Chapter 1. RDF: An Introduction

The Resource Description Framework (RDF) is an extremely flexible technology, capable of addressing a wide variety of problems. Because of its enormous breadth, people often come to RDF thinking that it's one thing and find later that it's much more. One of my favorite parables is about the blind people and the elephant. If you haven't heard it, the story goes that six blind people were asked to identify what an elephant looked like from touch. One felt the tusk and thought the elephant was like a spear; another felt the trunk and thought the elephant was like a snake; another felt a leg and thought the elephant was like a tree; and so on, each basing his definition of an elephant on his own unique experiences.

RDF is very much like that elephant, and we're very much like the blind people, each grabbing at a different aspect of the specification, with our own interpretations of what it is and what it's good for. And we're discovering what the blind people discovered: not all interpretations of RDF are the same. Therein lies both the challenge of RDF as well as the value.

Tip

The main RDF specification web site is at http://www.w3.org/RDF/. You can access the core working group's efforts at http://www.w3.org/2001/sw/RDFCore/. In addition, there's an RDF Interest Group forum that you can monitor or join at *http://www.w3.org/RDF/Interest/*.

The Semantic Web and RDF: A Brief History

RDF is based within the Semantic Web effort. According to the W3C (World Wide Web Consortium) Semantic Web Activity Statement:

> The Resource Description Framework (RDF) is a language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web. RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange.

Though not as well known as other specifications from the W3C, RDF is actually one of the older specifications, with the first working draft produced in 1997. The earliest editors, Ora Lassila and Ralph Swick, established the foundation on which RDF rested—a mechanism for working with metadata that promotes the interchange of data between automated processes. Regardless of the transformations RDF has undergone and its continuing maturing process, this statement forms its immutable purpose and focal point.

In 1999, the first recommended RDF specification, the RDF Model and Syntax Specification (usually abbreviated as RDF M&S), again coauthored by Ora Lassila and Ralph Swick, was released. A candidate recommendation for the RDF Schema Specification, coedited by Dan Brickley and R.V. Guha, followed in 2000. In order to open up a previously closed specification process, the W3C also created the RDF Interest Group, providing a view into the RDF specification process for interested people who were not a part of the RDF Core Working Group.

As efforts proceeded on the RDF specification, discussions continued about the concepts behind the Semantic Web. At the time, the main difference between the existing Web and the newer, smarter

Web is that rather than a large amount of disorganized and not easily accessible data, something such as RDF would allow organization of data into knowledge statements—assertions about resources accessible on the Web. From a *Scientific American* article published May 2001, Tim Berners-Lee wrote:

> The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. Such an agent coming to the clinic's Web page will know not just that the page has keywords such as "treatment, medicine, physical, therapy" (as might be encoded today) but also that Dr. Hartman works at this clinic on Mondays, Wednesdays and Fridays and that the script takes a date range in *yyyy-mm-dd* format and returns appointment times.

As complex as the Semantic Web sounds, this statement of Berners-Lee provides the key to understanding the Web of the future. With the Semantic Web, not only can we find data about a subject, we can also infer additional material not available through straight keyword search. For instance, RDF gives us the ability to discover that there is an article about the Giant Squid at one of my web sites, and that the article was written on a certain date by a certain person, that it is associated with three other articles in a series, and that the general theme associated with the article is the Giant Squid's earliest roots in mythology. Additional material that can be derived is that the article is still "relevant" (meaning that the data contained in the article hasn't become dated) and still active (still accessible from the Web). All of this information is easily produced and consumed through the benefits of RDF without having to rely on any extraordinary computational power.

However, for all of its possibilities, it wasn't long after the release of the RDF specifications that concerns arose about ambiguity with certain constructs within the document. For instance, there was considerable discussion in the RDF Internet Group about containers—are separate semantic and syntactic constructs really needed?—as well as other elements within RDF/XML. To meet this growing number of concerns, an RDF Issue Tracking document was started in 2000 to monitor issues with RDF. This was followed in 2001 with the creation of a new RDF Core Working Group, chartered to complete the RDF Schema (RDFS) recommendation as well as address the issues with the first specifications.

The RDF Core Working Group's scope has grown a bit since its beginnings. According to the Working Group's charter, they must now:

- Update and maintain the RDF Issue Tracking document
- Publish a set of machine-processable test cases corresponding to technical issues addressed by the WG
- Update the errata and status pages for the RDF specifications
- Update the RDF Model and Syntax Specification (as one, two, or more documents) clarifying the model and fixing issues with the syntax
- Complete work on the RDF Schema 1.0 Specification
- Provide an account of the relationship between RDF and the XML family of technologies
- Maintain backward compatibility with existing implementations of RDF/XML

The WG was originally scheduled to close down early in 2002, but, as with all larger projects, the work slid until later in 2002. This book finished just as the WG issued the W3C Last Call drafts for all six of the RDF specification documents, early in 2003.

The Specifications

As stated earlier, the RDF specification was originally released as one document, the RDF Model and Syntax, or RDF M&S. However, it soon became apparent that this document was attempting to cover too much material in one document, and leaving too much confusion and too many questions in its wake. Thus, a new effort was started to address the issues about the original specification and, hopefully, eliminate the confusion. This work resulted in an updated specification and the release of six new documents: RDF Concepts and Abstract Syntax, RDF Semantics, RDF/XML Syntax Specification (revised), RDF Vocabulary Description Language 1.0: RDF Schema, the RDF Primer, and the RDF Test Cases.

The RDF Concepts and Abstract Syntax and the RDF Semantics documents provide the fundamental framework behind RDF: the underlying assumptions and structures that makes RDF unique from other metadata models (such as the relational data model). These documents provide both validity and consistency to RDF—a way of verifying that data structured in a certain way will always be compatible with other data using the same structures. The RDF model exists independently of any representation of RDF, including RDF/XML.

The RDF/XML syntax, described in the RDF/XML Syntax Specification (revised), is the recommended serialization technique for RDF. Though several tools and APIs can also work with N-Triples (described in Chapter 2) or N3 notation (described in Chapter 3), most implementation of and discussion about RDF, including this book, focus on RDF/XML

The RDF Vocabulary Description Language defines and constrains an RDF/XML vocabulary. It isn't a replacement for XML Schema or the use of DTDs; rather, it's used to define specific RDF vocabularies; to specify how the elements of the vocabulary relate to each other. An RDF Schema isn't required for valid RDF (neither is a W3C XML Schema or an XML 1.0 Document Type Definition—DTD), but it does help prevent confusion when people want to share a vocabulary.

A good additional resource to learn more about RDF and RDF/XML is the RDF Primer. In addition to examples and accessible descriptions of the concepts of RDF and RDFS, the primer also, looks at some uses of RDF. I won't be covering the RDF Primer in this book because its use is somewhat self-explanatory. However, the primer is an excellent complement to this book, and I recommend that you spend time with it either while you're reading this book or afterward if you want another viewpoint on the topics covered.

The final RDF specification document, RDF Test Cases, contains a list of issues arising from the original RDF specification release, their resolutions, and the test cases devised for use by RDF implementers to test their implementations against these resolved issues. The primary purpose of the RDF Test Cases is to provide examples for testing specific RDF issues as the Working Group resolved them. Unless you're writing an RDF/XML parser or something similar, you probably won't need to spend much time with that document, and I won't be covering it in the book.

When to Use and Not Use RDF

RDF is a wonderful technology, and I'll be at the front in its parade of fans. However, I don't consider it a replacement for other technologies, and I don't consider its use appropriate in all circumstances. Just because data is on the Web, or accessed via the Web, doesn't mean it has to be organized with RDF. Forcing RDF into uses that don't realize its potential will only result in a general push back against RDF in its entirety—including push back in uses in which RDF positively shines.

This, then, begs the question: when should we, and when should we not, use RDF? More specifically, since much of RDF focuses on its serialization to RDF/XML, when should we use RDF/XML and when should we use non-RDF XML?

As the final edits for this book were in progress, a company called Semaview published a graphic depicting the differences between XML and RDF/XML (found at http://www.semaview.com/c/RDFvsXML.html). Among those listed was one about the tree-structured nature of XML, as compared to RDF's much flatter triple-based pattern. XML is hierarchical, which means that all related elements must be nested within the elements they're related to. RDF does not require this nested structure.

To demonstrate this difference, consider a web resource, which has a history of movement on the Web. Each element in that history has an associated URL, representing the location of the web resource after the movement has occurred. In addition, there's an associated reason why the resource was moved, resulting in this particular event. Recording these relationships in non-RDF XML results in an XML hierarchy four layers deep:

```
<?xml version="1.0"?>
<resource>
  <uri>http://burningbird.net/articles/monsters3.htm</uri>
  <history>
    <movement>
       <link>http://www.yasd.com/dynaearth/monsters3.htm</link>
       <reason>New Article</reason>
    </movement>
  </history>
</resource>
```

In RDF/XML, you can associate two separate XML structures with each other through a Uniform Resource Identifier (URI, discussed in Chapter 2). With the URI, you can link one XML structure to another without having to embed the second structure directly within the first:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pstcn="http://burningbird.net/postcon/elements/1.0/"
  xml:base="http://burningbird.net/articles/">

  <pstcn:Resource rdf:about="monsters3.htm">

<!--resource movements-->
     <pstcn:history>
       <rdf:Seq>
        <rdf:_3 rdf:resource="http://www.yasd.com/dynaearth/monsters3.htm" />
      </rdf:Seq>
     </pstcn:history>
```

```
  </pstcn:Resource>

  <pstcn:Movement rdf:about="http://www.yasd.com/dynaearth/monsters3.htm">
     <pstcn:movementType>Add</pstcn:movementType>
     <pstcn:reason>New Article</pstcn:reason>
  </pstcn:Movement>

</rdf:RDF>
```

Ignore for the moment some of the other characteristics of RDF/XML, such as the use of namespaces, which we'll get into later in the book, and focus instead on the structure. The RDF/XML is still well-formed XML—a requirement of RDF/XML—but the use of the URI (in this case, the URL `"http://www.yasd.com/dynaearth/monsters3.htm"`) breaks us out of the forced hierarchy of standard XML, but still allows us to record the relationship between the resource's history and the particular movement.

However, this difference in structure can make it more difficult for people to read the RDF/XML document and actually see the relationships between the data, one of the more common complaints about RDF/XML. With non-RDF XML, you can, at a glance, see that the history element is directly related to this specific resource element and so on. In addition, even this small example demonstrates that RDF adds a layer of complexity on the XML that can be off-putting when working with it manually. Within an automated process, though, the RDF/XML structure is actually an advantage.

When processing XML, an element isn't actually complete until you reach its end tag. If an application is parsing an XML document into elements in memory before transferring them into another persisted form of data, this means that the elements that contain other elements must be retained in memory until their internal data members are processed. This can result in some fairly significant strain on memory use, particularly with larger XML documents.

RDF/XML, on the other hand, would allow you to process the first element quickly because its "contained" data is actually stored in another element somewhere else in the document. As long as the relationship between the two elements can be established through the URI, we'll always be able to reconstruct the original data regardless of how it's been transformed.

Another advantage to the RDF/XML approach is when querying the data. Again, in XML, if you're looking for a specific piece of data, you basically have to provide the entire structure of all the elements preceding the piece of data in order to ensure you have the proper value. As you'll see in RDF/XML, all you have to do is remember the triple nature of the specification, and look for a triple with a pattern matching a specific resource URI, such as a property URI, and you'll find the specific value. Returning to the RDF/XML shown earlier, you can find the reason for the specific movement just by looking for the following pattern:

```
<http://www.yasd.com/dynaearth/monsters3.htm> pstcn:reason ?
```

The entire document does not have to be traversed to answer this query, nor do you have to specify the entire element path to find the value.

Tip

If you've worked with database systems before, you'll recognize that many of the differences between RDF/XML and XML are similar to the differences between relational and hierarchical

databases. Hierarchical databases also have a physical location dependency that requires related data to be bilocated, while relational databases depend on the use of identifiers to relate data.

Another reason you would use RDF/XML over non-RDF XML is the ability to join data from two disparate vocabularies easily, without having to negotiate structural differences between the two. Since the XML from both data sets is based on the same model (RDF) and since both make use of namespaces (which prevent element name collision—the same element name appearing in both vocabularies), combining data from both vocabularies can occur immediately, and with no preliminary work. This is essential for the Semantic Web, the basis for the work on RDF and RDF/XML. However, this is also essential in any business that may need to combine data from two different companies, such as a supplier of raw goods and a manufacturer that uses these raw goods. (Read more on this in the sidebar Data Handshaking Through the Ages).

As excellent as these two reasons (less strain on memory and joining vocabularies) are for utilizing RDF as a model for data and RDF/XML as a format, for certain instances of data stored on the Web, RDF is clearly not a replacement. As an example, RDF is not a replacement for XHTML for defining web pages that are displayed in a browser. RDF is also not a replacement for CSS, which is used to control how that data is displayed. Both CSS and XHTML are optimized for their particular uses, organizing and displaying data in a web browser. RDF's purpose differs—it's used to capture specific statements about a resource, statements that help form a more complete picture of the resource. RDF isn't concerned about either page organization or display.

Now, there might be pieces of information in the XHTML and the CSS that could be reconstructed into statements about a resource, but there's nothing in either technology that specifically says "this is a statement, an assertion if you will, about this resource" in such a way that a machine can easily pick this information out. That's where RDF enters the picture. It lays all assertions out—bang, bang, bang—so that even the most amoeba-like RDF parser can find each individual statement without having to pick around among the presentational and organizational constructs of specifications such as XHTML and CSS.

Additionally, RDF/XML isn't necessarily well suited as a replacement for other uses of XML, such as within SOAP or XML-RPC. The main reason is, again, the level of complexity that RDF/XML adds to the process. A SOAP processor is basically sending a request for a service across the Internet and then processing the results of that request when it's answered. There's a mechanism that supports this process, but the basic structure of SOAP is request service, get answer, process answer. In the case of SOAP, the request and the answer are formatted in XML.

Though a SOAP service call and results are typically formatted in XML, there really isn't the need to persist these outside of this particular invocation, so there really is little drive to format the XML in such a way that it can be combined with other vocabularies at a later time, something that RDF/XML facilitates. Additionally, one hopes that we keep the SOAP request and return as small, lightweight, and uncomplicated answers as possible, and RDF/XML does add to the overhead of the XML. Though bandwidth is not the issue it used to be years ago, it is still enough of an issue to not waste it unnecessarily.

Ultimately, the decision about using RDF/XML in place of XML is based on whether there's a good reason to do so—a business rather than a technical need to use the model and related XML structure. If the data isn't processed automatically, if it isn't persisted and combined with data from

other vocabularies, and if you don't need RDF's optimized querying capability, then you should use non-RDF XML. However, if you do need these things, consider the use of RDF/XML.

Data Handshaking Through the Ages

I started working with data and data interchange at Boeing in the late 1980s. At that time, there was a data definition effort named Product Data Exchange Specification (PDES) underway between several manufacturing companies to define one consistent data model that could be used by all of them. With this model, the companies hoped to establish the ability to interchange data among themselves without having to renegotiate data structures every time a new connection was made between the companies, such as adding a new supplier or customer. (This effort is still underway and you can read more about it at http://pdesinc.com.)

PDES was just one effort on the part of specific industries to define common business models that would allow them to interoperate. From Boeing, I went to Sierra Geophysics, a company in Seattle that created software for the oil industry. Sierra Geophysics and its parent company, Halliburton, Inc., were hard at work on POSC, an effort similar to PDES but geared to the oil and gas industries. (You can read more about POSC at http://posc.org; be sure to check out POSC's use of XML, specifically, at http://posc.org/ebiz/xmlLive.shtml.)

One would think this wouldn't be that complex, but it is almost virtually impossible to get two companies to agree on what "data" means. Because of this difficulty, to this day, there's never been complete agreement as to data interchange formats, though with the advent of XML, there was hope that this specification would provide a syntax that most of the companies could agree to use. One reason XML was hailed as a potential savior is that it represented a neutral element in the discussions—no one could claim either the syntax or the syntactic rules.

Would something like RDF/XML work for both of these organizations and their efforts? Yes and no. If the interest in XML is primarily for network protocol uses, I wouldn't necessarily recommend the use of RDF/XML for the same reasons I wouldn't recommend its use with SOAP and XML/RPC—RDF/XML adds a layer of complexity and overhead that can be counterproductive when you're primarily doing nothing more than just sending messages to and from services. However, RDF/XML would fit the needs of POSC and PDES if the interest were on merging data between organizations for more effective supply chain management—in effect, establishing a closer relationship between the supplier of raw goods on one hand and a manufacturer of finished goods on the other. In particular, with an established ontology built on RDF/XML (ontologies are discussed in Chapter 12) defining the business data, it should be a simple matter to add new companies into an existing supply chain.

When one considers that much of the cost of a manufactured item resides in the management of the supply chain and within the manufacturing process, not in the raw material used to manufacture the item, I would expect to see considerable progress from industry efforts such as POSC and PDES in RDF/XML.

Some Uses of RDF/XML

The first time I saw RDF/XML was when it was used to define the table of contents (TOC) structures within Mozilla, when Mozilla was first being implemented. Since then, I've been both surprised and pleased at how many implementations of RDF and RDF/XML exist.

One of the primary users of RDF/XML is the W3C itself, in its effort to define a Web Ontology Language based on RDF/XML. Being primarily a data person and not a specialist in markup, I wasn't familiar with some of the concepts associated with RDF when I first started exploring its use and meaning. For instance, there were references to *ontology* again and again, and since my previous exposure to this word had to do with biology, I was a bit baffled. However, *ontology* in the sense of RDF and the Semantic Web is, according to dictionary.com, "An explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them."

As mentioned previously, RDF provides a structure that allows us to make assertions using XML (and other serialization techniques). However, there is an interest in taking this further and expanding on it, by creating just such an ontology based on the RDF model, in the interest of supporting more advanced agent-based technologies. An early effort toward this is the DARPA Agent Markup Language program, or DAML. The first implementation of DAML, DAML+OIL, is tightly integrated with RDF.

A new effort at the W3C, the Web Ontology Working Group, is working on creating a Web Ontology Language (OWL) derived from DAML+OIL and based in RDF/XML. The following quote from the OWL Use Cases and Requirements document, one of many the Ontology Working Group is creating, defines the relationship between XML, RDF/XML, and OWL:

> The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The next element required for the Semantic Web is a Web ontology language which can formally describe the semantics of classes and properties used in web documents. In order for machines to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema.

Drawing analogies from other existing data schemes, if RDF and the relational data model were comparable, then RDF/XML is also comparable to the existing relational databases, and OWL would be comparable to the business domain applications such as PeopleSoft and SAP. Both PeopleSoft and SAP make use of existing data storage mechanisms to store the data and the relational data model to ensure that the data is stored and managed consistently and validly; the products then add an extra level of business logic based on patterns that occur and reoccur within traditional business processes. This added business logic could be plugged into a company's existing infrastructure without the company having to build its own functionality to implement the logic directly.

OWL does something similar except that it builds in the ability to define commonly reoccurring inferential rules that facilitate how data is queried within an RDF/XML document or store. Based on this added capability, and returning to the RDF/XML example in the last section, instead of being limited to queries about a specific movement based on a specific resource, we could query on movements that occurred because the document was moved to a new domain, rather than because the document was just moved about within a specific domain. Additional information can then allow us to determine that the document was moved because it was transferred to a different owner, allowing us to infer information about a transaction between two organizations even if this "transactional" information isn't stored directly within elements.

In other words, the rules help us discover new information that isn't necessarily stored directly within the RDF/XML.

Tip

Chapter 12 covers ontologies, OWL, and its association with RDF/XML. Read more about the W3C's ontology efforts at http://www.w3.org/2001/sw/WebOnt/. The Use Cases and Requirements document can be found at http://www.w3.org/TR/webont-req/.

Another very common use of RDF/XML is in a version of RSS called RSS 1.0 or RDF/RSS. The meaning of the RSS abbreviation has changed over the years, but the basic premise behind it is to provide an XML-formatted feed consisting of an abstract of content and a link to a document containing the full content. When Netscape originally created the first implementation of an RSS specification, RSS stood for RDF Site Summary, and the plan was to use RDF/XML. When the company released, instead, a non-RDF XML version of the specification, RSS stood for Rich Site Summary. Recently, there has been increased activity with RSS, and two paths are emerging: one considers RSS to stand for Really Simple Syndication, a simple XML solution (promoted as RSS 2.0 by Dave Winer at Userland), and one returns RSS to its original roots of RDF Site Summary (RSS 1.0 by the RSS 1.0 Development group).

RSS *feeds*, as they are called, are small, brief introductions to recently released news articles or weblog postings (weblogs are frequently updated journals that may include links to other stories, comments, and so on). These feeds are picked up by *aggregators*, which format the feeds into human consumable forms (e.g., as web pages or audio notices). RSS files normally contain only the most recent feeds, newer items replacing older ones.

Given the transitory nature of RSS feeds as I just described them, it is difficult to justify the use of RDF for RSS. If RDF's purpose is to record assertions about resources that can be discovered and possibly merged with other assertions to form a more complete picture of the resource, then that implies some form of permanence to this data, that the data hangs around long enough to be discovered. If the data has a life span of only a minute, hour, or day, its use within a larger overall "semantic web" tends to be dubious, at best.

However, the data contained in the RSS feeds—article title, author, date, subject, excerpt, and so on—is a very rich source of information about the resource, be it article or weblog posting, information that isn't easily scraped from the web page or pulled in from the HTML `meta` tags. Additionally, though the purpose of the RSS feed is transitory in nature, there's no reason tools can't access this data and store it in a more permanent form for mergence with other data. For instance, I've long been amazed that search tools don't use RSS feeds rather than the HTML pages themselves for discovering information.

Based on these latter views of RSS, there is, indeed, a strong justification for building RSS within an RDF framework—to enhance the discovery of the assertions contained within the XML. The original purpose of RSS might be transitory, but there's nothing to stop others from pulling the data into more permanent storage if they so choose or to use the data for other purposes.

I'll cover the issue of RSS in more detail in Chapter 13, but for now the point to focus on is that when to use RDF isn't always obvious. The key to knowing when to make extra effort necessary to overlay an RDF model on the data isn't necessarily based on the original purpose for the data or

even the transitory nature of the data—but on the data itself. If the data is of interest, descriptive, and not easily discovered by any other means, little RDF alarms should be ringing in our minds.

As stated earlier, if RDF isn't a replacement for some technologies, it is an opportunity for new ones. In particular, Mozilla, my favorite open source browser, uses RDF extensively within its architecture, for such things as managing table of contents structures. RDF's natural ability to organize XML data into easily accessible data statements made it a natural choice for the Mozilla architects. Chapter 14 explores how RDF/XML is used within the Mozilla architecture, in addition to its use in other open source and noncommercial applications such as MIT's DSpace, a tool and technology to track intellectual property, and FOAF, a toolkit for describing the connections between people.

Chapter 15 follows with a closer look at the commercial use of RDF, taking a look at OSA's Chandler, Plugged In Software's Tucana Knowledge Store, Siderean Software's Seamark, the Intellidimension RDF Gateway, and how Adobe is incorporating RDF data into its products.

Related Technologies

Several complementary technologies are associated with RDF. As previously discussed, the most common technique to serialize RDF data is via RDF/XML, so influences on XML are likewise influences on RDF. However, other specifications and technologies also impact on, and are impacted by, the ongoing RDF efforts.

Though not a requirement for RDF/XML, you can use XML Schemas and DTDs to formalize the XML structure used within a specific instance of RDF/XML. There's also been considerable effort to map XML Schema data types to RDF, as you'll see in the next several chapters.

One issue that arises again and again with RDF is where to include the XML. For instance, if you create an RDF document to describe an HTML page resource, should the RDF be in a separate file or contained within the HTML document? I've seen RDF embedded in HTML and XML using a variety of tricks, but the consensus seems to be heading toward defining the RDF in a separate file and then linking it within the HTML or XHTML document. Chapter 3 takes a closer look at issues related to merging RDF with other formats.

A plethora of tools and utilities work with RDF/XML. Chapter 7 covers some of these. In addition, several different APIs in a variety of languages, such as Perl, Java, Python, C, C++, and so on, can parse, query, and generate RDF/XML. The remainder of the second section of the book explores some of the more stable or representative of these, including a look at Jena, a Java-based API, RAP (RDF API for PHP), Redland's multilanguage RDF API, Perl and Python APIs and tools, and so on.

Going Forward

The RDF Core Working Group spent considerable time ensuring that the RDF specifications answered as many questions as possible. There is no such thing as a perfect specification, but the group did its best under the constraints of maintaining connectivity with its charter and existing uses of RDF/XML.

RDF/XML has been used enough in so many different applications that I consider it to be at a release level with the publication of the current RDF specification documents. In fact, I think you'll

find that the RDF specification will be quite stable in its current form after the documents are released—it's important that the RDF specification be stabilized so that we can begin to build on it. Based on this hoped-for stability, you can use the specification, including the RDF/XML, in your applications and be comfortable about future compatibility.

We're also seeing more and more interest in and use of RDF and its associated RDF/XML serialization in the world. I've seen APIs in all major programming languages, including Java, Perl, PHP, Python, C#, C++, C, and so on. Not only that, but there's a host of fun and useful tools to help you edit, parse, read, or write your RDF/XML documents. And most of these tools, utilities, APIs, and so on are free for you to download and incorporate into your current work.

With the release of the RDF specification documents, RDF's time has come, and I'm not just saying that because I wrote this book. I wrote this book because I believe that RDF is now ready for prime time.

Chapter 2. RDF: Heart and Soul

RDF's purpose is fairly straightforward: it provides a means of recording data in a machine-understandable format, allowing for more efficient and sophisticated data interchange, searching, cataloging, navigation, classification, and so on. It forms the cornerstone of the W3C effort to create the Semantic Web, but its use isn't restricted to this specific effort.

Perhaps because RDF is a description for a data model rather than a description of a specific data vocabulary, or perhaps because it has a foothold in English, logic, and even in human reasoning, RDF has a strong esoteric element to it that can be intimidating to a person wanting to know a little more about it. However, RDF is based on a well-defined set of rules and constraints that governs its format, validity, and use. Approaching RDF through the specifications is a way of grounding RDF, putting boundaries around the more theoretical concepts.

The chapter takes a look at two RDF specification documents that exist at opposite ends of the semantic spectrum: the RDF Concepts and Abstract Model and the RDF Semantics documents. In these documents we're introduced to the concepts and underlying strategy that form the basis of the RDF/XML that we'll focus on in the rest of the book. In addition, specifically within the Semantics document, we'll be exposed to the underlying meaning behind each RDF construct. Though not critical to most people's use of RDF, especially RDF/XML, the Semantics document ensures that all RDF consumers work from the same basic understanding; therefore, some time spent on this document, primarily in overview, is essential.

Both documents can be accessed directly online, so I'm not going to duplicate the information contained in them in this chapter. Instead, we'll take a look at some of the key elements and unique concepts associated with RDF.

Tip

The RDF Concepts and Abstract Syntax document can be found at http://www.w3.org/TR/rdf-concepts/. The RDF Semantics document can be found at http://www.w3.org/TR/rdf-mt/.

The Search for Knowledge

Occasionally, I like to write articles about non-Internet-related topics, such as marine biology or astronomy. One of my more popular articles is on *Architeuthis Dux*—the giant squid. The article is currently located at http://burningbird.net/articles/monsters1.htm.

According to the web profile statistics for this article, it receives a lot of visitors based on searches performed in Google, a popular search engine. When I go to the Google site, though, to search for the article based on the term *giant squid*, I find that I get a surprising number of links back. The article was listed on page 13 of the search results (with 10 links to a page). First, though, were several links about a production company, the Jules Verne novel *10,000 Leagues Under the Sea*, something to do with a comic book character called the Giant Squid, as well as various other assorted and sundry references such as a recipe for cooking giant squid steaks (as an aside, giant squids are ammonia based and inedible).

For the most part, each link does reference the giant squid as a marine animal; however, the context doesn't match my current area of interest: finding an article that explores the giant squid's roots in mythology.

I can refine my search, specifying separate keywords such as *giant*, *squid*, and *mythology* to make my article appear on page 6 of the list of links—along with links to a Mexican seafood seller offering giant squid meat slabs and a listing of books that discuss a monster called the Giant Squid that oozes green slime.

The reason we get so many links back when searching for specific resources is that most search engines use keyword-based search engine functionality, rather than searching for a resource *within the context of a specific interest*. The search engines' data is based on the use of automated agents or robots and web spiders that traverse the Web via in-page links, pulling keywords from either HTML `meta` tags or directly from the page contents themselves.

A better approach for classifying resources such as the giant squid article would be to somehow attach information about the context of the resource. For instance, the article is part of a series comparing two legendary creatures: the giant squid and the Loch Ness Monster. It explores what makes a creature legendary, as well as current and past efforts to find living representatives of either creature. All of this information forms a description of the resource, a *picture* that's richer and more complex than a one-dimensional keyword-based categorization.

What's missing in today's keyword-based classification of web resources is the ability to record statements about a resource. Statements such as:

- The article's title is "*Architeuthis Dux*."
- The article's author is Shelley Powers.
- The article is part of a series.
- A related article is ...
- The article is about the giant squid and its place in the legends.

General keyword scanning doesn't return this type of specific information, at least, not in such a way that a machine can easily find and process these statements without heroic computations.

RDF provides a mechanism for recording statements about resources so that machines can easily interpret the statements. Not only that, but RDF is based on a domain-neutral model that allows one set of statements to be merged with another set of statements, even though the information contained in each set of statements may differ dramatically.

One application's interest in the resource might focus on finding new articles posted on the Web and providing an encapsulated view of the articles for news aggregators. Another application's interest might be on the article's long-term relevancy and the author of the article, while a third application may focus specifically on the topics covered in the article, and so on. Rather than generating one XML file in a specific XML vocabulary for all of these different applications' needs, one RDF file can contain all of this information, and each application can pick and choose what it needs. Better yet, new applications will find that everything they need is already being provided, as the information we record about each resource gets richer and more comprehensive.

And the basis of all this richness is a simple little thing called the RDF triple.

Note

I use the word *context* in this chapter and throughout the book. However, the folks involved with RDF, including Tim Berners-Lee, director of the W3C, are hesitant about using the term *context* in association with RDF. The main reason is there's a lot of confusion about what *context* actually means. Does it mean the world of all possible conditions at any one point? Does it mean a specific area of interest?

To prevent confusion when I use *context* in the book, I use the term to refer to a certain aspect of a subject at a given time. For instance, when I look for references for a subject, I'm searching for information related to one specific aspect of the subject—such as the giant squid's relevance to mythology—but only for that specific instance in time. The next time I search for information related to the giant squid, I might be searching for information based on a different aspect of giant squids, such as cooking giant squid steaks.

The RDF Triple

Three is a magical number. For instance, three legs are all you need to create a stable stool, and a transmitter and two receivers are all you need to triangulate a specific transmission point. You can create a perfect sphere with infinitely small triangles. (Triangles are a very useful geometric shape, also used to find the heights of mountains and the distances between stars.)

RDF is likewise based on the principle that three is a magic number—in this case, that three pieces of information are all that's needed in order to fully define a single bit of knowledge. Within the RDF specification, an RDF *triple* documents these three pieces of information in a consistent manner that ideally allows both human and machine consumption of the same data. The RDF triple is what allows human understanding and meaning to be interpreted consistently and mechanically.

Of the three pieces of information, the first is the *subject*. A property such as *name* can belong to a dog, cat, book, plant, person, car, nation, or insect. To make finite such an infinite universe, you must set boundaries, and that's what subject does for RDF. The second piece of information is the *property type* or just plain *property*. There are many facts about any individual subject; for instance, I have a gender, a height, a hair color, an eye color, a college degree, relationships, and so on. To define which aspect of me we're interested in, we need to specifically focus on one property.

If you look at the intersection of subject and property, you'll find the final bit of information quietly waiting to be discovered—the *value* associated with the property. X marks the spot. I (subject) have a name (property), which is Shelley Powers (property value). I (subject) have a height (property), which is five feet eleven inches (property value). I (subject) also have a location (property), which is St. Louis (property value). Each of these assertions adds to a picture that is *me*; the more statements defined, the better the picture. Stripping away the linguistic filler, each of these statements can be written as an RDF triple.

With consideration of the differing linguistics based on different languages, simple facts can almost always be defined given three specific pieces of information: the subject of the fact, the property of the subject that is currently being defined, and its associated value. This correlates to what we understand to be a complete thought, regardless of differing syntaxes based on language.

A basic rule of English grammar is that a complete sentence (or statement) contains both a *subject* and a *predicate*: the subject is the *who* or *what* of the sentence and the predicate provides

information about the subject. A sentence about the giant squid article mentioned in the last section could be:

```
The title of the article is "Architeuthis Dux."
```

This is a complete statement about the article. The subject is *the article*, and the predicate is *title*, with a matching value of "*Architeuthis Dux*." Combined, the three separate pieces of information triangulate a specific, completely unique piece of knowledge.

In RDF, this English statement translates to an RDF *triple*. In RDF, the *subject* is the thing being described—in RDF terms, a *resource* identified by a URI (more fully explained in a later section with the same title)—and the *predicate* is a property type of the resource, such as an attribute, a relationship, or a characteristic. In addition to the subject and predicate, the specification also introduces a third component, the *object*. Within RDF, the object is equivalent to the value of the resource property type for the specific subject.

Working with the example sentence earlier, "The title of the article is `*Architeuthis Dux*,'" the generic reference to *article* is replaced by the article's URI, forming a new and more precise sentence:

```
The title of the article at http://burningbird.net/articles/monsters3.htm is
"Architeuthis Dux."
```

With this change, there is no confusion about which article titled "Architeuthis Dux" we're discussing—we're talking about the one with the URI at `http://burningbird.net/articles/monsters3.htm`. Providing a URI is equivalent to giving a person a unique identifier within a personnel system. The individual components of the statement we're interested in can be further highlighted, with each of the three components specifically broken out into the following format:

```
<subject> HAS <predicate> <object>
```

Don't let the angle brackets fool you within this syntax—this isn't XML; this is a representation of a statement whereby three components of the statement can be replaced by instances of the components to generate a specific statement. The example statement is converted to this format as follows:

```
http://burningbird.net/articles/monsters3.htm has a title of
"Architeuthis Dux."
```

In RDF, this new statement, redefined as an RDF triple, can be considered a complete RDF graph because it consists of a complete fact that can be recorded using RDF methodology, and that can then be documented using several different techniques. For instance, one shorthand technique is to use the following to represent a triple:

```
{subject, predicate, object}
```

If you're familiar with set theory, you might recognize this shorthand as a 3-tuple representation. The giant squid example then becomes:

```
{http://burningbird.net/articles/monsters3.htm,  title, "Architeuthis Dux"}
```

This representation of the RDF triple is just one of many ways of serializing RDF data. The formal way is the directed graph, discussed in the next section. Popular choices to serialize the data are N-Triples, a subset of N3 notation (both of which are briefly discussed in this chapter), and RDF/XML, which forms the basis of the remainder of this book.

Regardless of the manner in which an RDF triple is documented, four facts are immutable about each:

- Each RDF triple is made up of subject, predicate, and object.
- Each RDF triple is a complete and unique fact.
- An (RDF) triple is a 3-tuple, which is made up of a subject, predicate and object — which are respectively a uriref or bnode; a uriref; and a uriref, bnode or literal (This is from a comment made by Pat Hayes in http://lists.w3.org/Archives/Public/w3c-rdfcore-wg/2003Feb/0152.html)
- Each RDF triple can be joined with other RDF triples, but it still retains its own unique meaning, regardless of the complexity of the model in which it is included.

That last item is particularly important to realize about RDF triples—regardless of how complex an RDF graph, it still consists of only a grouping of unique, simple RDF triples, and each is made up of a subject, predicate, and object.

Tip

N3 notation does have some major fans as an approach to serializing RDF graphs, including Tim Berners-Lee. However, the W3C has officially sanctioned RDF/XML as the method to use for serializing RDF. One overwhelming advantage of RDF/XML is the wide acceptance of and technical support for XML. Though N3 notation is not covered in detail in this book, you can read a primer on N3 and RDF at http://www.w3.org/2000/10/swap/Primer.

The Basic RDF Data Model and the RDF Graph

The RDF Core Working Group decided on the RDF graph—a directed labeled graph—as the default method for describing RDF data models for two reasons. First, as you'll see in the examples, the graphs are extremely easy to read. There is no confusion about what is a subject and what are the subject's property and this property's value. Additionally, there can be no confusion about the statements being made, even within a complex RDF data model.

The second reason the Working Group settled on RDF graphs as the default description technique is that there are RDF data models that can be represented in RDF graphs, but not in RDF/XML.

Tip

The addition of `rdf:nodeID`s, discussed in Chapter 3, provided some of the necessary syntactic elements that allow RDF/XML to record all RDF graphs. However, RDF/XML still can't encode graphs whose properties (predicates) cannot be recorded as namespace-qualified XML names, or *QNames*. For more on QNames, see *XML in a Nutshell*, Second Edition (O'Reilly).

The RDF directed graph consists of a set of nodes connected by arcs, forming a pattern of *node-arc-node*. Additionally, the nodes come in three varieties: *uriref*, *blank nodes*, and *literals*.

A uriref node consists of a Uniform Resource Identifier (URI) reference that provides a specific identifier unique to the node. There's been discussion that a uriref must point to something that's accessible on the Web (i.e., provide a location of something that when accessed on the Internet returns something). However, there is no formal requirement that urirefs have a direct connectivity with actual web resources. In fact, if RDF is to become a generic means of recording data, it can't restrict urirefs to being "real" data sources.

Blank nodes are nodes that don't have a URI. When identifying a resource is meaningful, or the resource is identified within the specific graph, a URI is given for that resource. However, when identification of the resource doesn't exist within the specific graph at the time the graph was recorded, or it isn't meaningful, the resource is diagrammed as a blank node.

Within a directed graph, resource nodes identified as urirefs are drawn with an ellipse around them, and the URI is shown within the circle; blank nodes are shown as an empty circle. Specific implementations of the graph, such as those generated by the RDF Validator, draw a circle containing a generated identifier, used to distinguish blank nodes from each other within the single instance of the graph.

The literals consist of three parts—a character string and an optional language tag and data type. Literal values represent RDF objects only, never subjects or predicates. RDF literals are drawn with rectangles around them.

The arcs are directional and labeled with the RDF predicates. They are drawn starting from the resource and terminating at the object, with arrows documenting the direction from resource to object (in all instances of RDF graphs I've seen, this is from right to left).

Figure 2-1 shows a directed graph of the example statement discussed in the previous section. In the figure, the subject is contained within the oval to the left, the object literal is within the box, and the predicate is used to label the arrowed line drawn from the subject to the object.

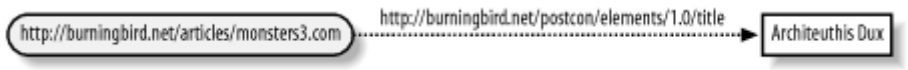

Figure 2-1. RDF directed graph of giant squid article statement

As you can see in the figure, the direction of the arrow is from the subject to the object. In addition, the predicate is given a uriref equal to the schema for the RDF vocabulary elements and the element that serves as predicate itself. Every arc, without exception, must be labeled within the graph.

Blank nodes are valid RDF, but most RDF parsers and building tools generate a unique identifier for each blank node. For example, Figure 2-2 shows an RDF graph generated by the W3C RDF Validator, complete with generated identifier in place of the blank node, in the format of:

```
genid(unique identifier)
```

The identifier shown in the figure is `genid:158`, the number being the next number available for labeling a blank node and having no significance by itself. The use of `genid` isn't required, but the recommended format for blank node identifiers is some form similar to that used by the validator.

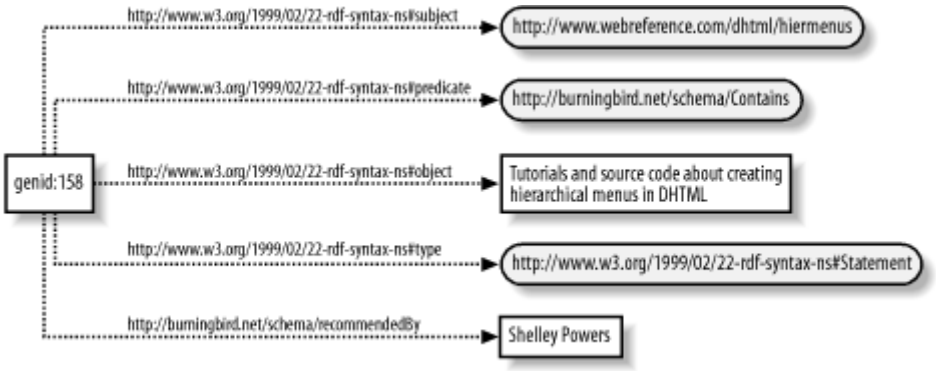

Figure 2-2. Example of autogenerated identifier representing blank node

Blank nodes (sometimes referred to as bnodes or, previously, anonymous nodes) can be problematic within automated processes because the identifier that's generated for each will change from one application run to the next. Because of this, you can't depend on the identifier remaining the same. However, since blank nodes represent placeholder nodes rather than more meaningful nodes, this shouldn't be a problem. Still, you'll want to be aware of the nonpersistent names given to blank nodes by RDF parsers.

Tip

The figures shown in this chapter were transformed from graphics generated by the RDF Validator, an online resource operated by the W3C for validation of RDF syntax (found at http://www.w3.org/RDF/Validator/). This tool will be used extensively throughout this book, and its use is detailed in Chapter 7.

The components of the RDF graph—the uriref, bnode, literal, and arc—are the only components used to document a specific instance of an RDF data model. This small number of components isn't surprising when you consider that, as demonstrated earlier, an RDF triple is a fact comprised of subject-predicate-object. Only when we start recording more complicated assertions and start merging several triples together do the RDF graph and the resulting RDF/XML begin to appear more complex.

URIs

Since an understanding of urirefs is central to working with RDF, we'll take a moment to look at what makes a valid URI—the identifiers contained within a uriref and used to identify specific predicates.

Resources can be accessed with different protocols and using different syntaxes, such as using `http://` to access a resource as a web page and `ftp://` to access another resource using FTP. However, one thing each approach shares is the need to access a specific object given a unique name or identifier. URIs provide a common syntax for naming a resource regardless of the protocol used to access the resource. Best of all, the syntax can be extended to meet new needs and include new protocols.

URIs are related to URLs (Uniform Resource Locators) in that a URL is a specific instance of a URI scheme based on a known protocol, commonly the Hypertext Transfer Protocol (HTTP). URIs, and URLs for that matter, can include either a complete location or path to a resource or a partial or relative path. The URI can optionally include a fragment identifier, separated from the URI by a

pound sign (#). In the following example, `http://burningbird.net/articles/monsters3.htm` is the URI and `introduction` is the fragment:

```
http://burningbird.net/articles/monsters3.htm#introduction
```

A URI is only an identifier. A specific protocol doesn't need to be specified, nor must the object identified physically exist on the Web—you don't have to specify a resolvable protocol such as `http://` or `ftp://`, though you can if you like. Instead, you could use something as different as a UUID (Universally Unique Identifier) referencing a COM or other technology component that exists locally on the same machine or within a network of machines. In fact, a fundamental difference between a URL and a URI is that a URL is a location of an object, while a URI can function as a name or a location. URIs also differ from URNs (Uniform Resource Name) because URIs can refer to a location as well as a name, while URNs refer to globally unique names.

The RDF specification constrains all urirefs to be absolute or partial URIs. An absolute URI would be equivalent to the URL:

```
http://burningbird.net/articles/monsters3.htm
```

A relative URI is just as it sounds—relative to an absolute path. A relative reference to the Monsters article could be:

```
Monsters3.htm
```

If a reference to the base location of the relative URI is not given, it's assumed to be base to the URI of the containing document. The use of URIs and the concepts of namespaces and QNames are discussed in more detail in Chapter 3.

RDF Serialization: N3 and N-Triples

Though RDF/XML is the serialization technique used in the rest of this book, another serialization technique supported by many RDF applications and tools is N-Triples. This format breaks an RDF graph into its separate triples, one on each line. Regardless of the shorthand technique used within RDF/XML, N-Triples generated from the same RDF graph always come out the same, making it an effective way of validating the processing of an RDF/XML document. For instance, the test cases in the RDF Test Cases document, part of the RDF specification, are given in both the RDF/XML format and the N-Triples format to ensure that the RDF/XML (and the underlying RDF concepts) are consistently interpreted.

Tip

Though other techniques for serialization exist, as has been previously discussed, the only serialization technique officially adopted by the RDF specifications is RDF/XML.

N-Triples itself is based on another notation, called N3.

A Brief Look at N3

RDF/XML is the official serialization technique for RDF data, but another notation is also used frequently, which is known as N3 or Notation3. It's important you know how to read it; however, since this book is focusing on RDF/XML, we'll look only briefly at N3 notation.

Tip

N3 exists independent of RDF and can extend RDF in such a way as to violate the semantics of the underlying RDF graph. Some prefer N3 to RDF/XML; I am not one of them, primarily because I believe RDF/XML is a more comfortable format for people more used to markup (such as XML or HTML).

The basic structure of an N3 triple is:

```
subject
predicate
object .
```

In this syntax, the subject, predicate, and object are separated by spaces, and the triple is terminated with a period (`.`). An actual example of N3 would be:

```
<http://weblog.burningbird.net/fires/000805.htm>
                              <http://purl.org/dc/elements/1.1/creator> Shelley
.
```

In this example, the absolute URIs are surrounded by angle brackets. To simplify this even further, namespace-qualified XML names (QNames) can be used instead of the full namespace, as long as the namespaces are declared somewhere within the document. If QNames are used, the angle brackets are omitted for the predicates:

```
<bbd:000805.htm> dc:creator Shelley.
```

To represent multiple triples, with related resources, just list out the triples. Converting the RDF/XML in Example 3-9 into N3 we have:

```
<bbd:monsters1.htm> pstcn:bio <#monster1> .
<#monster1> pstcn.title "Tale of Two Monsters: Legends .
<#monster1> pstcn.description Part 1 of four-part series on cryptozoology,
legends,
Nessie the Loch Ness Monster and the giant squid" .
<#monster1> pstcn:creator "Shelley" Powers .
<#monster1> pstcn:created "1999-08-01T00:00:00-06:00" .
```

To represent bnodes or blank nodes, use whatever designation you would prefer to identify the bnode identifier. An example from the RDF Primer is:

```
exstaff:85740  exterms:address  _:johnaddress .
_:johnaddress  exterms:street   "1501 Grant Avenue" .
_:johnaddress  exterms:city     "Bedford" .
_:johnaddress  exterms:state    "Massachusetts" .
_:johnaddress  exterms:Zip      "01730" .
```

Though brief, these notes should enable you to read N3 notation all through the RDF specification documents. However, since the focus of this book is RDF/XML, N3 notation won't be used again.

N-Triples

Since N-Triples is a subset of N3, it supports the same format for RDF triples:

```
subject
predicate
object .
```

According to the Extended Backus-Naur Form (EBNF) for N-Triples, a space or a tab separates the three elements from each other, and a space or a tab can precede the elements. In addition, each triple is ended with a period (`.`) followed by a line-feed or carriage-return/line-feed. An N-Triples file can also contain comments in the following format:

```
# comment
```

Each line in an N-Triples file consists of either a triple or a comment, but not both.

As for the triple elements themselves, the subject can consist of either a uriref or a blank node identifier The latter is a value generated for blank nodes within N-Triples syntax, as there can be no blank subjects within legal N-Triples-formatted output. The blank node identifier format is:

```
_:name
```

where *`name`* is a string. The predicate is always a uriref, and the object can be a uriref, a blank node, or a literal.

Given an RDF graph as shown in Figure 2-3, N-Triples would be returned representing both the title triple and the author triple. Adding in a comment, the output in Example 2-1 is valid N-Triples output for the same RDF graph.

Figure 2-3. RDF graph with two RDF triples and one subject

Example 2-1. N-Triples output

```
# Chapter 2 Example1
<http://burningbird.net/articles/monsters3.htm> .
<http://burningbird.net/postcon/
elements/1.0/author> "Shelley Powers" .
<http://www.burningbird.net/articles/monsters3.htm>
<http://burningbird.net/postcon/
elements/1.0/title> "Architeuthis Dux" .
```

Note that angle brackets are used in N-Triples notation only when the object enclosed is a complete, absolute URI. QNames are not enclosed in angle brackets.

A slightly more complex example of N-Triples can be seen in Example 2-2. In this example, four triples are given for one subject, which in this case happens to be a blank node. Since nodes without labels are not allowed in N-Triples format, the RDF parser (NTriple, included with the ARP parser discussed in Chapter 7) generated an identifier to represent the subject in each triple.

Example 2-2. N-Triples output with generated blank node identifier

```
_:j0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject>
<http://www.webreference.com/
```

```
dhtml/hiermenus> .

_:j0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate>
<http://burningbird.net/
schema/Contains> .

_:j0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#object>
"Tutorials and source code about creating hierarchical menus in DHTML" .

_:j0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement> .

_:j0 <http://burningbird.net/schema/recommendedBy> "Shelley Powers" .
```

Talking RDF: Lingo and Vocabulary

Right at this moment, you have enough understanding of the RDF graph to progress into the RDF/XML syntax in the next chapter. However, if you follow any of the conversations related to RDF, some terms and concepts might cause confusion. Before ending this chapter on the RDF graph, I thought I would spend some time on these potentially confusing concepts.

Graphs and Subgraphs

In any RDF graph, a *subgraph* of the graph would be a subset of the triples contained in the graph. As I said earlier, each triple is uniquely its own RDF graph, in its own right, and can actually be modeled within a separate directed graph. In Figure 2-3, the triple represented by the following is a subgraph of the entire set of N-Triples representing the entire graph:

```
<http://burningbird.net/articles/monsters3.htm> <http://burningbird.net/postcon/
elements/1.0/title> "Architeuthis Dux"
```

Taking this concept further, a union of two or more RDF graphs is a new graph, which the Model document calls a *merge* of the graphs. For instance, Figure 2-4 shows one graph containing exactly one RDF triple (one statement).

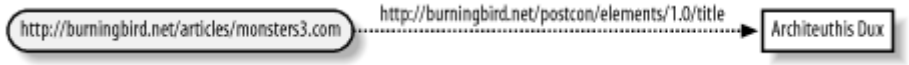

Figure 2-4. RDF graph with exactly one triple

Adding the following triple results in a new merged graph, as shown previously in Figure 2-3. Since both triples share the same subject, as determined by the URI, the mergence of the two attaches the two different triples to the same subject:

```
<http://burningbird.net/articles/monsters3.htm> <http://burningbird.net/postcon/
elements/1.0/author> "Shelley Powers"
```

Now, if the subjects differed, the merged graph would still be valid—there is no rule or regulation within the RDF graph that insists that all nodes be somehow connected with one another. All the RDF graph insists on is that the triples are valid and that the RDF used with each is valid. Figure 2-5 shows an RDF graph of two merged graphs that have disconnected nodes.

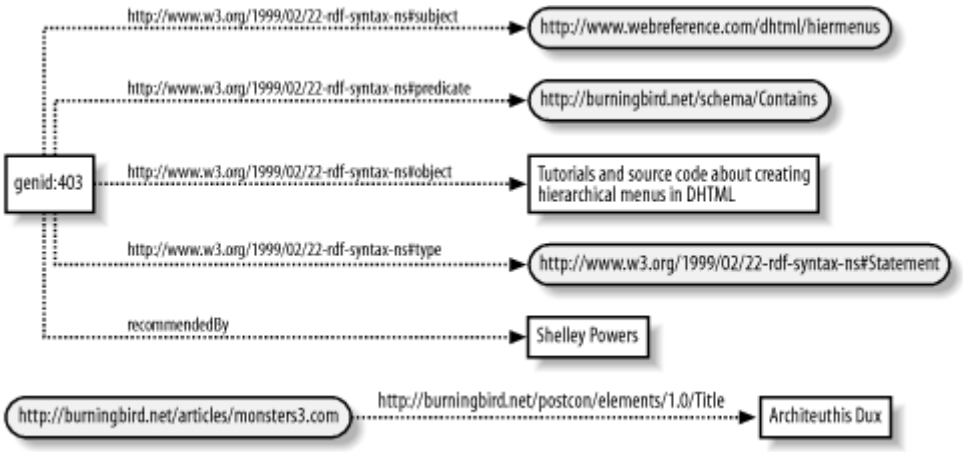

Figure 2-5. Merged RDF graph with disconnected nodes

Blank nodes are never merged in a graph because there is no way of determining whether two nodes are the same—one can't assume similarity because of artificially generated identifiers. The only components that are merged are urirefs and literals (because two literals that are syntactically the same can be assumed to be the same). In fact, when tools are given two graphs to merge and each graph contains blank nodes, each blank node is given a unique identifier in order to separate it from the others before the mergence.

Ground and Not Graph

An RDF graph is considered *grounded* if there are no blank nodes. Figure 2-4 is an example of a grounded RDF graph, while Figure 2-5 is not because of the blank node (labeled *genid:403*). Additionally, an *instance* of an RDF graph is a graph in which each blank node has been replaced by an identifier, becoming a named node. In Figure 2-5, a named node replaced the blank node; if I were to run the RDF Validator against the RDF/XML that generated this example I would get a second instance, and the names used for the blank nodes would differ. Semantically the two graphs would represent the same RDF graph but are considered separate instances of the graph.

Finally, an RDF *vocabulary* is the collection of all urirefs from a specific RDF graph. Much discussion is made of the Dublin Core vocabulary or the RSS vocabulary and so on (discussed more in Chapter 6). However, a true RDF vocabulary can differ from an official implementation of it by the very fact that the urirefs may differ between the two.

Since this is a bit confusing, for the rest of the book when I refer to an RDF vocabulary, I'm referring to a schema of a particular vocabulary, rather than any one particular implementation or document derived from it.

Entailment

Within the RDF Semantics document, *entailment* describes two graphs, which are equal in all aspects. By this I mean that every assertion made about one RDF graph can be made with equal truth about the other graph. For instance, statements made in one graph are implicitly made in the other; if you believe the statement in the first, you must, through entailment, believe the same statement in the other.

As examples of entailment, the formal term *subgraph lemma* states that a graph entails all of its subgraphs, because whatever assertions can be made about the whole graph can also be made

against the subgraphs, aside from differences associated with the subgraphing process (e.g., the original graph had two statements, while the subgraph had only one). Another lemma, *instance lemma*, states that all instances of a graph are entailed by the graph—*instance* in this case an implementation of a graph in which all blank nodes have been replaced by a literal or a uriref.

Earlier I talked about merging graphs. The *merging lemma* states that the merged graph entails all the graphs that form its final construction. Another lemma, *monotonicity lemma*, states that if a subgraph of a graph entails another graph, then the original graph also entails that second graph.

Tip

Within web specifications, one hopes not to run into terms such as *lemma*, which means "subsidiary proposition assumed to be valid and used to demonstrate a principal proposition," according to the dictionary. However, I know that the main purpose of the Semantics document within the RDF specification is to provide fairly concrete interpretations of the RDF graph theory so that implementers of the technology can provide consistent implementations. For those who primarily use RDF/XML technology rather than create parsers or RDF databases, an understanding of the pure RDF semantics isn't essential—but it is helpful, which is why I'm covering it, however lightly.

The *interpolation lemma* actually goes more into the true nature of entailment than the others, and so I'll cover it in more detail.

The interpolation lemma states:

```
S entails a graph E if and only if a subgraph of the merge of S is an instance
of E.
```

This lemma basically states that you can tell whether one set of graphs entails another if you take a subgraph of the mergence of the graphs, replace the named nodes with blank nodes, and, if the result is an instance of the second set of graphs, the first set is said to entail them. From an editor's draft:

> "To tell whether a set of RDF graphs entails another, check that there is some instance of the entailed graph which is a subset of the merge of the original set of graphs."

Oversimplification aside, what's important to realize about entailment is that it's not the same thing as equality. Equality is basically two graphs that are identical, even down to the same named nodes. Entailment implies something a little more sophisticated—that the semantics of an RDF construct as shown in a specific implementation of a graph map to that which is defined within the formal semantics of the model theoretic viewpoint of the abstract RDF graph. The information in the entailed graph is the same as the information in the other but may have a different physical representation. It is entailment that allows us to construct a graph using a node-edge-node pattern and know that this instance of the RDF graph is a valid one, and that whatever semantic constraints exist within the model theoretic viewpoint of RDF also exist within this real-world instance of RDF. Additionally, entailment allows different manipulations of the data in the graphs, as long as the original information is preserved