3 The Building Block of the Semantic Web: RDF

3.1 OVERVIEW: WHAT IS RDF?

In the first two chapters, we established an initial understanding of the Semantic Web. A typical Web page in the Semantic Web world looks like the one shown in Figure 3.1.

In Figure 3.1, the markup document is the special file we mentioned repeatedly in Chapter 2. It asserts the following fact: the semantics of the term SLR used on this page is the same as that defined in the `mySimpleCamera.owl` vocabulary. This file is indeed quite special; its existence has turned the crawler into a smart agent. When the crawler reaches this page, it follows the link on it to locate this markup file and, furthermore, it is able to "understand" it. With the help of the `mySimpleCamera.owl` file (pointed to by a link in the markup file), the crawler is finally able to understand what this whole page is about. In fact, in Chapter 2 we saw some fairly complex inferences made by the crawler with the help of the markup file.

Obviously, this special file is the key; this file, when understood by the computer (i.e., the agent), triggers the rest of the chain of inferences and understanding. It is the starting point for making the vision of the Semantic Web a reality. Therefore, the two key questions we will have to address are as follows: First, what is this special file? Second, how is it created?

The answer to the first question is simple: this special file can also be called a markup file. It is a file that describes some facts about the underlying Web page. Does this remind you of metadata? Yes, it defines metadata about the Web page; it is a metadata file. To some extent, the Semantic Web is all about metadata.

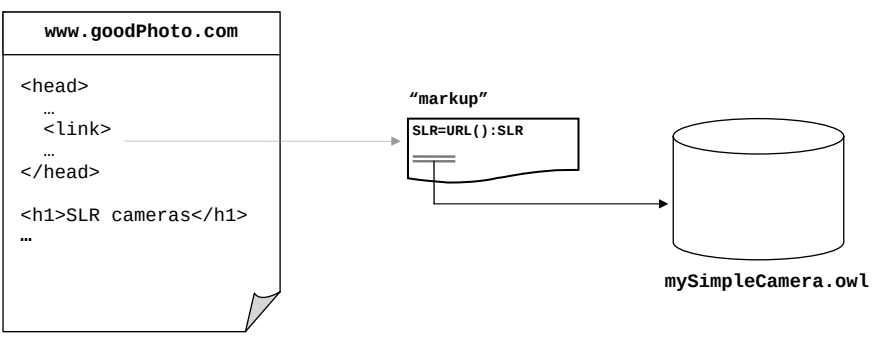


FIGURE 3.1 A Web page in the Semantic Web environment.

The answer to the second question leads to the concept of RDF (Resource Description Framework); to create a markup file, one can use RDF language; that is why we need RDF. This entire chapter concentrates on RDF.

We already know how important metadata is to the Semantic Web, and that RDF is the language we use to construct these metadata files; we therefore realize the importance of RDF. Let us now summarize the key points as follows:

- RDF is the basic building block for supporting the Semantic Web.
- RDF is to the Semantic Web what HTML has been to the Web.

Now that we know how RDF fits into the whole picture, let us take a look at what is RDF. RDF is an XML-based language for describing information contained in a Web resource. A Web resource can be a Web page, an entire Web site, or any item on the Web that contains information in some form. In this chapter, we will learn the following facts about RDF:

- RDF is a language recommended by W3C [13], and it is all about metadata.
- RDF is capable of describing any fact (resource) independent of any domain.
- RDF provides a basis for *coding*, *exchanging*, and *reusing* structured metadata.
- RDF is structured; i.e., it is machine-understandable. Machines can do useful operations with the knowledge expressed in RDF.
- RDF allows *interoperability* among applications exchanging *machine-understandable* information on the Web.

After reading this chapter, all the preceding facts will become clear to you. I suggest you review these points after finishing this chapter; they will start to make sense. If not, you will need to review this chapter again, because RDF is indeed the building block of the Semantic Web.

3.2 THE BASIC ELEMENTS OF RDF

There are several basic elements you need to know about RDF. They are the prerequisites to learning RDF. Let us discuss them first.

3.2.1 RESOURCE

The first key element is the resource. RDF is a standard for metadata; i.e., it offers a standard way of specifying data about something. This something can be anything, and in the RDF world we call this something, *resource*.

A resource therefore is anything that is being described by RDF expressions. It can be a Web page, part of a Web page (a word on a page, for instance), the whole Web site, or even a real-world object, such as a book, a human being, a dog — it can be anything.

A resource is identified by a uniform resource identifier (URI), and this URI is used as the name of the resource. Why do we have to use a URI as the name of the resource? The reason is summarized in the following rule:

> Rule #1: The name of a resource must be global. In other words, if you have a doubt that someone else might also use the same name to refer to something else, then you cannot use that name.

Think about this. If you and someone else happen to use the same name to identify two different resources, then the same name could have two different meanings. This semantic ambiguity is exactly what we want to avoid in the world of the Semantic Web.

URI is not something new. For example, URL (uniform resource locator) is a particular type of URI used in the Web world, such as `www.w3c.org`. When used to identify resources, URIs can take the same format as the URLs. The main reason is that the domain name used in the URL is guaranteed to be unique; therefore, the uniqueness of the resource is guaranteed — here, the domain-name part is used just as a namespace. In other words, there may or may not be an actual Web site at that address, and it does not matter at all. What matters is that the resource is uniquely identified globally.

Here is an example. The following URI uniquely identifies a resource:

```
http://www.yuchen.net/photography/SLR#Nikon-D70
```

Let us understand more about this resource:

1. This resource is a real-world object, i.e., a Nikon D70 camera; it is a single lens reflex (SLR) camera.
2. URL "`http://www.yuchen.net/photography/SLR`" is used as the first part of the URI. More precisely, it is used as a namespace to guarantee that the underlying resource is uniquely identified; this URL may or may not exist.
3. At the end of the namespace, "#" is used as the fragment identifier symbol to separate the namespace from the local resource name, i.e., Nikon-D70.
4. Now the namespace + "#" + localResourceName gives us the final URI for the resource; it is globally named.

Let us move on to the next basic element, property.

3.2.2 Property

Property is a resource that has a name and can be used as a property; i.e., it can be used to describe some specific aspect, characteristic, attribute, or relation of the given resource. As we have already studied the concept of resource, it is not at all hard to understand property. The following is an example of a property:

```
http://www.yuchen.net/photography/SLR#weight
```

This property describes the weight of the D70 camera — you certainly do not want to carry something very heavy when you prefer to have your camera with you all the time to ensure you do not miss those important moments!

3.2.3 Statement

An RDF statement is used to describe properties of resources. It has the following format:

```
resource (subject) + property (predicate) + property value (object)
```

The property value can be a string literal or a resource. Therefore, in general, an RDF statement indicates that a resource (the subject) is linked to another resource (the object) via an arc labeled by a relation (the predicate). It can be interpreted as follows:

```
<subject> has a property <predicate>, whose value is <object>
```

For example:

```
http://www.yuchen.net/photography/SLR#Nikon-D70 has a
http://www.yuchen.net/photography/SLR#weight whose value
is 1.4 lb.
```

This is certainly clear, but the drawback is that it is too long. Let us define the following namespace:

```
xmlns:mySLR="http://www.yuchen.net/photography/SLR#"
```

The statement can be rewritten in a much shorter form:

```
mySLR:Nikon-D70 has a mySLR:weight whose value is 1.4 lb.
```

As you might have already noticed, any RDF statement can be expressed as a triple (presented in a table format). Before we show an example, let us introduce rule #2 first:

> Rule #2: Knowledge (or information) is expressed as a statement in the form of subject, predicate, and object, and this order should never be changed.

For example, following rule 2, the preceding statement can be expressed in the triple format, as shown in Table 3.1. This can also be expressed in a graph model, as shown in Figure 3.2.

Up to this point, we have covered the basic components of RDF, and the rest are RDF syntax issues. It is also interesting to see, from the preceding examples, how knowledge is coded in the RDF statement. In fact, just by using these triples, the computer has already gained much more inference power than you might realize. We will show you a small example before we get into the syntax details.

TABLE 3.1
An RDF Triple Expressed in a Table Format

Subject	Predicate	Object
mySLR:Nikon-D70	mySLR:weight	1.4 lb

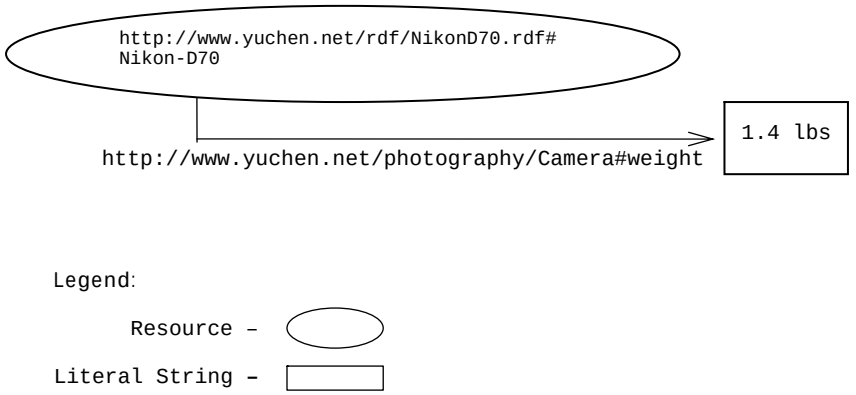


FIGURE 3.2 RDF graph model.

TABLE 3.2
A Set of RDF Statements

Subject	Predicate	Object
mySLR:Nikon-D70	mySLR:weight	1.4 lb
mySLR:Nikon-D70	mySLR:pixel	6.1 M
mySLR:Nikon-D50	mySLR:weight	1.3 lb

3.3 RDF TRIPLES: KNOWLEDGE THAT MACHINES CAN USE

Let us take a detour here, just to see how RDF statements can be used to express knowledge, and based on these simple statements, what kind of inference power a machine can have.

Let us assume that we have the following statements (again, use `mySLR` namespace), as shown in Table 3.2. Our first impression is that Table 3.2 looks like a table in a database. In fact, RDF triples can be stored in a database file. Now, let us ask the machine the following question:

> What properties did we define in order to describe Nikon D70?

We can express the question using the following RDF format:

```
question.subject = mySLR:Nikon-D70
question.predicate = mySLR:*;
```

Note that `mySLR:*` is used as a wild card. The pseudocode in List 3.1 can help the computer answer the question. This code will present the following answer:

```
mySLR:weight
mySLR:pixel
```

LIST 3.1
Pseudocode Used by the Computer to Draw Inferences Based on Table 3.2

```
// format my question
question.subject   = mySLR:Nikon-D70
question.predicate = mySLR:*;

// read all the RDF statements and store them in some array
rdfStatement[0].subject   = mySLR:Nikon-D70;
rdfStatement[0].predicate = mySLR:weight;
rdfStatement[1].subject   = mySLR:Nikon-D70;
rdfStatement[1].predicate = mySLR:pixel;
rdfStatement[2].subject   = mySLR:Nikon-D50;
rdfStatement[2].predicate = mySLR:weight;

// answer the question!
foreach s in rdfStatement[] {

   if ( (s.subject==question.subject || question.subject=='*') &&
      (s.predicate==question.predicate || question.predicate ==
         '*') )
   {
      System.out.println(s.predicate.toString());
   }
};
```

This means that we have defined `mySLR:weight` and `mySLR:pixel` properties for Nikon D70. Clearly, based on the knowledge presented in the RDF statements (Table 3.2), the machine can indeed perform some useful work for us.

In fact, you can construct more interesting examples than the one shown here by adding more RDF statements and more complex predicates and objects. We will see more examples along these lines in subsequent chapters.

3.4 A CLOSER LOOK AT RDF

The fact that RDF is the basic building block of the Semantic Web demands a more detailed study of RDF itself. In fact, to understand the rest of the book, one needs a solid grasp of RDF. In this section, we will first study basic RDF constructs, including the fundamental syntax and most of the frequently used words from the RDF vocabulary. This will not only teach you RDF, but will also clarify most of the common confusions about RDF. At the end of this section, a summary of the fundamental rules in the world of RDF will be presented.

3.4.1 Basic Syntax and Examples

Now, we will discuss RDF syntax. Here, we have both good news and bad news.

Let us consider the bad news first. This is, after all, a new language to be learned, and it does take a while to get used to it. Indeed, there are some issues associated with the syntax that can be very confusing before you completely understand them. In this section, one of the goals is to clarify the confusing issues by walking you through several examples.

Now for the good news: First, after you have read this section, you would be very comfortable with RDF documents and literature about RDF, because we are going to cover all the confusing topics in RDF. Second, RDF does not have a large vocabulary set at all. In fact, it is extremely small. More precisely, the RDF vocabulary consists of the following names:

- Syntax names: `RDF`, `Description`, `ID`, `about`, `parseType`, `resource`, `li`, `nodeID`, `datatype`.
- Class names: `Seq`, `Bag`, `Alt`, `Statement`, `Property`, `XMLLiteral`, `List`.
- Property names: `subject`, `predicate`, `object`, `type`, `value`, `first`, `rest_`*n* (where *n* is a decimal integer greater than zero with no leading zeros).
- Resource names: `nil`.

Keep these names in mind, and once we reach the end of this section you will see how many of these names we have covered. Let us proceed.

Refer back to our earlier example, "Nikon-D70 has a weight of value 1.4 lb." For the RDF version of this fact, see List 3.2.

LIST 3.2
First Example of an RDF Document

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns="http://www.yuchen.net/photography/Camera#">
3:   <SLR rdf:ID="Nikon-D70">
4:      <weight>1.4 lbs</weight>
5:   </SLR>
6: </rdf:RDF>
```

This simple document describes a resource (Nikon-D70 camera) in RDF format. Let us understand it line by line.

The first line says this document is in XML format (I assume you are comfortable with XML). The second line further indicates that this document is an RDF document by using the RDF keyword `RDF`. It also shows the RDF namespace URI reference, i.e., `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, and `rdf` is used as a shortcut to represent this namespace. After establishing this namespace and its shortcut, a name from this namespace will be expressed as `rdf:name`, using the shortcut as the prefix. For instance, `rdf:RDF` uses the `RDF` name from the RDF vocabulary, and it has an RDF URI reference constructed by the concatenation of RDF namespace,

URI reference, and name; i.e., `rdf:RDF` has the RDF URI reference `http://www.w3.org/1999/02/22-rdf-syntax-ns#RDF`.

The second line defines another namespace, a namespace created by us, namely, `http://www.yuchen.net/photography/Camera#`, and it is used as the default namespace. Therefore, any name that does not have a prefix in this document is assumed to be in this namespace. For example, the keyword `SLR` on the third line is defined in this namespace.

The third line uses the RDF keyword `rdf:ID` to identify the resource being described by this RDF document; this resource is called Nikon-D70. The term `SLR` defines the class (type) of this resource. What exactly is an `SLR`? We do not know yet, but we do know that it is defined in the default namespace. We can interpret the third line as follows:

> The resource being described in this document is identified as Nikon-D70; it is an instance of the class SLR, which is further defined in the namespace `http://www.yuchen.net/photography/Camera`.

The fourth line specifies that the `SLR` class has one property, whose name is `weight`, and for this resource (`Nikon-D70`), the value of this property is `1.4 lb`.

The rest of the document is easy. Now, to put all these together, we can interpret this RDF code as follows:

> The RDF document describes a resource whose name is Nikon-D70; it is an instance of the class SLR, and its weight is 1.4 lb.

Up to this point, we have covered the following RDF vocabulary: `rdf:RDF`, `rdf:ID`.

Now, let us put the aforementioned RDF document into the subject-predicate-object format:

subject: `http://www.yuchen.net/photography/cameras#Nikon-D70`
predicate: `http://www.yuchen.net/photography/Camera#weight`
object: `1.4 lb`

This is what we would expect to see, but it is wrong; it could be one of the reasons why RDF appears very confusing at the beginning. In fact, RDF/XML prescribes that the statement look like this:

subject: `http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-D70`
predicate: `http://www.yuchen.net/photography/Camera#weight`
object: `1.4 lb`

The URL `http://www.yuchen.net/rdf/NikonD70.rdf` is in fact the location of this RDF document. How can this URL become part of the subject? This is because the complete URI of the subject is obtained by concatenating the following three pieces together:

```
in-scope base URI + "#" + rdf:ID value
```

Because the in-scope base URI is not explicitly stated in the RDF document, it is provided by the parser based on the location of the file in which it was parsed. In this example, `http://www.yuchen.net/rdf/NikonD70.rdf` is the location of the document; therefore, the URI of the subject is constructed as follows: `http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-D70`.

Clearly, using `rdf:ID` results in a relative URI for the subject; the URI changes if the location of the RDF document changes. This seems to contradict the very meaning of URI — the unique and global identifier of a resource. How can it change based on the location of some file, then? In most cases, we have an absolute URI in mind for the resources described in the file.

The best solution to this problem is to use `rdf:about` instead of `rdf:ID`. Let us discuss this solution later, because quite often you may see people using `xml:base` to solve this problem. So, let us take a look at this solution first, just to prepare you for further literature.

More specifically, by placing the `xml:base` attribute in the RDF document, we will be able to control which base is used to resolve the `rdf:ID` value; the subject of the statement will then be fixed and it will be generated using the following mechanism:

```
xml:base + "#" + rdf:ID value
```

List 3.3 shows the new RDF document.

LIST 3.3
RDF Document Using `xml:base` Attribute

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns="http://www.yuchen.net/photography/Camera#"
           xml:base="http://www.yuchen.net/rdf/NikonD70.rdf">
3:   <SLR rdf:ID="Nikon-D70">
4:      <weight>1.4 lbs</weight>
5:   </SLR>
6: </rdf:RDF>
```

As we have mentioned earlier, however, `rdf:about` should always be used as the best solution. It provides an absolute URI for the resource, and that URI is taken verbatim as the subject; this certainly avoids all possible confusions. List 3.4 shows the document when `rdf:about` is used.

LIST 3.4
RDF Document Using `rdf:about`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns="http://www.yuchen.net/photography/Camera#">
```

```
3:   <SLR rdf:about="http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-
      D70">
4:      <weight>1.4 lbs</weight>
5:   </SLR>
6: </rdf:RDF>
```

Up to this point, we have covered the following RDF vocabulary: `rdf:RDF`, `rdf:ID`, `rdf:about`.

Next we will cover `rdf:type`, `rdf:Description`, and `rdf:resource`.

In our RDF document, we have described a resource called `Nikon-D70`, and it is an instance of the class `SLR`. This relationship is expressed by using `SLR` in the document together with the namespace where `SLR` is defined. Take a look at Figure 3.2; obviously, it fails to express the relationship. A revised version should appear as in Figure 3.3.

As shown in Figure 3.3, for the subject node `Nikon-D70` we need a `has-type` predicate to indicate that the underlying resource is an instance of some class. It is not hard to imagine that this requirement should be very common for RDF graph models.

In the RDF vocabulary, `rdf:type` exists for this reason. It is used to describe resources as instances of specific types or classes. In other words, subject nodes can have `rdf:type` predicates coming out from them, indicating that they are instances of some type. These nodes are conventionally called *typed nodes* in a graph, or *typed node elements* in RDF documents. Using `rdf:type`, Figure 3.3 should look as shown in Figure 3.4.

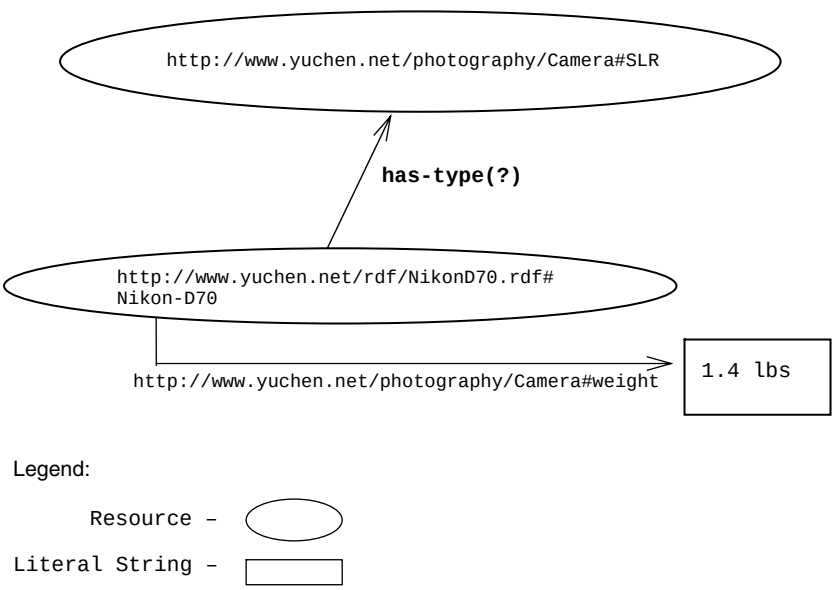


FIGURE 3.3 A revised RDF graph model.

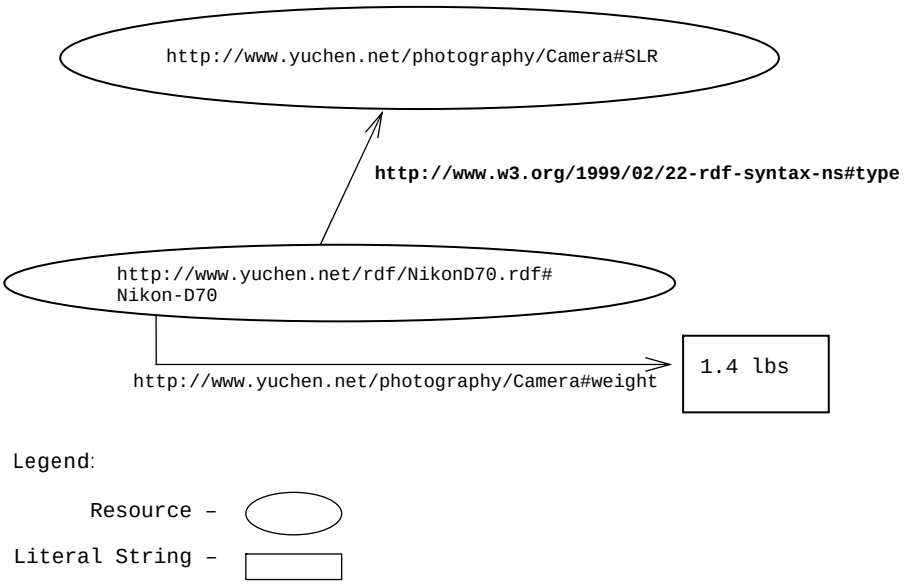


FIGURE 3.4 The final RDF graph model.

To use `rdf:type` in the example RDF document is a little more complex. Actually, the current document (as the one in List 3.4, for example) has been using a so-called "shorthand" form of the RDF expression. It is also called *concise typed node element* form. The full version is a combination of `rdf:Description`, `rdf:resource`, and `rdf:type`, as shown in List 3.5.

LIST 3.5
RDF Document Using `rdf:type`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.yuchen.net/photography/Camera#">
3:    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
          NikonD70.rdf#Nikon-D70">
4:       <rdf:type rdf:resource="http://www.yuchen.net/
            photography/Camera#SLR"/>
5:     <weight>1.4 lbs</weight>
6:   </rdf:Description>
7: </rdf:RDF>
```

This version is called the *long form*, and it appears more often in the literature than the abbreviated form. We can now interpret it as follows:

This RDF statement describes the following resource:

`http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-D70`

This resource is an instance of the following type (class):

`http://www.yuchen.net/photography/Camera#SLR`

The `http://www.yuchen.net/rdf/NikonD70.rdf#Nikon-D70` resource has a weight of 1.4 lb.

As a summary, it is always a good practice to use `rdf:about`, `rdf:Description`, and `rdf:resource`. Also, whenever you expect to see

```
<rdf:Description ...
```

and instead you see

```
<ns:className rdf:about ...
```

you should realize that it is the abbreviated form for

```
<rdf:Description rdf:about="...">
    <rdf:type resource="&ns;className"/>
...
```

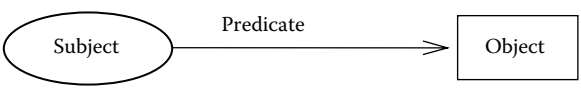
where `&ns;` is the namespace URI bound to the `ns` prefix.

> Up to this point, we have covered the following RDF vocabulary: `rdf:RDF`, `rdf:ID`, `rdf:about`, `rdf:type`, `rdf:Description`, `rdf:resource`.

3.4.2 Literal Values and Anonymous Resources

Before we go deeper into the syntax of RDF, let us again review the basic RDF statement structure, as shown in Figure 3.5. Keeping this structure in mind, let us remember the rule that the property value must be literal or a resource. In our previous example, the property `weight` had a literal value of "`1.4 lb.`"

However, given that the Web itself is such a global resource, it might not be a good idea to use a literal value such as `1.4 lb`; when we do this, we assume that anyone who accesses this property will be able to understand the unit that is being



equivalent to:

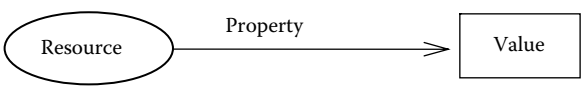


FIGURE 3.5 RDF statement model.

used. This assumption is not always safe. For instance, someone who is not from the United States would assume the weights are expressed in kilograms.

A better solution is to explicitly express the value and the unit in separate property values. In other words, the value of the weight property will have two components, the literal for the decimal value and an indication of the unit of measure (e.g., pounds). In this situation, the decimal value acts as the main value of the weight property, and the unit component exists just to provide additional contextual information that qualifies the main value.

How do we then implement this? The solution is to model such a qualified property as another kind of structured value. More specifically, a completely separate resource could represent this structured value as a whole and be used as the object of the original statement (given the rule that the property can be another resource). This new resource can have properties representing the individual components of the structured value. In our example, it should have two properties: one for the decimal value and the other for the unit.

RDF provides a predefined `rdf:value` property to describe the main value of a structured value. Therefore, in our example, the decimal could be given as the value of the `rdf:value` property, and another resource should be used as the value of the unit property, as shown in List 3.6. Now the property `weight` will have a resource as its value. This resource, as we have discussed earlier, has two properties: the first is the predefined `rdf:value` property, whose value is `1.4`; the second is the `units` property, defined in the `uom` namespace. What is the value of the `uom:units` property? Well, interestingly enough, it uses another resource as its value. Here, we assume that this resource, whose URI is `http://www.something.org/units#lbs`, is already defined by someone else.

LIST 3.6
RDF Document Using `rdf:value`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:uom="http://www.standards.org/measurements#"
4:          xmlns="http://www.yuchen.net/photography/Camera#">
5:   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         NikonD70.rdf#Nikon-D70">
6:      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
7:      <weight>
8:        <rdf:Description>
9:           <rdf:value>1.4</rdf:value>
10:            <uom:units rdf:resource="http://www.something.org/
                   units#lbs"/>
11:       </rdf:Description>
12:     </weight>
13:   </rdf:Description>
14: </rdf:RDF>
```

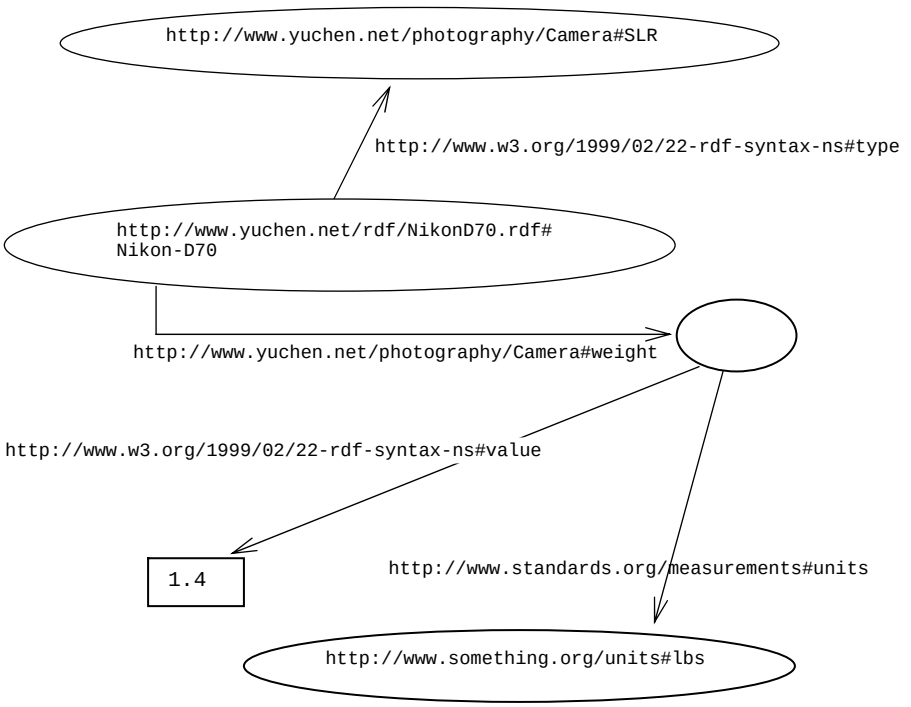


FIGURE 3.6 Use anonymous resource as the value of property `weight`.

Another important point to note is that the property `weight` does indeed have a resource as its value, but what is the name of this resource? In the example, this resource is defined using lines 8 to 11. On line 8, the `<rdf:Description>` tag does not have anything like `rdf:ID` or `rdf:about`. This resource is an *anonymous* resource.

Why is the resource used by the `weight` property made anonymous? This is because its purpose is just to provide a context for the two properties to exist. Other RDF documents will have no need to use or add any new details to this resource. Therefore, there is no need to give this resource an identifier.

An anonymous resource is also called a *blank node*. This fact becomes clearer in Figure 3.6, which describes the foregoing example using an RDF graph. In the graph, the anonymous resource is represented by the blank node. The RDF parsers will normally generate a unique identifier for anonymous resources just to distinguish one anonymous resource from another; it is mainly an internal usage within the parsers.

In RDF models, there is an easier way to implicitly create a blank node. This is considered to be a shorthand method provided by RDF that involves the usage of `rdf:parseType` keyword, as shown in List 3.7.

List 3.7 is identical to List 3.6. `rdf:parseType="Resource"` in line 7 is used as the attribute of the `weight` element, and it indicates to the RDF parser that the contents of the `weight` element (lines 8 and 9) should be interpreted as the description of a new resource (a blank node), and should be treated as the value of property `weight`. Without seeing a nested `rdf:Description` tag, the RDF parser creates a

LIST 3.7
RDF Document Using `rdf:parseType`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:uom="http://www.standards.org/measurements#"
4:          xmlns="http://www.yuchen.net/photography/Camera#">
5:    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
           NikonD70.rdf#Nikon-D70">
6:      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
7:        <weight rdf:parseType="Resource">
8:          <rdf:value>1.4</rdf:value>
9:          <uom:units rdf:resource="http://www.something.org/
                units#lbs"/>
10:       </weight>
11:   </rdf:Description>
12: </rdf:RDF>
```

blank node as the value of the `weight` property, and then uses the enclosed two elements as the properties of that blank node, which is exactly what we wish the parser to accomplish.

Another way to represent a blank node is to use the so-called blank node identifier; as this is also quite a popular approach, let us include it in this section. Basically, we assign a blank node identifier to each blank node that we have in the RDF model. However, this identifier serves to identify the blank node within a particular RDF document; it is completely unknown outside the scope of the document. Compared to the URI of a named resource, the URI always remains visible outside the document in which it is assigned.

This blank node identifier method uses the RDF keyword `rdf:nodeID`. More specifically, a statement using a blank node as its subject should use an `rdf:Description` element with an `rdf:nodeID` attribute instead of an `rdf:about` or `rdf:ID` attribute. A statement using a blank node as its object should use a property element with an `rdf:nodeID` attribute instead of an `rdf:resource` attribute. List 3.8 shows the details. So much for the blank node, but let us remember that it is quite handy and also frequent in many RDF documents; so ensure you are familiar with it.

LIST 3.8
RDF Document Using `rdf:nodeID`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:uom="http://www.standards.org/measurements#"
4:          xmlns="http://www.yuchen.net/photography/Camera#">
5:    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
           NikonD70.rdf#Nikon-D70">
```

```
6:      <rdf:type rdf:resource="http://www.yuchen.net/photography/
            Camera#SLR"/>
7:        <weight rdf:nodeID="youNameThisNode"/>
8:    </rdf:Description>
9:    <rdf:Description rdf:nodeID="youNameThisNode">
10:      <rdf:value>1.4</rdf:value>
11:      <uom:units rdf:resource="http://www.something.org/
             units#lbs"/>
12:   </rdf:Description>
13: </rdf:RDF>
```

The last issue in this section is about the "typed" literal. In List 3.8, in line 10 we use `1.4` as the value of the `rdf:value` property. Here, `1.4` is a plain "untyped" literal, and only we know that the intention is to treat it as a decimal number; there is no information in List 3.8 that explicitly indicates this.

Now, let us use the `rdf:datatype` keyword to provide information about how to interpret the type of a given literal value. This is the so-called typed literal, and you might also assume that RDF would have to provide a set of data type definitions. Interestingly, RDF does not have such a data type system of its own, such as data types for integers, real numbers, strings, dates, etc. It borrows an external data type system and uses the `rdf:datatype` tag to explicitly indicate which external data type system the RDF document is using.

The current practice is to use XML schema data types. The reason is that because XML enjoys such great success, its schema data types would most likely be interoperable among different software agents. RDF documents can certainly use other data type systems, provided that the software systems could process these sets of data types as well.

Now, let us use `rdf:datatype` to clearly indicate that the value `1.4` should be treated as a decimal value, as shown in List 3.9. In line 9, property `rdf:value` now has an attribute named `rdf:datatype` whose value is the URI of the data type. In our example, this URI is `http://www.w3.org/2001/XMLSchema#decimal`. The result is the value of the `rdf:value` property, namely, `1.4`, will be treated as a decimal value as defined in the XML schema data types.

LIST 3.9
RDF Document Using `rdf:datatype`

```
1: <?xml version="1.0"?>
2: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:          xmlns:uom="http://www.standards.org/measurements#"
4:          xmlns="http://www.yuchen.net/photography/Camera#">
5:    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
           NikonD70.rdf#Nikon-D70">
6:      <rdf:type rdf:resource="http://www.yuchen.net/photography/
            Camera#SLR"/>
7:      <weight>
8:         <rdf:Description>
```

```
9:            <rdf:value rdf:datatype=
           "http://www.w3.org/2001/XMLSchema#decimal">1.4</rdf:value>
10:           <uom:units rdf:resource="http://www.something.org/
                   units#lbs"/>
11:        </rdf:Description>
12:     </weight>
13:   </rdf:Description>
14: </rdf:RDF>
```

Note that there is no absolute need to use `rdf:value` in this example. A user-defined property name, such as `weightAmount`, could have been used instead of `rdf:value`, and the `rdf:datatype` attribute can still be used together with this user-defined property. In fact, RDF does not associate any special meaning with `rdf:value`; it is simply provided as a convenience for use in the cases described in our example.

Also note that because the URI `http://www.w3.org/2001/XMLSchema#decimal` is used as an attribute value, it has to be written out fully; it cannot be abbreviated. However, this makes the line quite long, which might hurt readability in some cases. To improve readability, some RDF documents would use XML entities. Recall that an XML entity can associate a name with a string of characters, and this name can be referenced anywhere in the XML document. When XML processors reach such a name, they will replace it with the string of characters that normally represents the real content. As we can make the name really short, this enables us to abbreviate the long URI. To declare the entity, we can do the following:

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
```

A reference name `xsd` is defined here to be associated with the namespace URI that contains the XML schema data types. We can use `&xsd:` (note the "`:`"; it is necessary) anywhere in the RDF document to represent the preceding URI. Using this abbreviation, we have the following more readable version, as shown in List 3.10.

LIST 3.10
A More Readable RDF Document

```
1: <?xml version="1.0"?>
2: <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
     XMLSchema#">]>
3: <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4:          xmlns:uom="http://www.standards.org/measurements#"
5:          xmlns="http://www.yuchen.net/photography/Camera#">
6:   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         NikonD70.rdf#Nikon-D70">
7:     <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8:       <weight>
9:         <rdf:Description>
10:           <rdf:value rdf:datatype="&xsd;decimal">1.4</rdf:value>
```

```
11:           <uom:units rdf:resource="http://www.something.org/
                   units#lbs"/>
12:        </rdf:Description>
13:     </weight>
14:   </rdf:Description>
15: </rdf:RDF>
```

In the aforementioned example, `xsd` is declared on line 2 and used on line 10. This is quite common in RDF documents, so ensure that you are comfortable with it.

> Up to this point, we have covered the following RDF vocabulary: `rdf:RDF`, `rdf:ID`, `rdf:about`, `rdf:type`, `rdf:Description`, `rdf:resource`, `rdf:value`, `rdf:parseType`, `rdf:nodeID`, `rdf:datatype`.

We have now covered the most frequently used RDF syntax. This section does not intend to be a full tutorial of the RDF language; however, we have covered enough material already, so you can not only understand the rest of the book but also read more about RDF on your own. There are certainly other capabilities provided by RDF. In the next section, we will discuss them very briefly so you can see how they fit into the whole picture of RDF.

3.4.3 Other RDF Capabilities

The first such capability is the RDF containers. They are provided by RDF to describe groups of things: for instance, all the SLR cameras produced by Nikon. The following three types of containers are provided by RDF using a predefined container vocabulary:

- `rdf:Bag`
- `rdf:Seq`
- `rdf:Alt`

A resource can have type `rdf:Bag`. In this case, it represents a group of resources or literals, such as all the SLR cameras produced by Nikon. The order of these members is not significant; you only care about the whole group, not their individual attributes, such as the release date of each model.

An `rdf:Seq` is the same as the `rdf:Bag`, except that the order is indeed significant. For instance, if we also want to know the release date of each Nikon SLR camera, we will have to represent them using `rdf:Seq`.

`rf:Alt` is also a container; however, items in the container are alternatives. For instance, you can use this container to describe a set of flight legs, each one of them from Atlanta to Honolulu but at different times; and you really just need one of them.

The problem with the RDF containers is that the containers are always open. To be more precise, a container just claims the identified resources are members; it never excludes other resources as members. For instance, if one RDF document includes some members, there might be another RDF document that adds some other

members to the same resource. This could be a very serious problem. To solve this problem, we have the second RDF capability: RDF collection.

RDF uses the collection construct to describe a group that contains only the specified resources as members. Its vocabulary includes the following keywords:

- `rdf:first`
- `rdf:rest`
- `rdf:List`
- `rdf:nil`

For specific examples, check with the online RDF documents at `www.w3.org`.

There are other RDF capabilities, such as RDF reification and XML Literals (some authors suggest that we limit the use of XML Literals), which I leave to the readers to explore; let us move on to more exciting topics about RDF.

3.5 FUNDAMENTAL RULES OF RDF

We have covered most of the topics about RDF, and it is time for us to summarize some basic RDF rules. There are three basic rules; two of them have already been presented in previous sections and should be self-evident. However, the third rule may need more explanation; in fact, it is very important and we will devote another section just to it. Let us take a look at the rules first:

> Rule #1: The name of a resource must be global. In other words, if you doubt that someone else might use the same name to refer to something else, then you cannot use the same name.

Now we should have a better understanding of this rule. All the three basic components of the RDF model — subject, predicate, and object — can be resources, and this rule states that you have to use a URI to identify them (except for an anonymous resource). The whole point is to ensure that the following is always true: if two or more RDF documents use the same URI to describe a resource, then this resource represents exactly the same concept in the real world.

However, assuming that everyone uses URIs to globally identify subjects, predicates, and objects, matters can still (and very likely) go awry; two different RDF documents can still use different URIs to refer to the same thing or concept, and the authors of these documents could be unaware of the fact that the same concept has already been named in the other document. To avoid reinventing the wheel, it is always a good idea to search around and use the resources from existing vocabularies, if possible, instead of making up new URIs every time. Therefore, using URIs to globally and uniquely identify resources in RDF statements supports and promotes the development and use of shared vocabularies on the Web. In fact, domain-specific vocabularies are being developed constantly. As we will see later in this book, this shared and common understanding of the concepts and classes is the key to Semantic Web applications.

Rule #2: Knowledge (or information) is expressed as a statement in the form of subject-predicate-object, and this order should never be changed.

This rule plays an important role in enabling machines to understand the knowledge expressed in RDF statements. Before we get into the details, let us take a look at this triple pattern once more.

Because the value of a property can be a literal or a resource, a given RDF statement can take the form of alternating sequences of resource-property, as shown in List 3.11.

LIST 3.11
The Pattern of an RDF Document

```
1:  <rdf:Description rdf:resources="#resource-0">
2:    <someNameSpace:propertyName-0>
3:       <rdf:Description rdf:resource="#resource-1">
4:          <someNameSpace:propertyName-1>
5:             <rdf:Description rdf:resource="#resource-2">
6:                <someNameSpace:propertyName-2>
7:                  ...
8:                </someNameSpace:propertyName-2>
9:             </rdf:Description>
10:         </someNameSpace:propertyName-1>
11:      </rdf:Description>
12:   </someNameSpace:propertyName-0>
13: </rdf:Description>
```

In List 3.11, `#resource-0` has a property named `propertyName-0`; its value is another resource described using lines 3 to 11 (`#resource-1`). On the other hand, `#resource-1` has a property named `propertyName-1`, whose value is yet another resource described using lines 5 to 9. This pattern can go on and on; however, the `resource-property-value` structure is never changed.

Why is this order so important? It is important because if we follow this order when we create RDF statements, an RDF-related application (agent) will be able to understand the meaning of these statements. To see this, let us study the following example:

Let `myCamera` represent `http://www.yuchen.net/photography/Camera#`. List 3.2 to List 3.10 all express the following fact:

```
myCamera:Nikon-D70 myCamera:weight 1.4
```

We, as the creator of this statement, understand its meaning. However, for an agent, the triple looks more like this:

```
$#!6^:af#@dy $#!6^:3pyu9a dcfa
```

However, the agent does understand the following:

```
$#!6^:af#@dy is the subject
$#!6^:3pyu9a is the predicate
dcfa         is the object
```

And now, here is the interesting part: the agent also has a vocabulary it can access, and the following fact is stated in this vocabulary:

```
property($#!6^:3pyu9a) is used exclusively on resource($#!6^:Af5%)
```

We will see what exactly is this vocabulary (in fact, it is called RDF schema), and we will also find out how to express the above-mentioned fact using this vocabulary in Chapter 4, but for now let us just assume that the fact is well expressed in the vocabulary.

Given all these, the agent, without really associating any special meaning with the preceding statement, can draw the following conclusion:

```
resource($#!6^:af#@dy) is an instance of resource($#!6^:Af5%)
```

When the agent displays this conclusion in the screen, it looks like this:

```
Nikon-D70 is an instance of DigitalCamera
```

It makes perfect sense. The key point here is, a given application cannot associate any special meaning with the RDF statements. However, with some extra work (the schema, for instance), the given application can act as if it does understand these statements. In fact, once we understand more about the RDF schema, we will see more of this exciting inference power in Chapter 4.

Do you want to see something exciting before going further? Well, study the next rule.

> Rule #3: The most exciting one! I can talk about resource at my will, and if I choose to use an existing URI to identify the resource I am talking about, then the following is true:
>
> 1. The resource I am talking about and the resource already identified by this existing URI represents exactly the same concept.
> 2. Everything I have said about this resource is considered to be additional knowledge about that resource.

This seems to be trivial and almost like a given. However, it can be quite powerful in many cases. Let us recall the situation in the current Web. One fact about the Internet that is quite attractive to all of us is that you can talk about anything you want and you can publish anything you want. When you do this, you can also link your document to any other page you would like to.

For example, assume that I have a small Web site on which I present several articles about digital photography. I also have linked my page to `www.goodPhoto.com`. Someone else perhaps has done the same and has a link to `www.goodPhoto.com`, too. What will this do to `www.goodPhoto.com`? Not much at all, except that some search engines will realize that quite a few pages have linked to `www.goodPhoto.com`, and therefore, the rank of this site should be upgraded. But this is pretty much all of it — the final result is still the same: the Internet is a huge distributed database, from which it is extremely hard to get information.

On the other hand, based on the preceding rule, all the RDF documents containing a resource identified by the same known URI can be connected, based on a URI that has a well-defined meaning. Although these RDF documents are most likely distributed

everywhere on the Internet, each one of them presents some knowledge about that resource, and adding them together can produce some very powerful results.

Let us discuss further details in the next section.

3.6 AGGREGATION AND DISTRIBUTED INFORMATION

3.6.1 An Example of Aggregation

As we stated in the previous section, if any RDF document mentions some resource using an existing URI, then the RDF statements will be talking about the same concept and will be adding extra information to that resource. Also, this extra data is just a small piece of knowledge that the whole Web contains.

The first thing to do then is to aggregate all these statements so we can get a closer look at the whole picture. Let us take a look at one such example.

Assume that one of my friends is also interested in Nikon D70 and he is also aware that I have created an RDF document describing this camera. After reading my document, he decides to add more predicates to describe Nikon D70 better; List 3.12 shows what he has come up with.

LIST 3.12
Another RDF Document for the Same Resource

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns:ychen="http://www.yuchen.net/photography/Camera#"
5          xmlns="http://www.yufriend.net/photography/Camera#">
6   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
        NikonD70.rdf#Nikon-D70">
7     <rdf:type rdf:resource="&yuchen;SLR"/>
8       <minSensitivity rdf:datatype="&xsd;integer">200
        </minSensitivity>
9         <maxSensitivity rdf:datatype="&xsd;integer">1600
          </maxSensitivity>
10  </rdf:Description>
11 </rdf:RDF>
```

In his document, he clearly indicates that he is describing something that already exists by using the URI that I had created, and then adds two new properties, `minSensitivity` and `maxSensitivity`, to describe the same camera.

Let us now assume that I have some aggregation tool available, which will combine the RDF document he created with the one I have. After using this tool, the new RDF document will appear as shown in List 3.13. We see that the new

information added by my friend is considered to be extra data and is added to the original RDF document. But why is this powerful? This simple example may not clearly show this; in the next section, we use a hypothetical real-world example to further illustrate this.

LIST 3.13
A New RDF Document Generated by Combining Two RDF Documents

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns:newp="http://www.yufriend.net/photography/Camera#"
6          xmlns="http://www.yuchen.net/photography/Camera#">
7   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
        NikonD70.rdf#Nikon-D70">
8      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
9      <weight>
10        <rdf:Description>
11           <rdf:value rdf:datatype="&xsd;decimal">1.4</rdf:value>
12           <uom:units rdf:resource="http://www.something.org/
                units#lbs"/>
13        </rdf:Description>
14     </weight>
15     <newp:minSensitivity rdf:datatype="&xsd;integer">200</newp:
           minSensitivity>
16     <newp:maxSensitivity rdf:datatype="&xsd;integer">1600</newp:
           maxSensitivity>
17   </rdf:Description>
18 </rdf:RDF>
```

One last point needs to be discussed here. It is clear to us by now that only a named resource can be aggregated (given the URI of this resource is a reused URI). Therefore, an anonymous resource cannot be aggregated. The reason is simple: if a resource in a document is anonymous, an aggregation tool will simply not be able to tell if this resource is talking about some resource already defined and described. This is probably one disadvantage of using anonymous resources.

3.6.2 A Hypothetical Real-World Example

To see the power of RDF aggregation, let us consider the following scenario: I would like to buy a digital SLR camera, and I am currently considering either Nikon D70 or Canon 20D — both models seem to be quite impressive, and it is hard for me to decide which one to go with.

There are three Nikon vendors and two Canon vendors in my neighborhood, and there is also a discussion group over the Internet that mainly concentrates on

reviewing different SLR models. I would like to have the following information before I can make my final decision about which camera to buy:

- How many Nikon and Canon SLRs are sold daily by each of these five vendors?
- What are the reviews of the discussion group members? I would like to read as many reviews as I can.

For a situation such as this, a database system is not a solution; these five vendors are competitors (especially if they sell different brands), and none of them will be willing to maintain a database that may give away their sales performance secrets. Also, the reviewers will simply not bother to update their database, even if one exists.

My solution is to ask each vendor to produce some RDF statements that I can use (by paying them some money? No, I have friends working for each of these vendors!); the only condition is that they need to use my URIs, instead of inventing their own. I then e-mail the discussion group, informing them the URIs and telling them that if anyone would like to provide a review, they should use the given URIs.

The two URIs that I give them are as follows:

```
<rdf:Description rdf:about="http://www.yuchen.net/rdf/NikonD70
.rdf#Nikon-D70">

<rdf:Description rdf:about="http://www.yuchen.net/rdf/Canon20D
.rdf#Canon-20D">
```

I would then use a crawler that visits the Web sites of these five vendors to search for the RDF documents. List 3.14 shows one such document about D70.

LIST 3.14
An RDF Document Collected by My Crawler (Nikon)

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns="http://www.yuchen.net/photography/Camera#">
6    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
          NikonD70.rdf#Nikon-D70">
7      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8        <itemSold rdf:datatype="&xsd;integer">12</itemSold>
9    </rdf:Description>
10 </rdf:RDF>
```

Besides the preceding document for Nikon, I have also collected one RDF statement about Canon, which is shown in List 3.15.

LIST 3.15
An RDF Document Collected by My Crawler (Canon)

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns="http://www.yuchen.net/photography/Camera#">
6   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         Canon20D.rdf#Canon-20D">
7      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8      <itemSold rdf:datatype="&xsd;integer">8</itemSold>
9   </rdf:Description>
10 </rdf:RDF>
```

About the reviews, I have no idea when someone will present a review, or if there is going to be any review at all; I would have my crawler visit the discussion group quite often just to get reviews. List 3.16 is one review returned by the crawler.

LIST 3.16
An RDF Document Created by Reviewer

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns="http://www.yuchen.net/photography/Camera#">
6   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         NikonD70.rdf#Nikon-D70">
7      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8      <review rdf:datatype="&xsd;string">excellent!</review>
9   </rdf:Description>
10 </rdf:RDF>
```

After I get all the necessary RDF documents, I start to aggregate them. For the Nikon D70 camera, the results are as shown in List 3.17. List 3.18 is the result for Canon.

LIST 3.17
Aggregation Result for Nikon

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
```

```
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns="http://www.yuchen.net/photography/Camera#">
6   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
        NikonD70.rdf#Nikon-D70">
7      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8      <weight>
9        <rdf:Description>
10         <rdf:value rdf:datatype="&xsd;decimal">1.4</rdf:value>
11           <uom:units rdf:resource="http://www.something.org/
                units#lbs"/>
12       </rdf:Description>
13     </weight>
14     <soldItem rdf:datatype="&xsd;integer">12</soldItem>
15     <soldItem rdf:datatype="&xsd;integer">16</soldItem>
16     <soldItem rdf:datatype="&xsd;integer">11</soldItem>
17     <review rdf:datatype="&xsd;string">excellent!</review>
18     <review rdf:datatype="&xsd;string">I like this the best
       </review>
19     <review rdf:datatype="&xsd;string">cool stuff</review>
20     <review rdf:datatype="&xsd;string">good for everything
       </review>
21   </rdf:Description>
22 </rdf:RDF>
```

LIST 3.18
Aggregation Result for Canon

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:uom="http://www.standards.org/measurements#"
5          xmlns="http://www.yuchen.net/photography/Camera#">
6   <rdf:Description rdf:about="http://www.yuchen.net/rdf/
        Canon20D.rdf#Canon-20D">
7      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
8       <weight>
9         <rdf:Description>
10           <rdf:value rdf:datatype="&xsd;decimal">1.6</rdf:value>
11           <uom:units rdf:resource="http://www.something.org/
                units#lbs"/>
12        </rdf:Description>
13     </weight>
14     <soldItem rdf:datatype="&xsd;integer">6</soldItem>
15     <soldItem rdf:datatype="&xsd;integer">9</soldItem>
16     <review rdf:datatype="&xsd;string">good enough</review>
```

```
17      <review rdf:datatype="&xsd;string">okay</review>
18   </rdf:Description>
19 </rdf:RDF>
```

Now you see my point. The two RDF documents have achieved what I wanted. My own application can now do all kinds of things, including showing the results in a nice table.

This example shows the power of RDF statements and also the power of aggregation. All the distributed information is finally collected together and used in an intelligent way. This is accomplished just by agreeing upon the URIs; different parties can still present their information as they wish. For instance, a Nikon vendor can add new properties (predicates) without breaking my application; there is also no need for other vendors to change anything. The same is true for the reviewers.

On the other hand, if we do not have anything in common, there will be a problem. It is be almost impossible to write an application to accomplish what I achieved earlier. Even if it were possible, it would be extremely expensive to maintain the code; a single change by any of the vendors would completely break my application.

In Chapter 4, we will start to look at the RDF schema. After we have enough knowledge of RDF schema, we will be able to see more exciting examples on the Web.

3.7 MORE ABOUT RDF

Before we get into RDF schema, let us discuss two more issues about RDF.

3.7.1 The Relationship between DC and RDF

We have discussed several important issues about creating RDF documents in the previous hypothetical example. One of the key points to remember is to make our own RDF documents have something in common with others that already exist. For instance, it is perfectly legal to make up our own URIs to represent the subjects and objects if no one ever created them yet. However, if we are indeed describing something that is already depicted by someone else and if we are sure that these subjects and objects do share the same semantics, the best thing to do is to reuse the URIs that are already being used to represent these resources.

In fact, this rule is applicable not only to subjects and objects, but also to predicates. One benefit of reusing existing predicates is that related applications can use the information in your RDF documents without requiring the developers to modify them to recognize your predicates.

Dublin Core (discussed in Chapter 1) is widely used as a vocabulary to describe documents (Web pages, for instance). Therefore, if we are using RDF to describe a document, or maybe part of our RDF document is to describe a document, we should use the DC predicates as much as we can: the `Title` predicate, `Creator` predicate, to name just a few.

By now, the relationship between Dublin Core and RDF should be clear: Dublin Core is a group of standard URIs that RDF documents should make use of whenever it is appropriate to do so.

Now, let us recall the last hypothetical example where we had used some aggregation tools to get the sales information about Nikon and Canon cameras. We are very happy with the result, and we want to give ourselves some credit by adding the creator and date information into the final RDF file. To do this, we can use Dublin Core vocabulary without inventing our own, as shown in List 3.19.

LIST 3.19
Aggregation Result with the `creator` and `date` Information

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
   XMLSchema#">]>
3 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4          xmlns:dc="http://www.purl.org/metadata/dublin-core#"
5          xmlns:uom="http://www.standards.org/measurements#"
6          xmlns="http://www.yuchen.net/photography/Camera#">
7    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         NikonD70.rdf#Nikon-D70">
8      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
9        <weight>
10         <rdf:Description>
11           <rdf:value rdf:datatype="&xsd;decimal">1.4</rdf:value>
11           <uom:units rdf:resource="http://www.something.org/
                 units#lbs"/>
13         </rdf:Description>
14      </weight>
15      <soldItem rdf:datatype="&xsd;integer">12</soldItem>
16      <soldItem rdf:datatype="&xsd;integer">16</soldItem>
17      <soldItem rdf:datatype="&xsd;integer">11</soldItem>
18      <review rdf:datatype="&xsd;string">excellent!</review>
19      <review rdf:datatype="&xsd;string">I like this the best
        </review>
20     <review rdf:datatype="&xsd;string">cool stuff</review>
21     <review rdf:datatype="&xsd;string">good for everything
       </review>
22     <dc:creator>Liyang Yu</dc:creator>
23     <dc:date>2005-06-21</dc:date>
24   </rdf:Description>
25 </rdf:RDF>
```

Lines 22 and 23 show the Dublin Core predicates used to describe the creator and creation date of the document, and we can certainly add more information if we want. But you see how easy it is to use it, as you just need to specify the Dublin Core namespace and use it anywhere you want in your document.

3.7.2 The Relationship between XML and RDF

The relationship between XML and RDF can be described very simply: RDF is a more restricted form of XML. RDF uses the XML syntax and its namespace concept; it is another XML vocabulary, designed to give meaning to information so that the information distributed over the Internet becomes machine-processable.

Given this relationship between XML and RDF, it is natural to ask why XML cannot accomplish what RDF has accomplished.

There are several reasons for this. First of all, XML provides very limited semantics, and even for this limited semantics, it is quite ambiguous. This is nicely summarized as follows:

> XML is only the first step to ensuring that computers can communicate freely. XML is an alphabet for computers and as everyone traveling in Europe knows, knowing the alphabet doesn't mean you can speak Italian or French.
>
> **— *Business Week*, March 18, 2002 [32]**

The key point here is, XML is by far the best format to share data on the Web and exchange information between different platforms and applications; however, it does not have enough restrictions to express semantics.

Here is an example. How do we use XML to express the fact that the author of *Introduction to the Semantic Web* is Liyang Yu? Using XML, you have several ways to do this (see List 3.20).

LIST 3.20
Ambiguity of XML Documents

```
<!-- form 1 -->
<author>
  <firstName>Liyang</firstName>
  <lastName>Yu</lastName>
  <book>
    <title>Introduction to the Semantic Web</title>
  </book>
</author>

<!-- form 2 -->
<author>
  <name>Liyang Yu</name>
  <book>
    <title>Introduction to the Semantic Web</title>
  </book>
</author>
```

```
<!-- form 3 -->
<author>
  <name>Liyang Yu</name>
  <book>Introduction to the Semantic Web</book>
</author>
```

You can tell there is no agreement on the structure you can use. An automatic agent that works on a large scale is virtually impossible, if not prohibitively expensive to build and maintain. On the other hand, using RDF to express the same idea is very straightforward and leaves no room for any ambiguity, as shown in List 3.21. The only part you can change in List 3.21 is the URIs of the resources (you have to name them if they do not already exist). Any tool or software agent can easily characterize this structure and understand which part of the structure is the subject, the property, and the value of that property.

LIST 3.21
Using an RDF Document to Express the Fact Described in List 3.20

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://www.purl.org/metadata/dublin-core#">

  <rdf:Description rdf:about="http://yuchen.net/book#semanticweb">
    <dc:title>Introduction to the Semantic Web</dc:title>
    <dc:creator>Liyang Yu</dc:creator>
  </rdf:description>

</rdf:RDF>
```

Second, parsing XML statements heavily depends on the tree structure, which is not quite scalable on a global basis. To be more specific, you can easily make up an XML document so that its representation in computer memory depends on the data structures, such as tree and character strings. In general, these data structures can be quite hard to handle, especially when large.

An RDF statement presents a very simple data structure: a directly labeled graph that has long been a well-understood data structure and is also quite scalable for a large data set. The nodes of the graph are the resources or literals, the edges are the properties, and the labels are URIs of nodes and edges. You can certainly change the graph into a collection of triples (subject-predicate-object) that fit into the framework of relational database very well. All these are extremely attractive, compared to XML documents.

The third reason, which is even more important, is that using the RDF format promotes the development and usage of standardized vocabularies (or ontologies, as you will see in subsequent chapters). The more you understand about the Semantic Web, the more you will appreciate the importance of these vocabularies. The following are some of the benefits of using standard vocabularies:

- Without a shared vocabulary, the same words can always mean different concepts, and different words can possibly have the same meaning.
- Without a shared vocabulary, there will be no way to semantically markup a Web page.
- Without a shared vocabulary, distributed information will likely remain "distributed"; an automatic agent that is capable of processing this distributed information on a global scale is just too hard to build.

By now, the reason why the RDF format needed to be invented should be clear to you. XML is unequalled as an information-exchange format over the Internet, but by itself, it simply does not provide what we need for the construction of the Semantic Web.

If you are still not convinced, try this small experiment. Take the hypothetical example we discussed earlier (three Nikon vendors and two Canon vendors), replace all the RDF documents by XML documents, and see how many more constraints you need to artificially impose to make it work and how much more case-specific code you need to write. The benefit of the RDF format will become apparent.

3.8 RDF TOOLS

In this last section, we take a look at the tools we can use for RDF documents. In particular, we are going to discuss one such tool: the RDF validator provided by `www.w3.org` at `www.w3.org/RDF/validator/`. We will look at other RDF tools in subsequent chapters.

I recommend the use of this tool, especially when you are learning the RDF language; it at least indicates whether the submitted RDF document is in proper RDF format or not. Let us take a look at some examples.

First, open the Web site of the validator; this tool will be displayed as shown in Figure 3.7.

Paste the RDF document shown in List 3.22 into the document window, and ask for the triples only (if you ask for the graph, it might take a long time).

LIST 3.22
A Simple RDF Document for the Validation Example

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3          xmlns:uom="http://www.standards.org/measurements#"
4          xmlns="http://www.yuchen.net/photography/Camera#">
5    <rdf:Description rdf:about="http://www.yuchen.net/rdf/
         NikonD70.rdf#Nikon-D70">
6      <rdf:type rdf:resource="http://www.yuchen.net/photography/
           Camera#SLR"/>
7       <weight>
8         <rdf:Description>
```

```
9           <rdf:value rdf:datatype="http://www.w3.org/2001/
                XMLSchema#decimal">1.4</rdf:value>
10            <uom:units rdf:resource="http://www.something.org/
                  units#lbs"/>
11        </rdf:Description>
12      </weight>
13    </rdf:Description>
14 </rdf:RDF>
```

After clicking the "parse RDF" key, we will get the result shown in Figure 3.8; so we know that this RDF document is in proper format. As we can see, the RDF statements are shown in the form of triples.

Up to this point, we have gained a solid understanding of RDF. To make the picture more complete, we need to learn the other half of RDF: the RDF schema. This will be the main topic of Chapter 4.

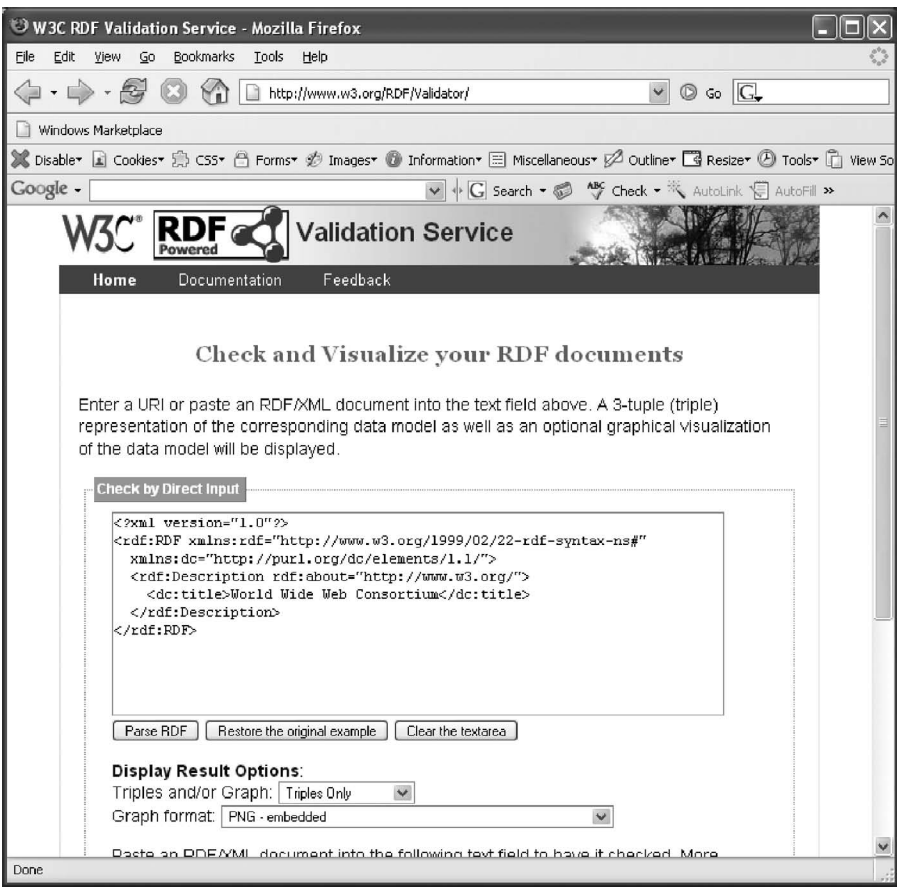


FIGURE 3.7 RDF document validator provided by `www.w3.org`.

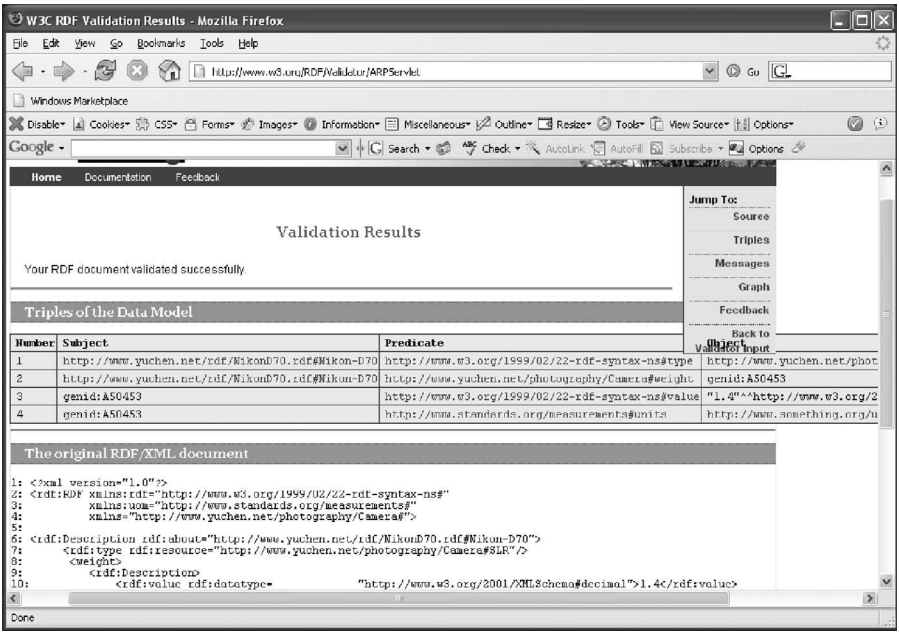


FIGURE 3.8 Validation result.