a set of tools like parsers, for interchange of data and metadata between applications. However, XML does not provide any means of talking about the semantics (meaning) of data. For example, there is no intended meaning associated with the nesting of tags; it is up to each application to interpret the nesting. Let us illustrate this point using an example. Suppose we want to express the following fact:

David Billington is a lecturer of Discrete Mathematics.

There are various ways of representing this sentence in XML. Three possibilities are

```xml
<course name="Discrete Mathematics">
  <lecturer>David Billington</lecturer>
</course>

<lecturer name="David Billington">
  <teaches>Discrete Mathematics</teaches>
</lecturer>

<teachingOffering>
  <lecturer>David Billington</lecturer>
  <course>Discrete Mathematics</course>
</teachingOffering>
```
Note that the first two formalizations include essentially an opposite nesting although they represent the same information. So there is no standard way of assigning meaning to tag nesting.

Although often called a “language” (and we commit this sin ourselves in this book), RDF is essentially a data-model. Its basic building block is an object-attribute-value triple, called a statement. The preceding sentence about Billington is such a statement. Of course, an abstract data model needs a concrete syntax in order to be represented and transmitted, and RDF has been given a syntax in XML. As a result, it inherits the benefits associated with XML. However, it is important to understand that other syntactic representations of RDF, not based on XML, are also possible; XML-based syntax is not a necessary component of the RDF model.

RDF is domain-independent in that no assumptions about a particular domain of use are made. It is up to users to define their own terminology in a schema language called RDF Schema (RDFS). The name RDF Schema is now widely regarded as an unfortunate choice. It suggests that RDF Schema has a similar relation to RDF as XML Schema has to XML, but in fact this is not the case. XML Schema constrains the structure of XML documents, whereas RDF Schema defines the vocabulary used in RDF data models. In RDFS we can define the vocabulary, specify which properties apply to which kinds of objects and what values they can take, and describe the relationships between objects. For example, we can write

*Lecturer* is a subclass of *academic staff member*.

This sentence means that all lecturers are also academic staff members. It is important to understand that there is an intended meaning associated with “is a subclass of”. It is not up to the application to interpret this term; its intended meaning must be respected by all RDF processing software. Through fixing the semantics of certain ingredients, RDF/RDFS enables us to model particular domains.

We illustrate the importance of RDF Schema with an example. Consider the following XML elements:

```xml
<academicStaffMember>Grigoris Antoniou</academicStaffMember>

<professor>Michael Maher</professor>

<course name="Discrete Mathematics">
  <isTaughtBy>David Billington</isTaughtBy>
</course>
```
Suppose we want to collect all academic staff members. A path expression in Xpath might be

```
//academicStaffMember
```

The result is only Grigoris Antoniou. While correct from the XML viewpoint, this answer is semantically unsatisfactory. Human readers would have also included Michael Maher and David Billington in the answer because

- All professors are academic staff members (that is, professor is a subclass of academicStaffMember).
- Courses are only taught by academic staff members.

This kind of information makes use of the semantic model of the particular domain, and cannot be represented in XML or in RDF but is typical of knowledge written in RDF Schema. Thus RDFS makes semantic information machine-accessible, in accordance with the Semantic Web vision.

In this chapter, sections 3.2 and 3.3 discuss RDF: the basic ideas of RDF and its XML-based syntax, and sections 3.4 and 3.5 introduce the basic concepts and the language of RDF Schema.

Section 3.6 shows the definition of some elements of the namespaces of RDF and RDF Schema. Section 3.7 presents an axiomatic semantics for RDF and RDFS. This semantics uses predicate logic and formalizes the intuitive meaning of the modeling primitives of the languages.

Section 3.8 provides a direct semantics based on inference rules, and section 3.9 is devoted to the querying of RDF/RDFS documents using RQL.

### 3.2 RDF: Basic Ideas

The fundamental concepts of RDF are resources, properties and statements.

#### 3.2.1 Resources

We can think of a resource as an object, a “thing” we want to talk about. Resources may be authors, books, publishers, places, people, hotels, rooms, search queries, and so on. Every resource has a URI, a Universal Resource Identifier. A URI can be a URL (Unified Resource Locator, or Web address) or some other kind of unique identifier; note that an identifier does not necessarily enable access to a resource. URI schemes have been defined not only
for web-locations but also for such diverse objects as telephone numbers, ISBN numbers and geographic locations. There has been a long discussion about the nature of URIs, even touching philosophical questions (for example, what is an appropriate unique identifier for a person?), but we will not go into detail here. In general, we assume that a URI is the identifier of a Web resource.

### 3.2.2 Properties

Properties are a special kind of resources; they describe relations between resources, for example “written by”, “age”, “title”, and so on. Properties in RDF are also identified by URIs (and in practice by URLs). This idea of using URIs to identify “things” and the relations between is quite important. This choice gives us in one stroke a global, worldwide, unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed datarepresentation until now.

### 3.2.3 Statements

Statements assert the properties of resources. A statement is an object-attribute-value triple, consisting of a resource, a property, and a value. Values can either be resources or literals. Literals are atomic values (strings), the structure of which we do not discuss further.

### 3.2.4 Three Views of a Statement

An example of a statement is

> David Billington is the owner of the Web page  

The simplest way of interpreting this statement is to use the definition and consider the triple

> (“David Billington”, http://www.mydomain.org/site-owner,  

We can think of this triple \( (x, P, y) \) as a logical formula \( P(x, y) \), where the binary predicate \( P \) relates the object \( x \) to the object \( y \). In fact, RDF offers only binary predicates (properties). Note that the property “site-owner” and one of
3.2 RDF: Basic Ideas

the two objects are identified by URLs, whereas the other object is simply identified by a string.

A second view is graph-based. Figure 3.1 shows the graph corresponding to the preceding statement. It is a directed graph with labeled nodes and arcs; the arcs are directed from the resource (the subject of the statement) to the value (the object of the statement). This kind of graph is known in the Artificial Intelligence community as a semantic net.

As we already said, the value of a statement may be a resource. Therefore, it may be linked to other resources. Consider the following triples:


The graphic representation is found in figure 3.2.

Graphs are a powerful tool for human understanding. But the Semantic Web vision requires machine-accessible and machine-processable representations.
Therefore, there is a third representation possibility based on XML. According to this possibility, 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, which is identified in one of three different ways:

- an about attribute, referencing an existing resource
- an ID attribute, creating a new resource
- without a name, creating an anonymous resource

We will discuss the XML-based syntax of RDF in section 3.3, here we just show the representation of our first statement:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<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>
      David Billington
    </mydomain:site-owner>
  </rdf:Description>
</rdf:RDF>
```

The first line specifies that we are using XML. In the following examples we omit this line, but keep in mind that it must be present in any RDF document with XML-based syntax.

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, and the content is the value of the property.

The descriptions are given in a certain order, in other words the XML syntax imposes a serialization. The order of descriptions (or resources) is not significant according to the abstract model of RDF. This again shows that the graph model is the real data model of RDF and that XML is just a possible serial representation of the graph.
3.2.5 **Reifficication**

In RDF it is possible to make statements about statements, such as

Grigoris believes that David Billington is the creator of the Web page http://www.cit.gu.edu.au/~db.

This kind of statement can be used to describe belief or trust in other statements, which is important in some kinds of applications. The solution is to assign a unique identifier to each statement, which can be used to refer to the statement. RDF allows this using a *reification mechanism* (see section 3.3.6).

The key idea is to introduce an auxiliary object, say, *belief1*, and relate it to each of the three parts of the original statement through the properties *subject*, *predicate* and *object*. In the preceding example the subject of *belief1* would be *David Billington*, the predicate would be *creator*, and the object http://www.cit.gu.edu.au/~db. Note that this rather cumbersome approach is necessary because there are only triples in RDF; therefore we cannot add an identifier directly to a triple (then it would be a quadruple).

3.2.6 **Data Types**

Consider the telephone number “3875507”. A program reading this RDF data model cannot know if the literal “3875507” is to be interpreted as an integer (an object on which it would make sense to, say, divide it by 17) or as a string, or indeed if it is a integer, whether it is in decimal or octal representation. A program can only know how to interpret this resource if the application is explicitly given the information that the literal is intended to represent a number, and which number the literal is supposed to represent. The common practice in programming languages or database systems is to provide this kind of information by associating a data type with the literal, in this case, a data type like decimal or integer. In RDF, *typed literals* are used to provide this kind of information.

Using a typed literal, we could describe David Billington’s age as being the integer number 27 using the triple:

```
(“David Billington”, http://www.mydomain.org/age, 
“27”^^http://www.w3.org/2001/XMLSchema#integer )
```

This example shows two things: the use of the `^^`-notation to indicate the type of a literal,1 and the use of data types that are predefined by XML.

---

1. This notation will take a different form in the XML-based syntax described in section 3.3.
3.2.7 A Critical View of RDF

We have already pointed out that RDF uses only binary properties. This restriction seems quite serious because often we use predicates with more than two arguments. Luckily, such predicates can be simulated by a number of binary predicates. We illustrate this technique for a predicate referee with three arguments. The intuitive meaning of referee(X, Y, Z) is:

X is the referee in a chess game between players Y and Z.

We now introduce a new auxiliary resource chessGame and the binary predicates ref, player1, and player2. Then we can represent referee(X, Y, Z) as follows:

ref(chessGame, X)
player1(chessGame, Y)
player2(chessGame, Z)

The graphic representation is shown in figure 3.3. Although the solution is sound, the problem remains that the original predicate with three arguments was simpler and more natural.
Another problem with RDF has to do with the handling of properties. As mentioned, properties are special kinds of resources. Therefore, properties themselves can be used as the object in an object-attribute-value triple (statement). While this possibility offers flexibility, it is rather unusual for modeling languages, and can be confusing for modelers.

Also, the reification mechanism is quite powerful and appears misplaced in a simple language like RDF. Making statements about statements introduces a level of complexity that is not necessary for a basic layer of the Semantic Web. Instead, it would have appeared more natural to include it in more powerful layers, which provide richer representational capabilities.

Finally, the XML-based syntax of RDF is well suited for machine processing but is not particularly human-friendly.

In summary, RDF has its idiosyncrasies and is not an optimal modeling language. However, we have to live with the fact that it is already a de facto standard. In the history of technology, often the better technology was not adopted. For example, the video system VHS was probably the technically weakest of the three systems that were available on the market at one time (the others were Beta and Video 2000), not to mention hardware and software standards in personal computing, which were arguably not adopted because of their technical merit.

On the positive side, it is true that RDF has sufficient expressive power (at least as a basis on which more layers can be built). And ultimately the Semantic Web will not be programmed in RDF, but rather with user-friendly tools that will automatically translate higher representations into RDF. Using RDF offers the benefit that information maps unambiguously to a model. And since it is likely that RDF will become a standard, the benefits of drafting data in RDF can be seen as similar to drafting information in HTML in the early days of the Web.

3.3 RDF: XML-Based Syntax

An RDF document consists of an rdf:RDF element, the content of which is a number of descriptions. For example, consider the domain of university courses and lecturers at Griffith University in the year 2001.

```xml
<!DOCTYPE owl [  
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">  ]>
```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:uni="http://www.mydomain.org/uni-ns#">

    <rdf:Description rdf:about="949352">
        <uni:name>Grigoris Antoniou</uni:name>
        <uni:title>Professor</uni:title>
    </rdf:Description>

    <rdf:Description rdf:about="949318">
        <uni:name>David Billington</uni:name>
        <uni:title>Associate Professor</uni:title>
        <uni:age rdf:datatype="&xsd;integer">27</uni:age>
    </rdf:Description>

    <rdf:Description rdf:about="949111">
        <uni:name>Michael Maher</uni:name>
        <uni:title>Professor</uni:title>
    </rdf:Description>

    <rdf:Description rdf:about="CIT1111">
        <uni:courseName>Discrete Mathematics</uni:courseName>
        <uni:isTaughtBy>David Billington</uni:isTaughtBy>
    </rdf:Description>

    <rdf:Description rdf:about="CIT1112">
        <uni:courseName>Concrete Mathematics</uni:courseName>
        <uni:isTaughtBy>Grigoris Antoniou</uni:isTaughtBy>
    </rdf:Description>

    <rdf:Description rdf:about="CIT2112">
        <uni:courseName>Programming III</uni:courseName>
        <uni:isTaughtBy>Michael Maher</uni:isTaughtBy>
    </rdf:Description>

    <rdf:Description rdf:about="CIT3112">
        <uni:courseName>Theory of Computation</uni:courseName>
        <uni:isTaughtBy>David Billington</uni:isTaughtBy>
    </rdf:Description>

    <rdf:Description rdf:about="CIT3116">

</rdf:RDF>
Let us make a few comments. First, the namespace mechanism of XML is used, but in an expanded way. In XML namespaces are only used for disambiguation purposes. In RDF external namespaces are expected to be RDF documents defining resources, which are then used in the importing RDF document. This mechanism allows the reuse of resources by other people who may decide to insert additional features into these resources. The result is the emergence of large, distributed collections of knowledge.

Second, the rdf:about attribute of the element rdf:Description is strictly speaking equivalent meaning to that of an ID attribute, but it is often used to suggest that the object about which a statement is made has already been “defined” elsewhere. Formally speaking, a set of RDF statements together simply forms a large graph, relating things to other things through properties, and there is no such thing as “defining” an object in one place and referring to it elsewhere. Nevertheless, in the serialized XML syntax, it is sometimes useful (if only for human readability) to suggest that one location in the XML serialization is the “defining” location, while other locations state “additional” properties about an object that has been “defined” elsewhere.

In fact the preceding example is slightly misleading. If we wanted to be absolutely correct, we should replace all occurrences of course and staff ID’s, such as 949352 and CIT3112, by references to the external namespace, for example

```xml
<rdf:Description rdf:about="http://www.mydomain.org/uni-ns/#CIT3112">
</rdf:Description>
```

We have refrained from doing so to improve readability of our initial example because we are primarily interested here in the ideas of RDF. However, readers should be aware that this would be the precise way of writing a correct RDF document.

The content of rdf:Description elements are called property elements. For example, in the description

```xml
<rdf:Description rdf:about="CIT3116">
  <uni:courseName>Knowledge Representation</uni:courseName>
  <uni:isTaughtBy>Grigoris Antoniou</uni:isTaughtBy>
</rdf:Description>
```
the two elements uni:courseName and uni:isTaughtBy both define property-value pairs for CIT3116. The preceding description corresponds to two RDF statements.

Third, the attribute rdf:datatype="&xsd;integer" is used to indicate the data type of the value of the age property. Even though the age property has been defined to have "&xsd;integer" as its range, it is still required to indicate the type of the value of this property each time it is used. This is to ensure that an RDF processor can assign the correct type of the property value even if it has not seen the corresponding RDF Schema definition before (a scenario that is quite likely to occur in the unrestricted World Wide Web).

Finally, the property elements of a description must be read conjunctively. In the preceding example, the subject is called “Knowledge Representation” and is taught by Grigoris Antoniou.

3.3.1 The rdf:resource Attribute

The preceding example was not satisfactory in one respect: the relationships between courses and lecturers were not formally defined but existed implicitly through the use of the same name. To a machine, the use of the same name may just be a coincidence: for example, the David Billington who teaches CIT3112 may not be the same person as the person with ID 949318 who happens to be called David Billington. What we need instead is a formal specification of the fact that, for example, the teacher of CIT1111 is the staff member with number 949318, whose name is David Billington. We can achieve this effect using an rdf:resource attribute:

```xml
<rdf:Description rdf:about="CIT1111">
  <uni:courseName>Discrete Mathematics</uni:courseName>
  <uni:isTaughtBy rdf:resource="949318"/>
</rdf:Description>

<rdf:Description rdf:about="949318">
  <uni:name>David Billington</uni:name>
  <uni:title>Associate Professor</uni:title>
</rdf:Description>
```

We note that in case we had defined the resource of the staff member with ID number 939318 in the RDF document using the ID attribute instead of the about attribute, we would have had to use a # symbol in front of 949318 in the value of rdf:resource:
3.3 RDF: XML-Based Syntax

The same is true for externally defined resources: For example, we refer to the externally defined resource CIT1111 by using

http://www.mydomain.org/uni-ns/#CIT1111

as the value of rdf:about, where www.mydomain.org/uni-ns/ is the URI where the definition of CIT1111 is found. In other words, a description with an ID defines a fragment URI, which can be used to reference the defined description.

3.3.2 Nested Descriptions

Descriptions may be defined within other descriptions. For example, we may replace the descriptions of the previous example with the following, nested description:

Other courses, such as CIT3112, can still refer to the new resource 949318. In other words, although a description may be defined within another description, its scope is global.

3.3.3 The rdf:type Element

In our examples so far, the descriptions fall into two categories: courses and lecturers. This fact is clear to human readers, but has not been formally de-
clared anywhere, so it is not accessible to machines. In RDF it is possible to make such statements using the rdf:type element. Here are a couple of descriptions that include typing information.

```
<rdf:Description rdf:about="CIT1111">
  <rdf:type rdf:resource="&uni;course"/>
  <uni:courseName>Discrete Mathematics</uni:courseName>
  <uni:isTaughtBy rdf:resource="#949318"/>
</rdf:Description>

<rdf:Description rdf:about="#949318">
  <rdf:type rdf:resource="&uni;lecturer"/>
  <uni:name>David Billington</uni:name>
  <uni:title>Associate Professor</uni:title>
</rdf:Description>
```

Note that rdf:type allows us to introduce some structure to the RDF document. More structuring possibilities are introduced later in this chapter when we discuss RDF Schema.

### 3.3.4 Abbreviated Syntax

It is possible to abbreviate the syntax of RDF documents. The simplification rules are

1. Childless property elements within description elements may be replaced by XML attributes, as in XML.

2. For description elements with a typing element we can use the name specified in the rdf:type element instead of rdf:Description.

For example, the description

```
<rdf:Description rdf:ID="CIT1111">
  <rdf:type rdf:resource="&uni;course"/>
  <uni:courseName>Discrete Mathematics</uni:courseName>
  <uni:isTaughtBy rdf:resource="#949318"/>
</rdf:Description>
```

is (according to rule 1 applied to uni:courseName) equivalent to

```
<rdf:Description rdf:ID="CIT1111">
  uni:courseName="Discrete Mathematics">
```

3.3 RDF: XML-Based Syntax

and also (by rule 2) to

<uni:course rdf:ID="CIT1111"
    uni:courseName="Discrete Mathematics">
    <uni:isTaughtBy rdf:resource="#949318"/>
</uni:course>

Keep in mind that these three representations are just syntactic variations of the same RDF statement. That is, they are equivalent according to the RDF data model, although they have different XML syntax.

3.3.5 Container Elements

Container elements are used to collect a number of resources or attributes about which we want to make statements as a whole. In our example, we may wish to talk about the courses given by a particular lecturer. Three types of containers are available in RDF:

* rdf:Bag an unordered container, which may contain multiple occurrences (not true for a set). Typical examples are members of the faculty board and documents in a folder — examples where an order is not imposed.

* rdf:Seq an ordered container, which may contain multiple occurrences. Typical examples are the modules of a course, items on an agenda, an alphabetized list of staff members — examples where an order is imposed.

* rdf:Alt a set of alternatives. Typical examples are the document home and mirrors, and translations of a document in various languages.

The content of container elements are elements which are named rdf:_1, rdf:_2, and so on. Let us reformulate our entire RDF document.

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

<uni:lecturer rdf:about="949352"
    uni:name="Grigoris Antoniou"
    uni:title="Professor">
```
<uni:coursesTaught>
  <rdf:Bag>
    <rdf:_1 rdf:resource="CIT1112"/>
    <rdf:_2 rdf:resource="CIT3116"/>
  </rdf:Bag>
</uni:coursesTaught>
</uni:lecturer>

<uni:lecturer rdf:about="949318"
  uni:name="David Billington"
  uni:title="Associate Professor">
  <uni:coursesTaught>
    <rdf:Bag>
      <rdf:_1 rdf:resource="CIT1111"/>
      <rdf:_2 rdf:resource="CIT3112"/>
    </rdf:Bag>
  </uni:coursesTaught>
</uni:lecturer>

<uni:lecturer rdf:about="949111"
  uni:name="Michael Maher"
  uni:title="Professor">
  <uni:coursesTaught rdf:resource="CIT2112"/>
</uni:lecturer>

<uni:course rdf:about="CIT1111"
  uni:courseName="Discrete Mathematics">
  <uni:isTaughtBy rdf:resource="949318"/>
</uni:course>

<uni:course rdf:about="CIT1112"
  uni:courseName="Concrete Mathematics">
  <uni:isTaughtBy rdf:resource="949352"/>
</uni:course>

<uni:course rdf:about="CIT2112"
  uni:courseName="Programming III">
  <uni:isTaughtBy rdf:resource="949111"/>
</uni:course>

<uni:course rdf:about="CIT3112"
  uni:courseName="Theory of Computation"/>
Instead of `rdf:_1, rdf:_2`... it is possible to write `rdf:li`. We use this syntactic variant in the following example. Suppose the course CIT1111 is taught by either Grigoris Antoniou or David Billington:

```xml
<uni:course rdf:about="CIT1111"
    uni:courseName="Discrete Mathematics">
  <uni:lecturer>
    <rdf:Alt>
      <rdf:li rdf:resource="949352"/>
      <rdf:li rdf:resource="949318"/>
    </rdf:Alt>
  </uni:lecturer>
</uni:course>
```

The container elements have an optional `ID` attribute, with which the container can be identified and referred to:

```xml
<uni:lecturer rdf:about="949318"
    uni:name="David Billington"
    uni:title="Associate Professor">
  <uni:coursesTaught>
    <rdf:Bag rdf:ID="DBcourses">
      <rdf:_1 rdf:resource="CIT1111"/>
      <rdf:_2 rdf:resource="CIT3112"/>
    </rdf:Bag>
  </uni:coursesTaught>
</uni:lecturer>
```

A typical application of container elements is the representation of predicates with more than two arguments. We reconsider the example `referee(X, Y, Z)`, where `X` is the referee of a chess game between players `Y` and `Z`. One solution is to distinguish the referee `X` from the players `Y` and `Z`. The graphic representation is found in figure 3.4. The solution in XML-based syntax looks like this:
Note that rdf:Bag defines an anonymous auxiliary resource. We chose to use a bag because we assumed that no distinction between the players is made. If order were important, say the first-named player has White and the second Black, we would use a sequence instead.

A limitation of these containers is that there is no way to close them, to say “these are all the members of the container”. This is because, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members. RDF provides support for describing groups containing only the specified members, in the form of RDF collections. An RDF collection is a group of things represented as a list structure in the RDF graph. This list structure is constructed using a predefined collection vocabulary consisting of the predefined type rdf:List, the predefined properties rdf:first and rdf:rest, and the predefined resource rdf:nil. This allows us to write

```xml
<referee rdf:about="...#X">
  <players>
    <rdf:Bag>
      <rdf:li rdf:resource="...#Y"/>
      <rdf:li rdf:resource="...#Z"/>
    </rdf:Bag>
  </players>
</referee>
```
This states that CIT2112 is taught by teachers identified as the resources 949111, 949352, and 949318, and nobody else (indicated by the terminator symbol nil). A shorthand syntax for this has been defined, using the "Collection" value for the rdf:parseType attribute:

```xml
<rdf:Description rdf:about="CIT2112">
  <uni:isTaughtBy rdf:parseType="Collection">
    <rdf:Description rdf:about="949111"/>
    <rdf:Description rdf:about="949352"/>
    <rdf:Description rdf:about="949318"/>
  </uni:isTaughtBy>
</rdf:Description>
```
3.3.6 Reification

As we have said, sometimes we wish to make statements about other statements. To do so we must be able to refer to a statement using an identifier. RDF allows such reference through a reification mechanism which turns a statement into a resource. For example, the description

```xml
<rdf:Description rdf:about="949352">
  <uni:name>Grigoris Antoniou</uni:name>
</rdf:Description>
```

reifies as

```xml
<rdf:Statement rdf:about="StatementAbout949352">
  <rdf:subject rdf:resource="949352"/>
  <rdf:predicate rdf:resource="&uni;name"/>
  <rdf:object>Grigoris Antoniou</rdf:object>
</rdf:Statement>
```

Note that `rdf:subject`, `rdf:predicate`, and `rdf:object` allow us to access the parts of a statement.

The ID of the statement can be used to refer to it, as can be done for any description. We can either write an `rdf:Description` if we don’t want to talk about it further, or an `rdf:Statement` if we wish to refer to it.

If more than one property element is contained in a description element, the elements correspond to more than one statement. These statements can either be placed in a bag and referred to as an entity, or they can reify separately (see exercise 3.1).

3.4 RDF Schema: Basic Ideas

RDF is a universal language that lets users describe resources using their own vocabularies. RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. It is up to the user to do so in RDF Schema (RDFS).

3.4.1 Classes and Properties

How do we describe a particular domain? Let us consider the domain of courses and lecturers at Griffith University. First we have to specify the “things” we want to talk about. Here we make a first, fundamental distinction. On one hand, we want to talk about particular lecturers, such as David
Billington, and particular courses, such as Discrete Mathematics; we have already done so in RDF. But we also want to talk about courses, first-year courses, lecturers, professors, and so on. What is the difference? In the first case we talk about individual objects (resources), in the second we talk about classes that define types of objects.

A class can be thought of as a set of elements. Individual objects that belong to a class are referred to as instances of that class. We have already defined the relationship between instances and classes in RDF using rdf:type.

An important use of classes is to impose restrictions on what can be stated in an RDF document using the schema. In programming languages, typing is used to prevent nonsense from being written (such as $A + 1$, where $A$ is an array; we lay down that the arguments of $+$ must be numbers). The same is needed in RDF. After all, we would like to disallow statements such as

Discrete Mathematics is taught by Concrete Mathematics.
Room MZH5760 is taught by David Billington.

The first statement is nonsensical because we want courses to be taught by lecturers only. This imposes a restriction on the values of the property “is taught by”. In mathematical terms, we restrict the range of the property.

The second statement is nonsensical because only courses can be taught. This imposes a restriction on the objects to which the property can be applied. In mathematical terms, we restrict the domain of the property.

### 3.4.2 Class Hierarchies and Inheritance

Once we have classes we would also like to establish relationships between them. For example, suppose that we have classes for

- staff members
- assistant professors
- academic staff members
- administrative staff members
- professors
- technical support staff members
- associate professors

These classes are not unrelated to each other. For example, every professor is an academic staff member. We say that “professor” is a subclass of “academic staff member”, or equivalently, that “academic staff member” is a superclass of “professor”. The subclass relationship defines a hierarchy of classes, as shown in figure 3.5. In general, $A$ is a subclass of $B$ if every instance of $A$ is also an instance of $B$. There is no requirement in RDF Schema that the classes
Figure 3.5  A hierarchy of classes

together form a strict hierarchy. In other words, a subclass graph as in figure 3.5 need not be a tree. A class may have multiple superclasses. If a class $A$ is a subclass of both $B_1$ and $B_2$, this simply means that every instance of $A$ is both an instance of $B_1$ and an instance of $B_2$.

A hierarchical organization of classes has a very important practical significance, which we outline now. Consider the range restriction

Courses must be taught by academic staff members only.

Suppose Michael Maher were defined as a professor. Then, according to the preceding restriction, he is not allowed to teach courses. The reason is that there is no statement specifying that Michael Maher is also an academic staff member. It would be counterintuitive to overcome this difficulty by adding that statement to our description. Instead we would like Michael Maher to inherit the ability to teach from the class of academic staff members. Exactly this is done in RDF Schema.

By doing so, RDF Schema fixes the semantics of “is a subclass of”. Now it is not up to an application to interpret “is a subclass of”; instead its intended meaning must be used by all RDF processing software. By making such semantic definitions RDFS is a (still limited), language for defining the
3.4 RDF Schema: Basic Ideas

semantics of particular domains. Stated another way, RDF Schema is a primitive ontology language.

Classes, inheritance, and properties are, of course, known in other fields of computing, for example in object-oriented programming. But while there are many similarities, there are differences, too. In object-oriented programming, an object class defines the properties that apply to it. To add new properties to a class means to modify the class.

However, in RDFS, properties are defined globally, that is, they are not encapsulated as attributes in class definitions. It is possible to define new properties that apply to an existing class without changing that class.

On one hand, this is a powerful mechanism with far-reaching consequences: we may use classes defined by others and adapt them to our requirements through new properties. On the other hand, this handling of properties deviates from the standard approach that has emerged in the area of modeling and object-oriented programming. It is another idiosyncratic feature of RDF/RDFS.

3.4.3 Property Hierarchies

We saw that hierarchical relationships between classes can be defined. The same can be done for properties. For example, “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. For example, $a$ may be the convener of the course, or a tutor who marks student homework but does not teach $c$.

In general, $P$ is a subproperty of $Q$ if $Q(x, y)$ whenever $P(x, y)$.

3.4.4 RDF versus RDFS Layers

As a final point, we illustrate the different layers involved in RDF and RDFS using a simple example. Consider the RDF statement

Discrete Mathematics is taught by David Billington.

The schema for this statement may contain classes such as lecturers, academic staff members, staff members, first-year courses, and properties such as is taught by, involves, phone, employee id. Figure 3.6 illustrates the layers of RDF and RDF Schema for this example. In this figure, blocks are properties, ellipses above the dashed line are classes, and ellipses below the dashed line are instances.
The schema in figure 3.6 is itself written in a formal language, RDF Schema, that can express its ingredients: subClassOf, Class, Property, subPropertyOf, Resource, and so on. Next we describe the language of RDF Schema in more detail.

3.5 RDF Schema: The Language

RDF Schema provides modeling primitives for expressing the information described in section 3.4. One decision that must be made is what formal lan-
guage to use. It should not be surprising that RDF itself will be used: the modeling primitives of RDF Schema are defined using resources and properties. This choice can be justified by looking at figure 3.6: we presented this figure as displaying a class/property hierarchy plus instances, but it is, of course, itself simply a labeled graph that can be encoded in RDF. Remember that RDF allows one to express any statement about any resource, and that anything that has a URI can be a resource. So, if we wish to say that the class “lecturer” is a subclass of “academic staff member”, we may

1. define resources lecturer, academicStaffMember, and subClassOf
2. define subClassOf to be a property
3. write the triple (subClassOf, lecturer, academicStaffMember)

All these steps are within the capabilities of RDF. So, an RDFS document (that is an RDF schema) is just an RDF document, and we use the XML-based syntax of RDF. In particular, all syntactic definitions of section 3.3 must be followed.

Now we define the modeling primitives of RDF Schema.

### 3.5.1 Core Classes

The core classes are

- rdfs:Resource, the class of all resources.
- rdfs:Class, the class of all classes.
- rdfs:Literal, the class of all literals (strings). At present, literals form the only “data type” of RDF/RDFS.
- rdf:Property, the class of all properties.
- rdf:Statement, the class of all reified statements.

For example, a class lecturer can be defined as follows:

```xml
<rdf:Class rdf:ID="lecturer"> ...
</rdf:Class>
```
3.5.2 Core Properties for Defining Relationships

The core properties for defining relationships are

`rdf:type`, which relates a resource to its class (see section 3.3.3). The resource is declared to be an instance of that class.

`rdfs:subClassOf`, which relates a class to one of its superclasses; all instances of a class are instances of its superclass. Note that a class may be a subclass of more than one class. As an example, the class `femaleProfessor` may be a subclass of both `female` and `professor`.

`rdfs:subPropertyOf`, which relates a property to one of its superproperties.

Here is an example stating that all lecturers are staff members:

```xml
<rdfs:Class rdf:about="lecturer">
  <rdfs:subClassOf rdf:resource="staffMember"/>
</rdfs:Class>
```

Note that `rdfs:subClassOf` and `rdfs:subPropertyOf` are transitive, by definition. Also, it is interesting that `rdfs:Class` is a subclass of `rdfs:Resource` (every class is a resource), and `rdfs:Resource` is an instance of `rdfs:Class` (`rdfs:Resource` is the class of all resources, so it is a class!). For the same reason, every class is an instance of `rdfs:Class`.

3.5.3 Core Properties for Restricting Properties

The core properties for restricting properties are

`rdfs:domain`, which specifies the domain of a property $P$, that is, the class of those resources that may appear as subjects in a triple with predicate $P$. If the domain is not specified, then any resource can be the subject.

`rdfs:range`, which specifies the range of a property $P$, that is, the class of those resources that may appear as values in a triple with predicate $P$.

Here is an example, stating that phone applies to staff members only and that its value is always a literal.

```xml
<rdf:Property rdf:ID="phone">
  <rdfs:domain rdf:resource="#staffMember"/>
  <rdfs:range rdf:resource="&rdf;Literal"/>
</rdf:Property>
```
3.5 RDF Schema: The Language

In RDF Schema there are also

\texttt{rdfs:ConstraintResource}, the class of all constraints

\texttt{rdfs:ConstraintProperty}, which contains all properties that define constraints. It has only two instances, \texttt{rdfs:domain} and \texttt{rdfs:range}. It is defined to be a subclass of \texttt{rdfs:ConstraintResource} and \texttt{rdf:Property}

Figures 3.7 and 3.8 show the relationships between core modeling primitives in RDFS.
3.5.4 Useful Properties for Reification

The following are some useful properties for reification (see section 3.3.6):

- `rdf:subject`, which relates a reified statement to its subject
- `rdf:predicate`, which relates a reified statement to its predicate
- `rdf:object`, which relates a reified statement to its object

3.5.5 Container Classes

As mentioned in section 3.3.5, the container elements are

- `rdf:Bag`, the class of bags
- `rdf:Seq`, the class of sequences
- `rdf:Alt`, the class of alternatives.
- `rdfs:Container`, which is a superclass of all container classes, including the three preceding ones.

3.5.6 Utility Properties

A resource may be defined and described in many places on the Web. The following properties allow us to define links to those addresses:

- `rdfs:seeAlso` relates a resource to another resource that explains it
- `rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso` and relates a resource to the place where its definition, typically an RDF schema, is found.

Often it is useful to provide more information, intended for human readers. This can be done with the following properties:

- `rdfs:comment`. Comments, typically longer text, can be associated with a resource.
- `rdfs:label`. A human-friendly label (name) is associated with a resource. Among other purposes, it may serve as the name of a node in a graphic representation of the RDF document.
3.5.7 Example: A University

We refer to the courses and lecturers example, and provide a conceptual model of the domain, that is, an ontology.

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

    <rdfs:Class rdf:ID="lecturer">
        <rdfs:comment>The class of lecturers
           All lecturers are academic staff members.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="academicStaffMember">
        <rdfs:comment>The class of academic staff members</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#staffMember"/>
    </rdfs:Class>

    <rdfs:Class rdf:ID="staffMember">
        <rdfs:comment>The class of staff members</rdfs:comment>
    </rdfs:Class>

    <rdfs:Class rdf:ID="course">
        <rdfs:comment>The class of courses</rdfs:comment>
    </rdfs:Class>

    <rdf:Property rdf:ID="involves">
        <rdfs:comment>It relates only courses to lecturers.</rdfs:comment>
        <rdfs:domain rdf:resource="#course"/>
        <rdfs:range rdf:resource="#lecturer"/>
    </rdf:Property>

    <rdf:Property rdf:ID="isTaughtBy">
        <rdfs:comment>Inherits its domain ("course") and range ("lecturer")
    </rdf:Property>

</rdf:RDF>
```
3.5.8 Example: Motor Vehicles

Here we present a simple ontology of motor vehicles. The class relationships are shown in figure 3.9.

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

![Figure 3.9 Class hierarchy for the motor vehicles example](image)

```xml
from its superproperty "involves"
   </rdfs:comment>
   <rdfs:subPropertyOf rdf:resource="#involves"/>
</rdf:Property>

<rdf:Property rdf:ID="phone">
   <rdfs:comment>
      It is a property of staff members
      and takes literals as values.
   </rdfs:comment>
   <rdfs:domain rdf:resource="#staffMember"/>
   <rdfs:range rdf:resource="#&rdf;Literal"/>
</rdf:Property>
```

```xml
</rdf:RDF>
```
Now that we know the main components of the RDF and RDFS languages, it may be instructive to look at the definitions of RDF and RDFS. These definitions are expressed in the language of RDF Schema. One task is to see how easily they can be read now that the meaning of each component has been clarified.

The following definitions are just part of the full language specification. The remaining parts are found in the namespaces specified in `rdf:RDF`.

### 3.6 RDF and RDF Schema in RDF Schema

Now that we know the main components of the RDF and RDFS languages, it may be instructive to look at the definitions of RDF and RDFS. These definitions are expressed in the language of RDF Schema. One task is to see how easily they can be read now that the meaning of each component has been clarified.

The following definitions are just part of the full language specification. The remaining parts are found in the namespaces specified in `rdf:RDF`.

#### 3.6.1 RDF

```xml
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdfs:Class rdf:ID="Statement"
    rdfs:comment="The class of triples consisting of a
```
Describing Web Resources in RDF

predicate, a subject and an object (that is, a reified statement)"

<rdfs:Class rdf:ID="Property"
  rdfs:comment="The class of properties"/>

<rdfs:Class rdf:ID="Bag"
  rdfs:comment="The class of unordered collections"/>

<rdfs:Class rdf:ID="Seq"
  rdfs:comment="The class of ordered collections"/>

<rdfs:Class rdf:ID="Alt"
  rdfs:comment="The class of collections of alternatives"/>

<rdf:Property rdf:ID="predicate"
  rdfs:comment="Identifies the property used in a statement when representing the statement in reified form">
  <rdfs:domain rdf:resource="#Statement"/>
  <rdfs:range rdf:resource="#Property"/>
</rdf:Property>

<rdf:Property rdf:ID="subject"
  rdfs:comment="Identifies the resource that a statement is describing when representing the statement in reified form">
  <rdfs:domain rdf:resource="#Statement"/>
</rdf:Property>

<rdf:Property rdf:ID="object"
  rdfs:comment="Identifies the object of a statement when representing the statement in reified form"/>

<rdf:Property rdf:ID="type"
  rdfs:comment="Identifies the class of a resource. The resource is an instance of that class."/>
3.6 RDF and RDF Schema in RDF Schema

3.6.2 RDF Schema

The namespaces do not provide the full definition of RDF and RDF Schema. Consider, for example, rdfs:subClassOf. The namespace specifies only that it applies to classes and has a class as a value. The meaning of being a subclass, namely, that all instances of one class are also instances of its superclass, is not expressed anywhere. In fact, it cannot be expressed in an RDF document. If it could, there would be no need for defining RDF Schema.

We provide a formal semantics in the next section. Of course, RDF parsers and other software tools for RDF (including query processors) must be aware of the full semantics.
3.7 An Axiomatic Semantics for RDF and RDF Schema

In this section we formalize the meaning of the modeling primitives of RDF and RDF Schema. Thus we capture the semantics of RDF and RDFS.

The formal language we use is predicate logic, universally accepted as the foundation of all (symbolic) knowledge representation. Formulas used in the formalization are referred to as axioms.

By describing the semantics of RDF and RDFS in a formal language like logic we make the semantics unambiguous and machine accessible. Also, we provide a basis for reasoning support by automated reasoners manipulating logical formulas.

3.7.1 The Approach

All language primitives in RDF and RDF Schema are represented by constants: Resource, Class, Property, subClassOf, and so on. A few predefined predicates are used as a foundation for expressing relationships between the constants.

An auxiliary theory of lists is used. It has function symbols

$$\text{nil}$$ (empty list)
$$\text{cons}(x, l)$$ (adds an element to the front of the list)
$$\text{first}(l)$$ (returns the first element)
$$\text{rest}(l)$$ (returns the rest of the list)

and predicate symbols

$$\text{item}(x, l)$$ (tests if an element occurs in the list)
$$\text{list}(l)$$ (tests whether \(l\) is a list)

Lists are used to represent containers in RDF. They are also needed to capture the meaning of certain constructs (such as cardinality constraints) in richer ontology languages.

Most axioms provide typing information. For example,

$$\text{Type}(\text{subClassOf}, \text{Property})$$

says that \(\text{subClassOf}\) is a property. We use predicate logic with equality. Variable names begin with \(?\). All axioms are implicitly universally quantified.
3.7 An Axiomatic Semantics for RDF and RDF Schema

Here we show the definition of most elements of RDF and RDF Schema. The axiomatic semantics of the full languages is found in an online document; see reference (Fikes and McGuinness 2001).

3.7.2 Basic Predicates

The basic predicates are

\[ PropVal(P, R, V), \] a predicate with three arguments, which is used to represent an RDF statement with resource \( R \), property \( P \) and value \( V \)

\[ Type(R, T), \] short for \( PropVal(type, R, T) \), which specifies that the resource \( R \) has the type \( T \)

\[ Type(?r, ?t) \longrightarrow PropVal(type, ?r, ?t) \]

3.7.3 RDF

An RDF statement (triple) \( (P, R, V) \) is represented as \( PropVal(P, R, V) \).

Classes

In our language we have constants \( Class, Resource, Property, Literal \). All classes are instances of \( Class \), that is, they have the type \( Class \):

\[ Type(Class, Class) \]
\[ Type(Resource, Class) \]
\[ Type(Property, Class) \]
\[ Type(Literal, Class) \]

\( Resource \) is the most general class: every object is a resource. Therefore, every class and every property is a resource:

\[ Type(?p, Property) \longrightarrow Type(?p, Resource) \]
\[ Type(?c, Class) \longrightarrow Type(?c, Resource) \]

Finally, the predicate in an RDF statement must be a property:

\[ PropVal(?p, ?r, ?v) \longrightarrow Type(?p, Property) \]
The **type** Property

`type` is a property:

\[
\text{Type}(\text{type}, \text{Property})
\]

Note that it is equivalent to `PropVal(type, type, Property)`: the type of `type` is `Property`. `type` can be applied to resources and has a class as its value:

\[
\text{Type}(?r, ?c) \rightarrow (\text{Type}(?r, \text{Resource}) \land \text{Type}(?c, \text{Class}))
\]

The Auxiliary **FuncProp** Property

A functional property is a property that is a function: it relates a resource to at most one value. Functional properties are not a concept of RDF but are used in the axiomatization of other primitives.

The constant `FuncProp` represents the class of all functional properties. `P` is a functional property if, and only if, it is a property, and there are no `x`, `y_1`, and `y_2` such that `P(x, y_1), P(x, y_2), and y_1 \neq y_2`.

\[
\text{Type}(?p, \text{FuncProp}) \leftrightarrow
(\text{Type}(?p, \text{Property}) \land \forall ?r \forall ?v_1 \forall ?v_2
\]
\[
(\text{PropVal}(?p, ?r, ?v_1) \land \text{PropVal}(?p, ?r, ?v_2) \rightarrow ?v_1 = ?v_2))
\]

Reified Statements

The constant `Statement` represents the class of all reified statements. All reified statements are resources, and `Statement` is an instance of `Class`:

\[
\text{Type}(?s, \text{Statement}) \rightarrow \text{Type}(?s, \text{Resource})
\]
\[
\text{Type}(\text{Statement}, \text{Class})
\]

A reified statement can be decomposed into the three parts of an RDF triple:

\[
\text{Type}(\text{st}, \text{Statement}) \rightarrow
\exists ?p \exists ?r \exists ?v (\text{PropVal}(\text{Predicate}, ?s, ?p) \land
\text{PropVal}(\text{Subject}, ?st, ?r) \land \text{PropVal}(\text{Object}, ?st, ?v))
\]

`Subject`, `Predicate`, and `Object` are functional properties, that is, every statement has exactly one subject, one predicate and one object:
3.7 An Axiomatic Semantics for RDF and RDF Schema

\[ Type(\text{Subject}, \text{FuncProp}) \]
\[ Type(\text{Predicate}, \text{FuncProp}) \]
\[ Type(\text{Object}, \text{FuncProp}) \]

Their typing information is
\[ PropVal(\text{Subject}, ?st, ?r) \rightarrow (Type(?st, \text{Statement}) \land Type(?r, \text{Resource})) \]
\[ PropVal(\text{Predicate}, ?st, ?p) \rightarrow (Type(?st, \text{Statement}) \land Type(?p, \text{Property})) \]
\[ PropVal(\text{Object}, ?st, ?v) \rightarrow (Type(?st, \text{Statement}) \land (Type(?v, \text{Resource}) \lor Type(?v, \text{Literal}))) \]

The last axiom says, if \text{Object} appears as the property in an RDF statement, then it must apply to a reified statement and have as value either a resource or a literal.

**Containers**

All containers are resources:
\[ Type(?c, \text{Container}) \rightarrow Type(?c, \text{Resource}) \]

Containers are lists:
\[ Type(?c, \text{Container}) \rightarrow \text{list}(?c) \]

Containers are bags or sequences or alternatives:
\[ Type(?c, \text{Container}) \leftarrow (Type(?c, \text{Bag}) \lor Type(?c, \text{Seq}) \lor Type(?c, \text{Alt})) \]

Bags and sequences are disjoint:
\[ \neg (Type(?x, \text{Bag}) \land Type(?x, \text{Seq})) \]

For every natural number \( n > 0 \), there is the selector \_\_\_\_n, which selects the \( n \)th element of a container. It is a functional property
\[ Type(_\_\_\_n, \text{FuncProp}) \]

and applies to containers only:
\[ PropVal(_\_\_\_n, ?c, ?o) \rightarrow Type(?c, \text{Container}) \]
3.7.4 RDF Schema

Subclasses and Subproperties

subClassOf is a property:

\[ \text{Type}(\text{subClassOf}, \text{Property}) \]

If a class \( C \) is a subclass of a class \( C' \), then all instances of \( C \) are also instances of \( C' \):

\[
\text{PropVal}(\text{subClassOf}, ?c, ?c') \leftarrow
(\text{Type}(?, \text{Class}) \land \text{Type}(?, \text{Class}) \land
\forall x (\text{Type}(?, ?c) \rightarrow \text{Type}(?, ?c')))\]

Similarly for subPropertyOf: \( P \) is a subproperty of \( P' \) if \( P'(x, y) \) whenever \( P(x, y) \):

\[
\text{Type}(\text{subPropertyOf}, \text{Property}) \quad \text{PropVal}(\text{subPropertyOf}, ?p, ?p') \leftarrow
(\text{Type}(?, \text{Property}) \land \text{Type}(?, \text{Property}) \land
\forall r (\forall v (\text{PropVal}(?, ?, r, ?v) \rightarrow \text{PropVal}(?, ?, r, ?v)))\]

Constraints

Every constraint resource is a resource:

\[
\text{PropVal}(\text{subClassOf}, \text{ConstraintResource}, \text{Resource})
\]

Constraint properties are all properties that are also constraint resources:

\[
\text{Type}(?, \text{ConstraintProperty}) \leftarrow
(\text{Type}(?, \text{ConstraintResource}) \land \text{Type}(?, \text{Property}))
\]

domain and range are constraint properties:

\[
\text{Type}(\text{domain}, \text{ConstraintProperty})
\quad \text{Type}(\text{range}, \text{ConstraintProperty})
\]

domain and range define the domain, respectively range, of a property. Recall that the domain of a property \( P \) is the set of all objects to which \( P \) applies. If the domain of \( P \) is \( D \), then for every \( P(x, y), x \in D \).
3.8 A Direct Inference System for RDF and RDFS

\[ PropVal(domain, ?p, ?d) \rightarrow \]
\[ \forall ?x \forall ?y (PropVal(?p, ?x, ?y) \rightarrow Type(?x, ?d)) \]

The range of a property \( P \) is the set of all values \( P \) can take. If the range of \( P \) is \( R \), then for every \( P(x, y) \), \( y \in R \).

\[ PropVal(range, ?p, ?r) \rightarrow \]
\[ \forall ?x \forall ?y (PropVal(?p, ?x, ?y) \rightarrow Type(?y, ?r)) \]

Formulas that can be inferred from the preceding ones:

\[ PropVal(domain, range, Property) \]
\[ PropVal(range, range, Class) \]
\[ PropVal(domain, domain, Property) \]
\[ PropVal(range, domain, Class) \]

Thus we have formalized the semantics of RDF and RDFS. An agent equipped with this knowledge is able to draw interesting conclusions. For example, given that the domain of \( teaches \) is \( academicStaffMember \), that \( academicStaffMember \) is a subclass of \( staffMembers \), and that \( teaches(DB, DiMa) \), the agent can automatically deduce \( staffMember(DB) \) using the predicate logic semantics or one of the predicate logic proof systems.

3.8 A Direct Inference System for RDF and RDFS

As stated above, the axiomatic semantics detailed in section 3.7 can be used for automated reasoning with RDF and RDF Schema. However, it requires a first-order logic proof system to do so. This is a very heavy requirement and also one that is unlikely to scale when millions of statements are involved (e.g. millions of statements of the form \( Type(?r, ?c) \)).

For this reason, RDF has also been given a semantics (and an inference systems that is sound and complete for this semantics) directly in terms of RDF triples instead of restating RDF in terms of first-order logic, as was done in the axiomatic semantics of section 3.7.

This inference system consists of rules of the form

\[
\begin{align*}
\text{IF} & \quad \text{E contains certain triples} \\
\text{THEN} & \quad \text{add to E certain additional triples}
\end{align*}
\]
(where $E$ is an arbitrary set of RDF triples).

Without repeating the entire set of inference rules (which can be found in the official RDF documents), we give here a few basic examples:

\[
\text{IF } E \text{ contains the triple } (?x, ?p, ?y) \\
\text{THEN } E \text{ also contains the triple } (?p, \text{rdf:} \text{type, rdf:property})
\]

This states that any resource $?p$ that is used in the property position of a triple can be inferred to be a member of the class rdf:property.

A somewhat more interesting example is the following rule:

\[
\text{IF } E \text{ contains the triples } (?u, \text{rdfs:} \text{subClassOf,} ?v) \\
\text{and } (?v, \text{rdfs:} \text{subClassOf,} ?w) \\
\text{THEN } E \text{ also contains the triple } (?u, \text{rdfs:} \text{subClassOf,} ?w)
\]

which encodes the transitivity of the subclass relation.

Closely related is the rule

\[
\text{IF } E \text{ contains the triples } (?x, \text{rdf:} \text{type,} ?u) \\
\text{and } (?u, \text{rdfs:} \text{subClassOf,} ?v) \\
\text{THEN } E \text{ also contains the triple } (?x, \text{rdf:} \text{type,} ?v)
\]

which is the essential definition of the meaning of rdfs:subClassOf.

A final example often comes as a surprise to people first looking at RDF Schema:

\[
\text{IF } E \text{ contains the triples } (?x, ?p, ?y) \\
\text{and } (?p, \text{rdfs:} \text{range,} ?u) \\
\text{THEN } E \text{ also contains the triple } (?y, \text{rdf:} \text{type,} ?u)
\]

This rule states that any resource $?y$ which appears as the value of a property $?p$ can be inferred to be a member of the range of $?p$. This shows that range definitions in RDF Schema are not used to restrict the range of a property, but rather to infer the membership of the range.

The total set of these closure rules is no larger than a few dozen and can be efficiently implemented without sophisticated theorem-proving technology.

### 3.9 Querying in RQL

In this section we will introduce a query language for RDF. Before doing so, we have to say why we need a new query language instead of using an XML query language. The answer is that XML is located at a lower level of abstraction than RDF. This fact would lead to complications if we were
querying RDF documents with an XML-based language. Let us illustrate this point.

As we have already seen, there are various ways of syntactically representing an RDF statement in XML. For example, suppose we wish to retrieve the titles of all lecturers. The description of a particular lecturer might look like this:

```xml
<rdf:Description rdf:about="949318">
  <rdf:type rdf:resource="&uni;lecturer"/>
  <uni:name>David Billington</uni:name>
  <uni:title>Associate Professor</uni:title>
</rdf:Description>
```

An appropriate XPath query is

```
/rdf:Description[rdf:type="http://www.mydomain.org/uni-ns#lecturer"]/uni:title
```

But we could have written the same description as follows:

```xml
<uni:lecturer rdf:about="949318">
  <uni:name>David Billington</uni:name>
  <uni:title>Associate Professor</uni:title>
</uni:lecturer>
```

Now the previous XPath query does not work; we have to write

```
//uni:lecturer/uni:title
```

instead. And a third possible representation of the same description is

```xml
<uni:lecturer rdf:about="949318"
             uni:name="David Billington"
             uni:title="Associate Professor"/>
```

For this syntactic variation, yet another XPath query must be provided:

```
//uni:lecturer/@uni:title
```

Since each description of an individual lecturer may have any of these equivalent forms, we must write different XPath queries.

A better way is, of course, to write queries at the level of RDF. An appropriate query language must understand RDF, that is, it must understand not only the syntax but also the data model of RDF and the semantics of RDF vocabulary.

In addition, a query language should also understand the semantics of RDF Schema. For example, given the information
3. Describing Web Resources in RDF

A query for the names of all lecturers should return both Grigoris Antoniou and David Billington.

At the time of writing (mid 2003), there is no standardization of query languages for RDF and RDFS, neither de jure by W3C, nor de facto by the community. In our discussion we have chosen to discuss RQL because it illustrates a number of features that will be part of any reasonable query language for RDF and RDFS, such as path expressions and schema awareness. However, other query languages exist (e.g., RDQL), and even RQL itself is subject to change.

3.9.1 Basic Queries

The query `Class` retrieves all classes, and the query `Property` retrieves all properties. To retrieve the instances of a class, for example, `course`, we write

```
course
```

This query will return all instances of the subclasses of `course`, too, which is perfectly correct. But if we do not wish to retrieve inherited instances, then we have to write

```
^course
```

The resources and values of triples with a specific property, for example, `involves`, are retrieved using simply the query `involves`. The result includes all subproperties of `involves`, for example, it retrieves also inherited triples from property `isTaughtBy`. If we do not want these additional results, then we have to write `^involves` instead.
3.9.2 Using select-from-where

As in SQL,

- `select` specifies the number and order of retrieved data
- `from` is used to navigate through the data model
- `where` imposes constraints on possible solutions

For example, to retrieve all phone numbers of staff members, we can write

```
select X,Y
from {X}phone{Y}
```

Here `X` and `Y` are variables, and `{X}phone{Y}` represents a resource-property-value triple. To retrieve all lecturers and their phone numbers, we can write

```
select X,Y
from lecturer{X}.phone{Y}
```

Here `lecturer{X}` collects all instances of the class `lecturer`, as discussed, and binds the result to the variable `X`. The second part collects all triples with predicate `phone`. But there is an implicit join here, in that we restrict the second query only to those triples, the resource of which is in the variable `X`; in our example, we restrict the domain of `phone` to lecturers. A dot `.` denotes the implicit join.

We demonstrate an explicit join by a query that retrieves the name of all courses taught by the lecturer with ID 949352.

```
select N
from course{X}.isTaughtBy{Y}, {C}name{N}
where Y="949352" and X=C
```

Apart from `=` there exist other comparison operators. For example, `X<Y` means “`X` is lower than `Y`”. In case `X` and `Y` are strings, `X` comes before `Y` in the lexicographic order. If `X` and `Y` are classes, `X` is a subclass of `Y`.

3.9.3 Querying the Schema

RQL allows us to retrieve schema information. Schema variables have a name with prefix `$` (for classes) or `@` (for properties). For example,
select X,$X,Y,$Y
from (X:$X)phone(Y:$Y)

retrieves all resources and values of triples with property phone, or any of its subproperties, and their classes. Note that these classes may not coincide with the defined domain and range of phone, because they may be subclasses of the domain or range. For example, given

phone("949352","5041")
type("949352",lecturer)
subclass(lecturer,staffMember)
domain(phone,staffMember)
range(phone,literal)

we get

("949352",lecturer,"5041",literal)

although lecturer is not the domain of phone.

The domain and range of a property can be retrieved as follows:

select domain(@P),range(@P)
from @P
where @P=phone

For more details see the RQL User Manual (v2.0) (2003).

3.10 Summary

- RDF provides a foundation for representing and processing metadata.
- RDF has a graph-based data model. Its key concepts are resource, property, and statement. A statement is a resource-property-value triple.
- RDF has an XML-based syntax to support syntactic interoperability. XML and RDF complement each other because RDF supports semantic interoperability.
- RDF has a decentralized philosophy and allows incremental building of knowledge, and its sharing and reuse.
- RDF is domain-independent. RDF Schema provides a mechanism for describing specific domains.
• RDF Schema is a primitive ontology language. It offers certain modelling primitives with fixed meaning. Key concepts of RDF Schema are class, subclass relations, property, subproperty relations, and domain and range restrictions.

• There exist query languages for RDF and RDFS.

Some points that will be discussed in the next chapter:

• RDF Schema is quite primitive as a modelling language for the Web. Many desirable modelling primitives are missing.

• Therefore we need an ontology layer on top of RDF/RDFS.

**Suggested Reading**

The following are some official online documents:


• F. Manola and E. Miller, eds. RDF Primer. <http://www.w3.org/TR/rdf-primer/>.


Here are some further useful readings:


An extensive list of tools and other resources is maintained at:

• <http://www.ilrt.bris.ac.uk/discovery/rdf/resources/>.

• <http://www.w3.org/RDF>

Exercises and Projects

3.1 Read the RDFS namespace and try to understand the elements that were not presented in this chapter.

3.2 Read the manual on RQL, focusing on points not discussed here.

3.3 The RDFS specification allows more than one domain to be defined for a property and uses the union of these domains. Discuss the pros and cons of taking the union versus taking the intersection of domains.

3.4 In an older version of the RDFS specification, rdfs:subClassOf was not allowed to have cycles. Try to imagine situations where a cyclic class relationship would be beneficial. (Hint: Think of equivalence between classes.)

3.5 Discuss the difference between the following statements, and draw graphs to illustrate the difference:
X supports the proposal; Y supports the proposal; Z supports the proposal.
The group of X, Y, and Z supports the proposal.

Draw graphs to illustrate the difference.

3.6 Compare rdfs:subClassOf with type extensions in XML Schema.

3.7 Consider the formal specification of rdf:_n in the axiomatic semantics. Does it capture the intended meaning of rdf:_n as the selector of the nth element of a collection? If not, suggest a full axiomatization.

3.8 Prove the inferred formulas at the end of section 3.7 using the previous axioms.

3.9 Discuss why RDF/S does not allow logical contradictions: any RDF/S document is consistent, thus it has at least one model.

3.10 Try to map the relational database model on RDF.

3.11 Compare entity-relationship modelling to RDF.

3.12 Model part of a library in RDF Schema: books, authors, publishers, years, copies, dates, and so on. Then write some statements in RDF, and query them using RQL.

3.13 Write an ontology about geography: cities, countries, capitals, borders, states, and so on.

3.14 In chapter 2 you were asked to consider various domains and develop appropriate vocabularies for them. Try to model these domains by defining suitable classes and properties, and a conceptual model. Then write sample statements in RDF.

In the following you are asked to think about limitations of RDFS, specifically, what should actually be expressed, and whether it can be represented in RDF Schema. These limitations will be relevant in chapter 4, where we present a richer modelling language.

3.15 Consider the classes of males, and females. Name a relationship between them that should be included in an ontology.

3.16 Consider the classes of persons, males and females. Name a relationship between all three that should be included in an ontology. Which part of this relationship can be expressed in RDF Schema?
3.17 Suppose we declare Bob and Peter to be the father of Mary. Obviously there is a semantic error here. How should the semantic model make this error impossible?

3.18 What relationship exists between “is child of” and “is parent of”?

3.19 Consider the property *eats* with domain *animal* and range *animal or plant*. Suppose we define a new class *vegetarian*. Name a desirable restriction on *eats* for this class. Do you think that this restriction can be expressed in RDF Schema by using *rdfs:range*?

3.20 Evaluate some RQL queries against the RDF repositories that are available at <http://sesame.aduna.biz>.

3.21 Construct an RDF Schema vocabulary on a topic of your choice, and use the FRODO RDFSViz visualisation tool² to construct a class and property diagram for your vocabulary.

² <http://www.dfki.uni-kl.de/frodo/RDFSViz/>