Part 1

The World of the Semantic Web

What is the Semantic Web? It is quite impressive that at the time of my writing, if you google "what is Semantic Web" (remember to include *what is Semantic Web* in a pair of double quotes), you get just about 290 Web pages containing this phrase. However, it is equally impressive that after reading some of the "top" pages (the most relevant pages are listed at the very top in your result list), you may quickly realize that even with these well-written answers, it is still quite unclear what the Semantic Web is, why we need it, how we build it, and how to use it.

This is normal. After all, the Semantic Web is quite different in many ways from the World Wide Web that we are familiar with, including the fact that I cannot simply point you to a Web site for you to understand what it is and how it works. It is therefore not surprising that none of the aforementioned 290 pages has given you a good answer.

So, for you to understand what the Semantic Web is, I am not going to give you another equally confusing page to read. Instead, we will begin by examining how we use the World Wide Web in our daily life (work, research, etc.). We will also include a detailed description of how a search engine works in the traditional Web environment. What we will learn from these studies will enable us to understand the common difficulties we are experiencing with the Web, and more importantly, the reasons for these difficulties. At this point, we will introduce the concept of the Semantic Web and, hopefully, this concept will be less confusing to you. Furthermore, based on this basic understanding of the Semantic Web, we will "add" some semantics to the Web, and reexamine the topic of search engine: How does the added semantics change the way a search engine works, and is the result returned by the search engine improved?

Let us accomplish these goals in Part 1. Once you finish this part, you should have a solid understanding about the Semantic Web. Let the journey begin.

1 From Traditional Web to Semantic Web

1.1 WHAT IS WWW?

WWW stands for World Wide Web or, simply, the Internet. It is a magical place indeed! Anyone with a server can publish documents for the rest of the world to see, and one can hyperlink any document to any other document. Even more amazing, it does not matter (you do not even know it anyway) if the page you are browsing is being served up by someone in Beijing, China, from a Unix server or whether your Web browser is in fact running on a Macintosh machine in Atlanta, GA — if you can browse the page, you can link to it.

This exciting place has been around for nearly two decades and will continue to excite. It has become the ultimate information source. With its sheer scale and wide diversity, it presents not only intriguing challenges but also promising opportunities, from information access to knowledge discovery. Perhaps a better way to understand the Internet is to examine briefly how we use it in our daily life.

1.1.1 How Are We Using the Internet?

The answer is simple: search, integration, and Web mining are the three main uses of the Internet.

1.1.1.1 Search

This is probably the most common usage of the Internet, and most of us have at least some experience searching the Web. The goal is to locate and access information or resources on the Web. For instance, we connect to the Internet using a Web browser to find different recipes for making margaritas or to locate a local agent who might be able to help us buy a house.

Quite often though, searching on the Internet can be very frustrating. For instance, using a common search engine, let us search using the word "SOAP," which is a World Wide Web Consortium (W3C) standard for Web services. We will get about 128,000,000 listings, which is hardly helpful; there would be listings for dish detergents, soaps, and even soap operas! Only after sifting through multiple listings and reading through the linked pages will we be able to find information about the W3C's SOAP (Simple Object Access Protocol) specifications.

The reason for this situation is that search engines implement their search based on which documents contain the given keyword. As long as a given document contains the keyword, it will be included in the candidate set that is later presented to the user as the search result. It is then up to the user to read and interpret the result and extract useful information. This will become clearer in subsequent chapters; we will show you exactly how a search engine is constructed in the traditional Web environment.

1.1.1.2 Integration

Integration may sound a little academic, but in fact, you are doing it more often than you realize. It means combining and aggregating resources on the Web so that they can be collectively useful.

For instance, you decide to try some Indian food for your weekend dining out. You first search the Web to find a restaurant that specializes in Indian cuisine (good luck on that, given the fact that searching on the Internet could be hard, as we have discussed earlier), pick the restaurant, and write down the address. Next you open up a new browser and go to your favorite map utility to get the driving directions from your house to the restaurant. This is a simple integration process: you first get some information (the address of the restaurant), you use it to get more information (the directions), and these collectively help you enjoy a nice dinner out.

This is certainly a somewhat tedious process; it would be extremely nice if you could make the process easier; for instance, some automatic "agent" might be able to help you out by conducting all the searches for you.

The idea of automation here might seem to be more like a dream to you, but it could be very realistic in some other occasions. In fact, a Web service is a good example of integration, and it is more often conducted by a variety of application systems. For example, company A provides a set of Web services via its Web site, and you write Java code (or whatever language you like) to `consume` these services, so you can, say, search their product database in your application system on the fly. By providing several keywords that should appear in a book title, the service will return a list of books whose titles contain the given keywords.

This is an integration between their system and your application. It does not matter what language they use to build their Web services and what platform these services are running on, and it does not matter either which language you are using or what platform you are on — as long as you follow some standards, this integration can happen quite nicely.

Furthermore, this simple integration can lead to a set of more complex integration steps. Imagine booking an airline ticket. The first step is to write some code to consume a Web service provided by your favorite airline, to get the flight schedules that work for you. After successfully getting the schedules, the second step is to feed the selected flights to the Web service offered by your travel agent to query the price. If you are comfortable with the price, your final step is to invoke the Web service to pay for the ticket.

This integration example involves three different Web services (in fact, this is what we call *composition* of Web services), and the important fact is that this integration process proceeds just as in the case where you wanted to have dinner in an Indian restaurant; you have to manually integrate these steps together. Wouldn't it be nice if you had an automated agent that can help you find the flight schedule, query the price, and finally book the ticket? It would be quicker, cleaner and, hopefully, equally reliable.

1.1.1.3 Web Data Mining

Intuitively speaking, data mining is the nontrivial extraction of useful information from large (and normally distributed) data sets or databases. The Internet can be viewed as a huge distributed database, so Web data mining refers to the activity of getting useful information from the Internet. Web data mining might not be as interesting as searching to a casual user, but it could be very important to and even be the daily work of those who work as analysts or developers for different companies and research institutes.

One example of Web data mining is as follows: Let us say that we currently work as consultants for the air traffic control tower at Atlanta International Airport, which is reportedly the busiest airport in the nation. The people in the control tower wanted to understand how weather conditions may affect the takeoff rate on the runways (*takeoff rate* is defined as the number of aircraft that have taken off in a given hour). Obviously, dramatically unfavorable weather conditions will force the control tower to shut down the airport so that the takeoff rate will go down to zero, and normally bad weather will just reduce the takeoff rate.

For a task such as this, we suggest that as much historical data as possible be gathered and analyzed to find the pattern of the weather effects. We are told that historical data (the takeoff rates at different major airports for the past, say, 5 years) do exist, but are published in different Web sites, and the data we need are normally mingled with other data that we do not need.

To handle this situation, we will develop an agent that acts like a crawler: it will visit these Web sites one by one, and once it reaches a Web site, it will identify the data we need and collect only the needed information (historical takeoff rates) for us. After it collects the data, it will store them into the data format we want. Once it finishes with a Web site, it will move on to the next until it has visited all the Web sites that we are interested in.

This agent is doing Web data mining. It is a highly specialized piece of software that is normally developed on a case-by-case basis. Inspired by this example, you might want to code your own agent that will visit all the related Web sites to collect some specific stock information for you and report these stock prices back to you, say, every 10 minutes. By doing so, you do not have to open up a browser every 10 minutes to check the prices, risking the possibility that your boss will catch you visiting these Web sites; yet, you can still follow the latest happenings in the stock market.

This agent you have developed is yet another example of Web data mining. It is a very specialized piece of software and you might have to recode it if something important has changed on the Web sites that this agent routinely visits. But it would be much nicer if the agent could "understand" the meaning of the Web pages on the fly so you do not have to change your code so often.

We have discussed the three major activities that you normally do with the Internet. You might be a casual visitor to the Internet or a highly trained professional developer, but whatever you do with the Internet will fall into one of these three categories (let us not worry about creating new Web sites and adding them to the Internet; it is a different use of the Internet from the ones we discuss throughout this book). The next questions, then, are as follows: What are the common difficulties that you have experienced in these activities? Does any solution exist to these difficulties at all? To make it easier, what would you do if you had the magic power to change the way the Internet is constructed so that we did not have to experience these difficulties at all?

Let us discuss this in the next section.

1.1.2 What Stops Us from Doing More?

Let us go back to the first main activity, search. Of the three major activities, this is conducted by literally every user, irrespective of his or her level in computer science training. It is interesting that this activity in fact shows the difficulty of the current Internet in a most obvious way: whenever we do a search, we want to get only relevant results; we want to minimize human intervention in finding the appropriate documents.

However, the conflict also starts here: The Internet is entirely aimed for reading and is purely display oriented. In other words, it has been constructed in such a way that it is oblivious to the actual information content; Web browsers, Web servers, and even search engines do not actually distinguish weather forecasts from scientific papers, and cannot even tell a personal homepage from a major corporate Web site. The search engines, for example, are therefore forced to do keyword matching only; as long as a given document contains the keyword, it will be included in the candidate set that is later presented to the user as the search result.

The real reason for our difficulty, therefore, is that the current Internet is not constructed well; computers can only present users with information, but they cannot "understand" the information well enough to display the data that is most relevant in a given circumstance.

If we only had the magic power, we would reconstruct the Internet so that computers could not only present the information contained in the Internet but also understand the very information they are presenting and make intelligent decisions on our behalf. If we could do this, we would not have to worry about irrelevant search results; the Internet would be very well constructed and computers would understand the meaning of the information stored in the Internet and filter the pages for us before they present them to us.

As for the second activity, integration, we experience another difficulty: there is too much manual work involved and we need more automation. At first glance, this difficulty seems to be quite different from the one we experienced with searching. For instance, let us reconsider the case in which we needed to book an airline ticket. We want to have an automated agent that can help us to find the flight, query the price, and finally book the ticket. However, to automatically composite and invoke these applications (Web services), the first step is to discover them. If you think about this process, you will soon realize that almost all your manual work is spent on the discovery of these services. Therefore, the first step of integration is to find the components that need to be integrated in a more efficient and automated manner.

Now, back to the previous question: when we conduct integration, how can we discover (or search, if you will) the desired components (for example, Web services) on the Internet more efficiently and with less or no human intervention? As far as Web services are concerned, this goes back to the topic of automated service discovery. Currently, this integration is hard to implement mainly because the discovery process of its components is far from efficient.

The reason, again, as you can guess, is that although all the components needed to be integrated do exist on the Internet, the Internet is not programmed to remember the meaning of any of these components. In other words, for the Internet all these components are created equal. As the Internet does not know the meaning of each component, there is no way for us to teach our computers to understand the meaning of each component. The final result is that the agent we use to search for a particular component can only do its work by simply matching keywords.

Now, about the last activity, namely, Web data mining. The difficulty here is that it could be very expensive. Again, this difficulty seems to be quite different from the previous two, but soon you will see that the underlying reason for this difficulty is precisely the same.

The reason why Web data mining is very costly is that each Web data mining application is highly specialized and has to be specially developed for a particular application context. To understand this, let us consider a given Web data mining task. Obviously, only the developer knows the meaning of each data element in the data source and how these data elements should interact to present some useful information. The developer has to program these meanings into the mining software before setting it to work; there is no way to let the mining agent learn and understand these meanings "on the fly." By the same token, the underlying decision tree has to be preprogrammed into the agent too. Again, the reason is that the agent simply cannot learn on the spot, so it cannot make intelligent selections other than the ones it is programmed to do.

Now the problem should become obvious: every Web data mining task is different, and we have to program each one from scratch; it is very hard to reuse anything. Also, even for a given task, if the meaning of the data element changes (this can easily happen, given the dynamic feature of Web documents), the mining agent has to be changed accordingly because it cannot learn the meaning of the data element dynamically. All these practical concerns have made Web data mining a very expensive task.

The real reason is that the Internet only stores the presentation of each data element; it does not record its meaning in any form. The meaning is only understood by human developers, so they have to teach the mining agent by programming the knowledge into the code. If the Internet were built to remember all the meanings of data elements, and if all these meanings could be understood by a computer, we could then simply program the agent in such a way that it would be capable of understanding the meaning of each data element and making intelligent decisions "on the fly"; we could even build a generic agent for some specific domain so that once we have a mining task in that domain, we would reuse it all the time — Web data mining would then not be as expensive as it is today.

Now, we have finally reached an interesting point. We have studied the three main uses of the Internet. For each one of these activities, there is something that needs to be improved: for searching activity, we want the results to be more relevant;

for integration, we want it to be more automated; and for Web mining, we want it to be less expensive. And it is surprising to see that the underlying reason for all of these seemingly different troubles is identical:

> The Internet is constructed in such a way that its documents only contain enough information for the computers to present them, not to understand them.

If the documents on the Web also contained information that could be used to guide the computers to understand them, all three main activities could be conducted in a much more elegant and efficient way.

The question now is whether it is still possible to reconstruct the Web by adding some information into the documents stored on the Internet so that the computers can use this extra information to understand what a given document is really about.

The answer is yes; and by doing so, we change the current (traditional) Web into something we call the *Semantic Web*, the main topic of this chapter.

1.2 A FIRST LOOK AT THE SEMANTIC WEB

There are many different ideas about what the Semantic Web is. It might be a good idea to first take a look at how its inventor, Tim Berners-Lee, describes it:

> The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.
>
> . . . a web of data that can be processed directly and indirectly by machines.
>
> **— Tim Berners-Lee, James Hendler, Ora Lassila [1]**

As the inventor of the World Wide Web, Berners-Lee hopes that eventually computers will be able to use the information on the Web, not just present the information. "Machines become capable of analyzing all the data on the Web — the content, links, and transactions between people and computers" [1]. Based on his idea, the Semantic Web is a vision and is considered to be the next step in Web evolution. It is about having data as well as documents on the Web so that machines can process, transform, assemble, and even act on the data in useful ways.

There is a dedicated team of people at World Wide Web Consortium (W3C) working to improve, extend, and standardize the system. What is the Semantic Web according to this group of people?

> . . . the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration, and reuse of data across various applications.
>
> **— W3C Semantic Web Activity [12]**

I could not agree more with this idea from W3C. In fact, in the previous discussions, I have shown you why automation, integration, and reuse (for Web data mining purposes) on the current Web are so difficult. With the realization of the

Semantic Web, performing these three major activities on the Web will become much easier. Another way to understand the idea from W3C is to see it as building a machine-readable Web. Using this machine readability, all kinds of smart tools (or agents) can be invented and can be shown to easily add great value to our daily life.

This book discusses and describes the Semantic Web in the light of this machine-readable view. I will present concrete examples to show how this view can help us realize the vision of the Semantic Web proposed by Berners-Lee. Because this concept is so important, let us again summarize what the Semantic Web is:

- The current Web is made up of many Web documents (pages).
- Any given Web document, in its current form (HTML tags and natural text), only gives the machine instructions about how to present information in a browser for human eyes.
- Therefore, machines have no idea about the meaning of the document they are presenting; in fact, every single document on the Web looks exactly the same to machines.
- Machines have no way to understand the documents and cannot make any intelligent decisions about these documents.
- Developers cannot process the documents on a global scale (and search engines will never deliver satisfactory performance).
- One possible solution is to modify the Web documents, and one such modification is to add some extra data to these documents; the purpose of this extra information is to enable the computers to understand the meaning of these documents.
- Assuming that this modification is feasible, we can then construct tools and agents running on this new Web to process the document on a global scale; and this new Web is now called the Semantic Web.

This long description should give us some basic understanding about the Semantic Web and what it is and why we need it. Later on in this book we will have a discussion on how we should actually build it. We should also remember that this is just a first look at defining the Semantic Web. Later, much of our current understanding will have to be enhanced or even modified as we proceed with the book.

For example, in the definition it was mentioned that one possible solution was to "add some extra data to these documents...." In later chapters of this book, you will see that this extra information can indeed be added directly into the document and can even be created spontaneously by some parser. In fact, in some cases it might be easier to generate this extra data by parsing the document on the fly. In other words, the extra data need not necessarily be added at the time of creation of the document.

If you go one step further, you will see new problems right away; if the extra data is indeed generated spontaneously, where are we going to store them? We certainly do not have the access to modify an extant document on the Web as we are not its authors. If we save this extra information on another page, how can we link the current document to this page so later on some intelligent agent will be able to follow this link to find the data when visiting this document? Or, can we store the extra information on another dedicated server?

You can see that there are many issues that need to be understood. Let us work together so that we can build a better understanding of this exciting vision. For now, let us ensure you understand the points in the foregoing long definition.

After establishing the initial concept of the Semantic Web, most books and articles immediately move on to the presentation of the different technical components that underlie the Semantic Web. For a mind that is new to the concept, however, getting into these nuts and bolts without first seeing how these components fit together to make the vision a reality may not be the best learning approach.

Therefore, before delving into the technical details, a deeper understanding of the Semantic Web would be beneficial. To accomplish this, in Chapter 2 we will use "search" as an example — because it is the most common activity conducted on the Web — and study in detail how a search engine works under the traditional Web, and how it might work under the Semantic Web. This comparison will clearly show the precise benefit of the Semantic Web, and understanding this benefit will provide us with a much better and deeper understanding of the Semantic Web. Furthermore, by studying the search engine under both traditional and Semantic Web environments, we will be able to identify the necessary components that will make the Semantic Web possible. When we start examining the nitty-gritty of these components in Part 2, you will not be confused and, in fact, you will be motivated.

However, there is one key (technical) idea we must know before we proceed: metadata. You will see this word throughout the book, and it is one of the key concepts in the area of the Semantic Web. Let us solve this problem once and for all and move on to the last section of this chapter.

1.3 AN INTRODUCTION TO METADATA

Before we go into the details of metadata, let us see the single most important reason why we need it (this will facilitate your understanding of metadata): metadata is structured data that machines can read and understand.

1.3.1 The Basic Concept of Metadata

In general, metadata is defined as "data about data;" it is data that describes information resources. More specifically, metadata is a systematic method for describing resources and thereby improving their access. It is important to note the word *systematic*. In the Web world, *systematic* means *structured* and, furthermore, structured data implies machine readability and understandability, a key idea in the vision of the Semantic Web.

Let us examine some examples of metadata from the Web world. Clearly, the Web is made up of many Web documents. Based on its definition, the metadata of a given Web document is the data used to describe the document. It may include the title of the document, the author of the document, and the date this document was created. Other metadata elements can also be added to describe a given document. Also, different authors may come up with different data elements to describe a Web document. The final result is that the metadata of each Web document has its own unique structure, and it is simply not possible for an automated agent to process these metadata in a uniform and global way, defeating the very reason for wanting metadata to start with.

Therefore, to ensure metadata can be automatically processed by machines, some metadata standard is needed. Such a standard is a set of agreed-on criteria for describing data. For instance, a standard may specify that each metadata record should consist of a number of predefined elements representing some specific attributes of a resource (in this case, the Web document), and each element can have one or more values. This kind of standard is called a *metadata schema*.

Dublin Core (DC) is one such standard. It was developed in the March 1995 Metadata Workshop sponsored by the Online Computer Library Center (OCLC) and the National Center for Supercomputing Applications (NCSA). It has 13 elements (subsequently increased to 15), which are called Dublin Core Metadata Element Set (DCMES); it is proposed as the minimum number of metadata elements required to facilitate the discovery of document-like objects in a networked environment such as the Internet (see Table 1.1, which shows some of the elements in DC).

An example of using DC is shown in List 1.1. As shown in List 1.1, a HTML `<meta>` tag is where an item of metadata about a Web page is stored. A given Web document may contain many `<meta>` tags to represent any number of metadata items.

TABLE 1.1
Element Examples in Dublin Core Metadata Schema

Element Name	Element Description
Creator	This element represents the person or organization responsible for creating the content of the resource; e.g., authors in the case of written documents
Publisher	This element represents the entity responsible for making the resource available in its present form; it can be a publishing house, a university department, etc.
Contributor	This element represents the person or organization not specified in a creator element who has made significant intellectual contributions to the resource but whose contribution is secondary to any person or organization specified in a creator element; e.g., editor, transcriber, illustrator
Title	This element represents the name given to the resource, usually by the creator
Subject	This element represents the topic of the resource; normally, it will be expressed as keywords or phrases that describe the subject or content of the resource
Date	This element represents the date associated with the creation or availability of the resource
Identifier	This element is a string or number uniquely identifies the resource; examples include URLs, Purls, ISBN, or other formal names
Description	This element is a free text description of the content of the resource; it can be a flexible format, including abstracts or other content descriptions
Language	This element represents the language used by the document
Format	This element identifies the data format of the document; this information can be used to identify the software that might be needed to display or operate the resource; e.g., postscript, HTML, text, jpeg, XML

LIST 1.1
An Example of Using DC Metadata

```
<html>
<head>
<title>a joke written by liyang</title>
<meta name="DC.Title" content="a joke written by Liyang">
<meta name="DC.Creator" content="a joke">
<meta name="DC.Type" content="text">
<meta name="DC.Data" content="2004">
<meta name="DC.Format" content="text/html">
<meta name="DC.Identifier" content= http://www.codeproject.com/
script/profile/whos_who.asp?id=736920">
</head>
<body>
I decided to make my first son a medical doctor so that later on when
I am old and sick I can get medical care any time I need and for
free … in fact, better to make my second son a medical doctor, too,
so I can get a second opinion.
</body>
</html>
```

Normally, these metadata are not displayed by the Web browser. They are mainly intended to be read by automated agents or tools.

You may wonder how much benefit the DC schema will give us; true, metadata is important, but if all the metadata that is added to a Web document only follows DC schema, then it would be a little boring. After all, DC schema only provides metadata that gives some very general information about the document. How can it help us realize the dream of the Semantic Web?

You are right. In the coming chapters, we will discuss some much more powerful schemas and tools that contain much more detailed information than DC schema can ever provide. However, what is important is that all the extra information exists in the form of metadata; metadata is the building block we use when we add some extra data (meaning) to an existing document. Let us summarize what we have discussed so far:

- The Semantic Web is an extension of the current Web; its main goal is to allow machine processing in a global scale.
- One way to accomplish this is to add metadata to the Web, as metadata is structured data, i.e., it is machine readable.
- DC schema seems simple, but it shows the key idea of adding metadata (meanings) to a given document.

The final issue we want to address in this chapter (about which you have probably already wondered) is the question about how the metadata gets there. We already have so many documents on the Web that do not have metadata; how are we going to add metadata to them? We do not own them, and we cannot force the owners to add metadata to them either. In the coming chapters, we will discuss this question in much more detail; here we present some basic considerations.

1.3.2 Metadata Considerations

1.3.2.1 Embedding the Metadata in Your Page

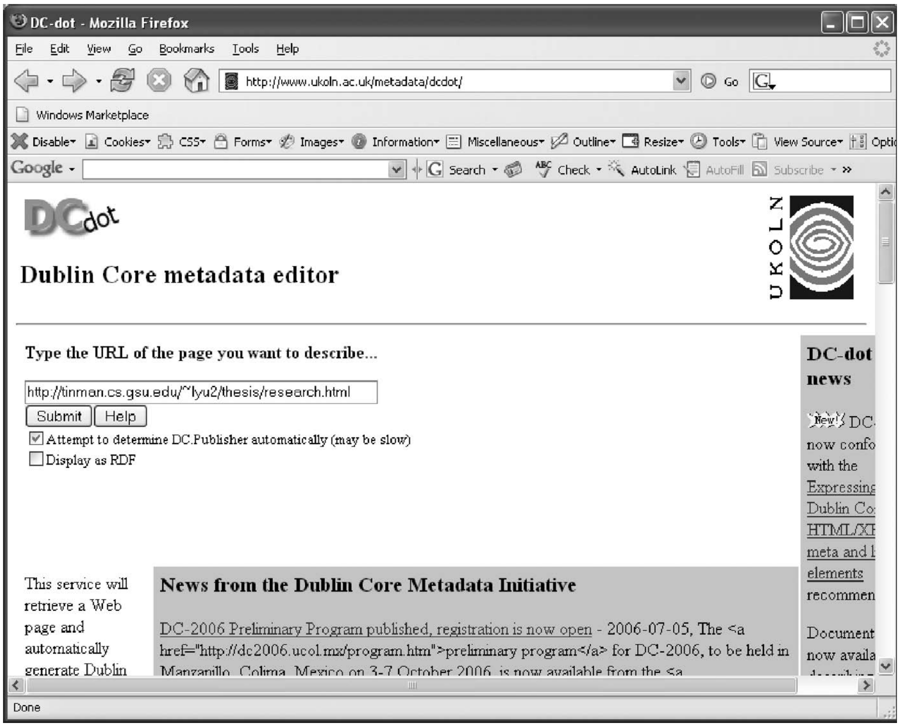
The easiest thing to do is to embed the metadata directly in your page when you create it — just use the `<meta>` tag in the `<head>` section. This is indeed a good practice that one should follow when publishing on the Web. Also, the added metadata should be prepared with the following assumption in mind: there might exist some automated agents or tools that can do something useful with the added metadata.

1.3.2.2 Using Metadata Tools to Add Metadata to Existing Pages

Another choice is to use a metadata tool to create metadata for an existing Web page. For example, you can find such a tool at:

```
http://www.ukoln.ac.uk/metadata/dcdot/
```

It will read the page you submit and automatically generate DC metadata for you. Figure 1.1 shows the interface of this tool, and the submitted Uniform Resource Locator (URL) is a page from http://tinman.cs.gsu.edu/~lyu2, my research project.


DC-dot - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ukoln.ac.uk/metadata/dcdot/
Go
Windows Marketplace
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Optic
Google Search Check AutoLink AutoFill Subscribe
DCdot
UKOLN
Submit
Help
Done

Dublin Core metadata editor

Type the URL of the page you want to describe...

http://tinman.cs.gsu.edu/~lyu2/thesis/research.html

☑ Attempt to determine DC.Publisher automatically (may be slow)
☐ Display as RDF

This service will retrieve a Web page and automatically generate Dublin

News from the Dublin Core Metadata Initiative

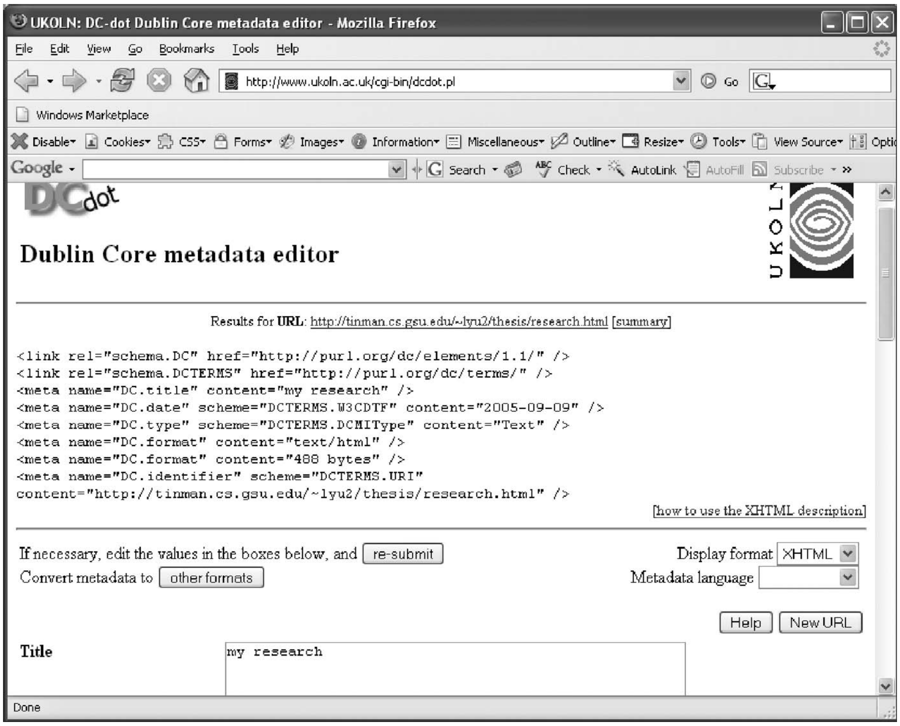
DC-2006 Preliminary Program published, registration is now open - 2006-07-05, The <a href="http://dc2006.ucol.mx/program.htm">preliminary program</a> for DC-2006, to be held in Manzanillo, Colima, Mexico on 3-7 October 2006, is now available from the <a

DC-dot news

New! DC now confo with the Expressing Dublin Co HTML/XH meta and l elements recommen

Document now availa

FIGURE 1.1 DCdot can be used to generate DC metadata for the page you submit.


UKOLN: DC-dot Dublin Core metadata editor - Mozilla Firefox
http://www.ukoln.ac.uk/cgi-bin/dcdot.pl
DCdot
Dublin Core metadata editor
UKOLN

Results for URL: http://tinman.cs.gsu.edu/~lyu2/thesis/research.html [summary]

```
<link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
<link rel="schema.DCTERMS" href="http://purl.org/dc/terms/" />
<meta name="DC.title" content="my research" />
<meta name="DC.date" scheme="DCTERMS.W3CDTF" content="2005-09-09" />
<meta name="DC.type" scheme="DCTERMS.DCMIType" content="Text" />
<meta name="DC.format" content="text/html" />
<meta name="DC.format" content="488 bytes" />
<meta name="DC.identifier" scheme="DCTERMS.URI"
content="http://tinman.cs.gsu.edu/~lyu2/thesis/research.html" />
```

[how to use the XHTML description]

If necessary, edit the values in the boxes below, and re-submit

Convert metadata to other formats

Display format XHTML

Metadata language

Title: my research

FIGURE 1.2 The DC metadata generated by DCdot.

Push the `Submit` button, and you will get the following output, as shown in Figure 1.2.

The problem with this solution is that you have to visit the Web pages one by one to generate the metadata, and the metadata that is generated is only DC metadata, which may not be enough for the applications you have in mind (as discussed in later chapters). Also, the generated metadata cannot be really added to the page itself, because you normally do not have access to it; you need to figure out some other place to store them.

1.3.2.3 Using a Text-Parsing Crawler to Create Metadata

This idea is based on the working of a crawler (we will discuss crawlers in more detail in Chapter 2). Once the crawler reaches a page and finds that it does not have any metadata, it attempts to discover some meaningful information by scanning through the text and creates some metadata for the page. For instance, the crawler may have a special table containing all the important keywords that it is looking for (these words may, for example, be some important terminologies in the area of bioinformatics), and on finding these words in the current page, the crawler starts to learn something about the page, and it writes what it learns into the metadata. This is certainly just one hypothetical case of using a crawler to create the metadata, but the point is clear. As the crawler is not able to really add the metadata to the page, there is the issue of how and where to store the generated metadata.

We have now gained enough knowledge about metadata and are ready to move on. As a summary, if a resource (such as a Web page) is important enough, then it might be useful to describe it with some metadata. In the area of the Semantic Web, the metadata is used to add meaning to the page, and this structured data can be easily understood by machines. Now, you can see why we need to cover the topic of metadata at this point, and you start to realize the fundamental relationship between metadata and the Semantic Web. Metadata provides the essential link between the page content and content meaning.

In Chapter 2, we will study how a search engine works in both traditional Web and Semantic Web environments; the goal is to gain a much better understanding of the Semantic Web. Also, you will begin to appreciate the value of metadata.