9 Mark Up Your Web Document, Please!

9.1 SEMANTIC MARKUP: A CONNECTION BETWEEN TWO WORLDS

9.1.1 What is Semantic Markup?

Before talking about semantic markup, let us review what we have learned so far. First, after all these chapters, we have gained a much better understanding of the Semantic Web. We can use one simple sentence to describe the Semantic Web: it is all about extending the current Web to make it more machine-understandable.

To accomplish this goal, we first need some languages to express semantics. This is the area we have learned the most about; we have covered RDF, RDF schema, and Web Ontology Language (OWL). These languages can be used to develop a formal ontology: a Semantic Web vocabulary expressing the semantics agreed on by different users in a specific domain; more importantly, this ontology can also be understood by machines.

In fact, we have learned quite a lot about ontology and its development. Swoogle is a search engine to help us to search and understand ontologies. Friend of a Friend (FOAF) is a great ontology example in the domain of social networking. It has also shown us the power of aggregating distributed information from the Web.

However, when we look at our goal and what we have accomplished so far, we realize that something is missing: the current Web is one world, and the machine-readable semantics expressed by ontologies is another world. Where is the connection between these two worlds? If these worlds stay separate, there will be no way to extend the current Web to make it more machine-readable.

Therefore, to accomplish our goal, we have to build a connection between the current Web and the semantic world. To make the connection even more explicit, we have to add semantics to the current Web. After this connection is created, the current Web will have machine-understandable content, and a smart agent can start to process the Web in a much more automatic fashion and also on a much larger scale.

As you might have guessed, adding semantics to the current Web is called *semantic mark up*; sometimes, it is also called *annotation*.

If you think about this, you will see that marking up a document is a natural follow-up step. Ontologies are very important for the Semantic Web vision, but they just provide some common definitions allowing a shared understanding of the context for some Web pages. However, ontologies are independent of the Web pages; a given ontology does not provide a link to any specific Web page and is not linked by any Web page either. To make Web pages machine-readable, they must be connected to the appropriate ontologies.

Now that you understand the need to markup a Web page using ontologies, you might wonder what a mark up looks like. This is the topic of this chapter, but here is a sneak preview: a markup file is normally a Resource Description Framework (RDF) document containing RDF statements that describe the content of a Web page by using the terms defined in one or several ontologies. For instance, suppose a Web page describes some entities in the real world; the markup document of this page may specify that these entities are instances of some classes defined in some ontology, and these instances have properties and share relationships among themselves.

It is important to realize that a markup document is mainly for the agents to read; it is not for human eyes. When an agent reaches a Web page and finds that the page has a markup document (more details on this later), it reads it and also loads the related ontologies into its memory. At this point, the agent turns itself into a smart agent; by reading the statements in the markup document and parsing the ontologies, it can "understand" the content of the current Web page and "discover" the implied facts about this page. The final result is that the same Web page not only continues to look great to human eyes, but also makes perfect sense to machines. Certainly, this is what is meant when we say, "extend the current Web by adding semantics to it so as to make it machine-readable."

9.1.2 The Procedure of Semantic Markup

There are several steps you need to follow when semantically marking up a Web page:

Step 1: Decide which ontology or ontologies to use for semantic markup. The first step is to decide which ontology to use. Sometimes, you may need more than one ontology. This involves reading and understanding the ontology to decide whether it fits your need and whether you agree with the semantics expressed by it. It is possible that you may have to come up with your own ontology, in which case, you need to remember the rule of always trying to reuse existing ontologies, or simply constructing your new ontology by extending an existing ontology.

Step 2. Markup the Web page. Once you have decided the ontology you are going to use, you can start to markup the page. At this point, you need to decide what content on your page you want to markup. Clearly, it is neither possible nor necessary to markup everything on your page. Having some kind of agent in mind would help you make the decision. The question you want to ask yourself is, for instance, if there were an agent visiting this page, what information on this page would the agent want to understand? Remember, your decision is also constrained by the ontology you have selected: the markup statements have to be constructed based on the ontology; therefore, you can only markup the contents that are supported by the selected ontology.

You can elect to write your markup document by using a simple editor or by using some tools. Currently, there are tools available to help you to markup your pages, as we will see in the markup examples later in this chapter. If you decide to use a simple editor to manually markup a Web

page, remember to use a validator to ensure your markup document does not contain any syntax errors. The reason is simple: the agent that reads this markup document may not be as forgiving as you hope: if you make some syntax errors, many of your markup statements can be totally skipped and ignored.

After you have finished creating the markup document, you need to put it somewhere on your Web server; it is vital that you grant enough rights to it so that the outside world can access it. This is also related to the last step, which follows.

Step 3: Let the world know your page has a markup document. The last step is to inform the world that your page has a markup document. For instance, your page could be visited by a soft agent that has no idea whether your page has a markup document or not. Therefore, you need to explicitly indicate on your Web page that it has a markup file.

At the time of this writing, there is no standard way of accomplishing this. A popular method is to add a link in the HTML header of the Web page. We will use this method in the examples presented in the next section of this chapter.

Let us now move on to the next section, where we will show you a real-world example of page markup. We will accomplish this by using both an editor and an available markup tool.

9.2 MARKING UP YOUR DOCUMENT MANUALLY

Let us use a small example to show the process of page markup. For this purpose I have created a HTML page, which is a simple review of Nikon's D70 digital camera. Figure 9.1 shows this page when you use a browser to open it. Our goal is to provide a markup document for this page. Let us follow the aforementioned steps discussed above.

The first step is to choose an ontology for markup. Clearly, our Web page is a simple review of D70; it is not about selling Nikon D70; i.e., it does not have any pricing information and does not tell you where you can get the best price either. It is all about the D70's performance and features from a photographer's point of view.

Given these considerations, our camera ontology seems to be the best choice. This ontology contains the basic terms such as `Digital`, `SLR`, `Specifications`, `Photographer`, etc.; it does not have anything to do with concepts such as price, vendors, stores, etc. There is also another reason for this choice: we had to learn a lot just to create this camera ontology, so we should certainly get some use out of it!

Now that we have selected our ontology, let us move on to the next step: creating the markup document. Before doing this, let us take a look at the HTML code that generates the page in Figure 9.1. The code is shown in List 9.1.

This is a typical page on the current Web. It provides enough information to render itself, but does not provide any information to make it understandable to a machine. Now, let us create a markup document for it.

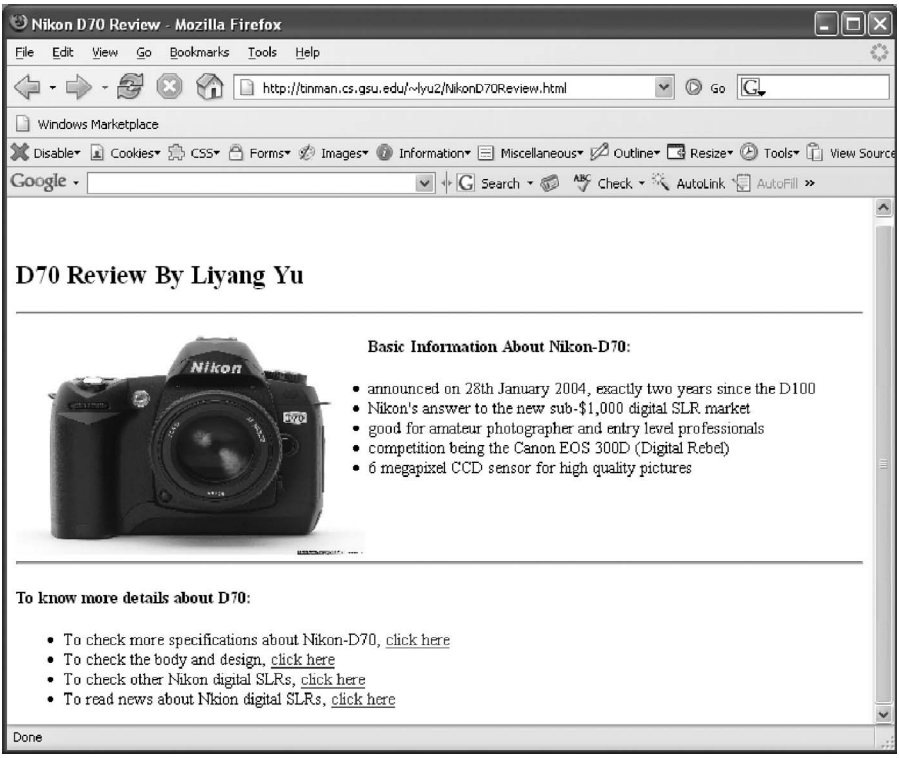

FIGURE 9.1 A Web page presenting a simple review of Nikon D70.

LIST 9.1
HTML Code for the Page Shown in Figure 9.1

```
<html>
<head>
   <title> Nikon D70 Review </title>
</head>

<body BGCOLOR=white TEXT=black LINK=blue VLINK=green>

<br><br>
<h2>D70 Review By Liyang Yu</h2>
<hr>

<img src="D70.jpg" align="left" width=300 height=200>
<h4>Basic Information About Nikon-D70:</h4>
<p>
<µl>
<li>announced on 28th January 2004,exactly two years since the D100
<li>
```

```
<li>Nikon's answer to the new sub-$1,000 digital SLR market
<li>good for amateur photographer and entry level professionals
<li>competition being the Canon EOS 300D (Digital Rebel)
<li>6 megapixel CCD sensor for high quality pictures
</ul>
</p><br><br><br>
<hr>

<h4>To know more details about D70:</h4>
<ul>
<li>To check more specifications about Nikon-D70, <a href="/D70Spec
    .html">click here</a>
<li>To check the body and design, <a href="/D70Design.html">
    click here</a>
<li>To check other Nikon digital SLRs, <a href="/Nikon.html">
    click here</a>
<li>To read news about Nkion digital SLRs, <a href="/NikonNews.html">
    click here</a>
</ul>

</body>

</html>
```

Again, it is up to you to decide what content in the page should be semantically marked up. Let us say we want to tell the future agent that this page is about a digital SLR, and its name is Nikon D70. Therefore, we start with the markup file shown in List 9.2.

LIST 9.2
The Initial Markup Document for Our Review Page

```
1:  <?xml version="1.0"?>

2:  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4:           xmlns:c="http://www.yuchen.net/photography/Camera.owl#">

5:  <rdf:Description
        rdf:about="http://www.yuchen.net/pageMarkup.rdf#Nikon-D70">
6:    <rdf:type
        rdf:resource="http://www.yuchen.net/photography/Camera
            .owl#SLR"/>
7:    <rdfs:label>Nikon-D70</rdfs:label>
8:    <rdfs:label>D-70</rdfs:label>
9:  </rdf:Description>

10: </rdf:RDF>
```

This is our initial markup document about our example page. Line 1 indicates this is an Extensible Markup Language (XML) document. Lines 2 to 4 give the namespaces used in the document. Note that line 4 provides an important clue to the agent; the agent knows the statements in the documents use the camera ontology on account of this line. The agent will load our camera ontology into memory for later use at this time.

Lines 6 to 10 define an instance of class SLR. This instance has the following URI:

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

Also, two `rdfs:labels` are added to the instance just to indicate that this instance can use names such as Nikon-D70 or simply D-70.

At this point, this markup document indicates that the Web page being annotated is a page that talks about a member of this class:

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

This is clearly not enough. For one thing, this page is written by someone named *Liyang Yu*, and (let us just assume) he is a photographer. So let us continue to add this information to our markup document. The current version is shown in List 9.3.

LIST 9.3
Improved Markup Document for the Review Page (1)

```
1:  <?xml version="1.0"?>

2:  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4:           xmlns:camera="http://www.yuchen.net/photography/Camera
                   .owl#">

5:  <rdf:Description
         rdf:about="http://www.yuchen.net/pageMarkup.rdf#Nikon-D70">
6:    <rdf:type
         rdf:resource="http://www.yuchen.net/photography/Camera
             .owl#SLR"/>
7:    <rdfs:label>Nikon-D70</rdfs:label>
8:    <rdfs:label>D-70</rdfs:label>
9:  </rdf:Description>

10: <camera:Photographer
            rdf:about="http://www.yuchen.net/people#LiyangYu">
11:   <rdfs:label>Liyang Yu</rdfs:label>
12: </camera:Photographer>

13: </rdf:RDF>
```

Lines 10 to 12 define an instance of the class `Photographer`; this instance has the following URI:

```
http://www.yuchen.net/people#LiyangYu
```

Note that I used a simpler version to define the `Photographer` instance, just to remind you there are several ways to define a member, as discussed in Chapter 3.

Now imagine what a soft agent sees when it reaches our current markup document. The Web page itself (Figure 9.1) does not make any sense; however, just by reading this markup document, the agent realizes that this page discusses some members of these two classes:

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

Unfortunately, this is all the agent can understand at this point. To let it know more about our page, we need to add more information into the markup document. For example, we can add the following facts: the underlying Web page has a model name for the camera; it also mentions some specifications such as 6-megapixel picture quality, and so on. After adding all this information to the markup document, we get a more detailed version, as shown in List 9.4.

LIST 9.4
Improved Markup Document for the Review Page (2)

```
1:  <?xml version="1.0"?>

2:  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3:           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4:           xmlns:dc="http://www.purl.org/metadata/dublin-core#"
5:           xmlns:camera="http://www.yuchen.net/photography/Camera
                   .owl#">

6:  <rdf:Description
        rdf:about="http://www.yuchen.net/Photography/NikonD70Review
            .html">
7:     <dc:title>D70 Review By Liyang Yu</dc:title>
8:     <dc:creator>Liyang Yu</dc:creator>
9:  </rdf:Description>

10: <rdf:Description
         rdf:about="http://www.yuchen.net/pageMarkup.rdf#Nikon-D70">
11:    <rdf:type
        rdf:resource="http://www.yuchen.net/photography/Camera
            .owl#SLR"/>
12:    <rdfs:label>Nikon-D70</rdfs:label>
13:    <rdfs:label>D-70</rdfs:label>
```

```
14:     <camera:pixel rdf:datatype="http://www.someStandard.org
         #MegaPixel">
         6
15:     </camera:pixel>
16:     <camera:has_spec>
17:        <rdf:Description>
18:           <camera:model
               rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
               Nikon-D70
19:           </camera:model>
20:        </rdf:Description>
21:     </camera:has_spec>
22: </rdf:Description>

23: <camera:Photographer
            rdf:about="http://www.yuchen.net/people#LiyangYu">
24:    <rdfs:label>Liyang Yu</rdfs:label>
25: </camera:Photographer>

26: </rdf:RDF>
```

Now this seems to be a fairly good markup document; it tells the following about the underlying Web page:

1. `http://www.yuchen.net/Photography/NikonD70Review.html` is the URL of this page, and this page is titled *D70 Review by Liyang Yu.* Its creator is Liyang Yu.
2. This page is about a digital camera. More specifically, this camera is an `SLR` instance whose URI is given by `http://www.yuchen.net/page-Markup.rdf#Nikon-D70`. The camera's `pixel` property has a value of `6`, and its `has_spec` property has a `model` property valued `Nikon-D70`.
3. This page also involves a Photographer whose URI is given by `http://www.yuchen.net/people#LiyangYu`.

This markup document seems to have captured the main content of the Web page, at least based on the current version of the `Camera` ontology. Therefore, at this point, we consider the markup work done.

Note that we have used Dublin Core code to identify the page title and page author (lines 6 to 9). Also, we used an anonymous class as the value of the `has_spec` property; this anonymous class has a type of `Specifications` class (lines 17 to 20) that has a `model` property. In this case, the `model` property has a value `Nikon-D70`. You have learned all this in Chapter 3, and if you do not recognize these constructs well, you need to review Chapter 3.

Now we are ready for the last step: explicitly indicate the fact that the review Web page has been marked up by an RDF file. Note that at the time of this writing, there is no standard yet on how this is done. A popular solution is to add a `link` tag in the header of the HTML document. In fact, we used the same solution earlier

to indicate that a FOAF document is associated with a homepage. This makes a lot of sense if you realize that a FOAF file can be considered a special markup to a person's homepage.

List 9.5 shows the HTML code for our original review Web page after we added the `<link>` tag in the HTML header.

LIST 9.5
HTML Code for the Page Shown in Figure 9.1 with the `<link>` Tag Added

```
<html>
<head>
  <link rel="meta" type="application/rdf+xml" href="/markup.rdf"/>
  <title> Nikon D70 Review </title>
</head>

<body BGCOLOR=white TEXT=black LINK=blue VLINK=green>

<br><br>
<h2>D70 Review By Liyang Yu</h2>
<hr>

<img src="D70.jpg" align="left" width=300 height=200>
<h4>Basic Information About Nikon-D70:</h4>
<p>
... (the rest of the page) ...
```

A reminder: if your markup document is created manually (as in this case), it is always a good idea to validate it. In Chapter 3 we discussed how to use tools to validate an RDF document; remember that a simple syntax error may cause a big chunk of your RDF document to be ignored.

Another way to markup your Web page is to use tools. Let us discuss this in the next section.

9.3 MARKING UP YOUR DOCUMENT BY USING TOOLS

There are several tools available for creating a markup file for a given Web page. We will use SMORE [38] as an example to show how this is done. SMORE is one of the projects developed by the researchers and developers in the University of Maryland at College Park, and you can take a look at their official Web page at `http://www.mindswap.org/`. SMORE allows the user to markup Web documents without requiring a deep knowledge of OWL terms and syntax. You can create different instances easily by using the provided Graphical User Interface (GUI), which is quite intuitive and straightforward. Also, SMORE lets you visualize your ontology, and therefore it can be used as an OWL ontology validator as well.

You can download SMORE from `http://www.mindswap.org/2005/SMORE/`, and after you fire it up, you will see the initial interface shown in Figure 9.2.

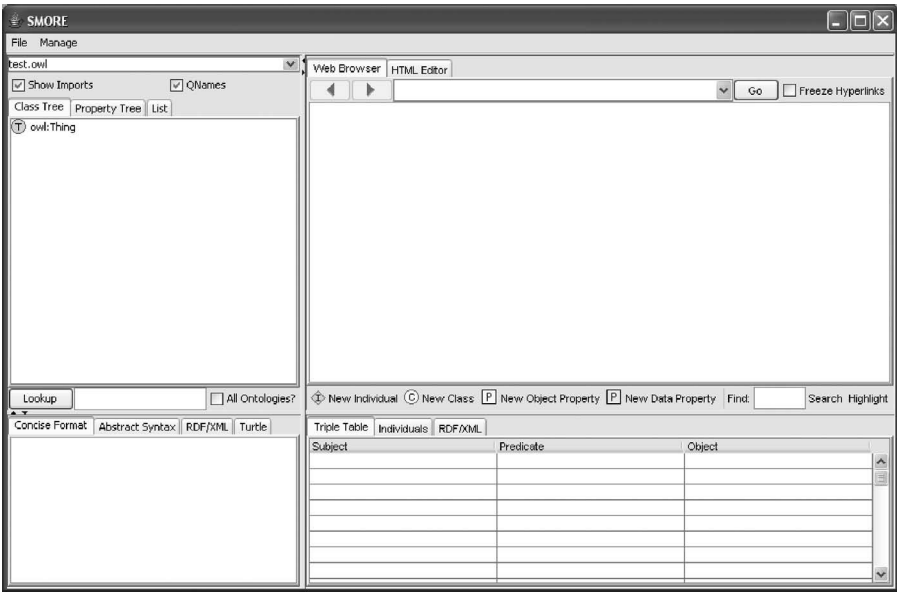

FIGURE 9.2 SMORE is a tool you can use to markup your Web page.

The first step is to load the ontology you want to use. Use the File menu to load it, either from your local machine or from the Web. There is no limit to how many ontologies you can load. Let us load our camera ontology from a local machine.

The second step is to load the Web page your want to markup by entering the URL in the upper-right textbox. This Web page can be either on your local machine or on some Web server.

Once you open both the ontology and the Web page, you will see the screen shown in Figure 9.3. As you can tell, this is quite convenient. Also, note that the upper-left window shows all the classes in the camera ontology in tree form. You can select the Property Tree tab to make it show all the properties. This can also be used to validate your ontology. In fact, if there is an error in your ontology file, SMORE will not even open it. If your ontology file is indeed a legal document (both syntactically and semantically), SMORE will open it, and you can view the tree structure to ensure that it does logically represent the knowledge you want to express.

Now you can start creating your markup file. For instance, you can click the New Individual tab, and a window with a textbox will pop up; you can enter a name for this member instance, and click and drag a class from the ontology view to indicate that this instance is a member of the class you just selected. Once you click the OK button, a member is created in the markup document and will be shown in the lower-right instance window.

The next step is to add some properties to this newly created individual. Click the Individuals tab in the member window; you can see all the available properties for this new member will show in a form, and you can simply fill out the form to indicate the proper values for these properties. This process is shown in Figure 9.4.

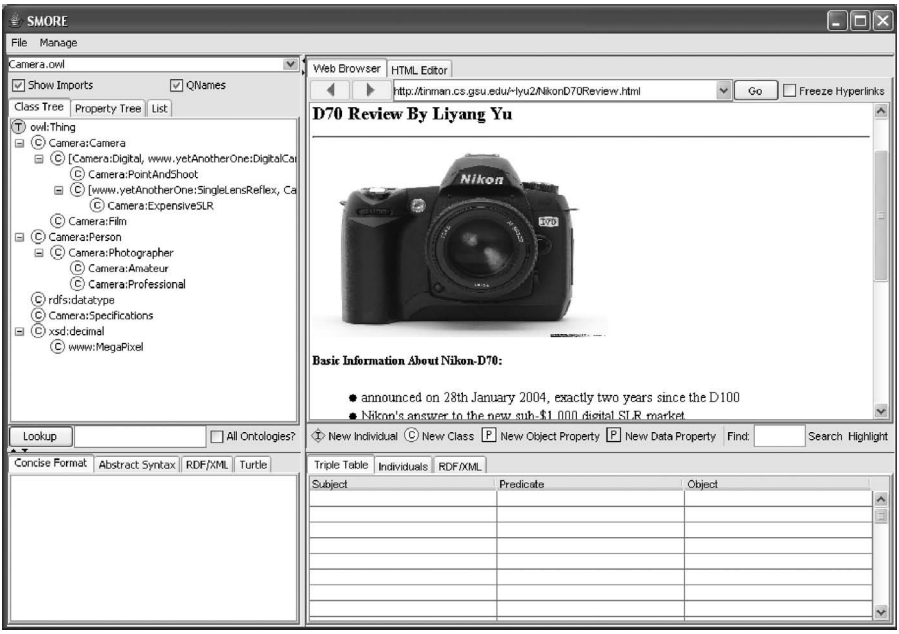

FIGURE 9.3 Using SMORE to open the ontology and Web page you are working on.

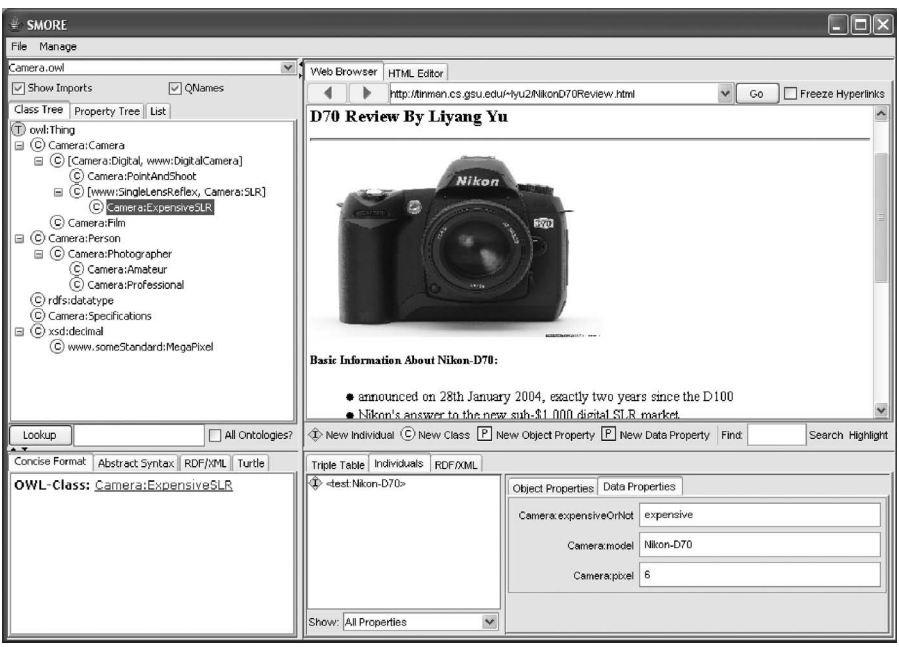

FIGURE 9.4 Using SMORE to create a markup document for the Web page.

I am not going to list all the details; you just need to continue this process until you finish creating your markup document. Once you have completed the markup process, you can click the RDF/XML tab to see the document itself. You might want to make modifications if necessary; however, you do not have to worry about the correct syntax, the necessary namespaces, etc. Generally speaking, it is always a good idea to use a tool to create your markup file.

9.4 SEMANTIC MARKUP ISSUES

As mentioned earlier, marking up a document is the process of building the critical link between the current Web and the machine-readable semantics. It is the actual implementation of adding semantics to the Web. It is so important that we have devoted a whole chapter to this topic. What is equally important is that there are many still unsolved issues associated with Web page markup.

9.4.1 Who and Why?

In this chapter, we created a simple markup document for a simple Web page. The first thing you might have noticed is that whether we decided to markup a page manually or by using some tools, a lot of work is required just to markup the simple page shown in Figure 9.1.

The question then is, how do we markup all the Web pages on the Internet? Given the huge number of pages on the Web, it is just not possible. Should we then ask the owner of each page to markup his or her page? Before we even talk about the feasibility of this solution — all these owners have to learn at least something about ontology and OWL, among other things — the first question to ask is, why should they even agree to go through the trouble of marking up their pages?

This is quite a dilemma: without a killer Semantic Web application to demonstrate some significant benefit to the world, there will be no motivation for the page owners to markup their pages; on the other hand, without the link between the current Web and the machine-readable semantics, the killer application (whatever it is) simply cannot be created.

9.4.2 Is Automatic Markup Possible?

A possible solution, given the preceding dilemma, is to somehow create the markup document automatically for each page. But is this possible?

In recent years, there has been some research in this area, and some automatic markup solutions have been proposed. However, most of these techniques are applied to technical texts. For the Internet, which contains highly heterogeneous text types that are mainly made up of natural languages, there seems to be no efficient solution yet.

9.4.3 Centralized or Decentralized?

Let us take one step back: let us not try to markup every page on the Internet by using some hypothetical magic power; instead, we can take one specific domain as an example, for instance, the domain of Bioinformatics, and (manually) markup a

majority of the pages within this domain. This solution may be feasible as long as we can solve another problem: where to save the markup document and how to let the original Web page know that it has a markup document on the Web? Remember, we are adding the markup document for the page owner, but we do not have write access to his Web server, so we cannot add a link into the header section of his or her HTML document.

As we do not have write access to his or her Web server, storing the markup document on the server is not possible. In other words, we have to consider a centralized solution: to build a server just to store all the markup documents. A carefully designed indexation system is also needed so that we can locate the markup document for a specific Web page very efficiently. By doing this, we avoid having to add any link to the header of the HTML code of each page: the agent will query the centralized markup database for each Web page it encounters to find out whether the page has a markup document or not.

This sounds like a solution. In fact, this could also be a solution to solve the dilemma we mentioned earlier: building an excellent Semantic Web application in one specific domain will show the power of the Semantic Web to the rest of the world and, hopefully, realizing the value of the Semantic Web, the Internet community will start marking up their own documents, which will make the dream of building even better Semantic Web applications possible.

This is feasible. Recall the days when only a few big companies had Web sites? Soon everyone realized that without a Web site, there could be a huge loss of potential business, and soon we had so many Web sites being built on the Internet, one could get a high-paying job by just building Web sites (not true now).

Let us hope that such a day will soon arrive for the Semantic Web.