Solving Problems with the Resource Description Framework

Practical RDF

O'REILLY®

Shelley Powers

Practical RDF

Shelley Powers

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Practical RDF
by Shelley Powers

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Simon St.Laurent

Production Editor: Mary Brady

Cover Designer: Ellie Volckhausen

Interior Designer: David Futato

Printing History:

July 2003: First Edition.

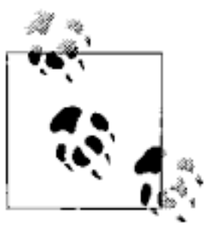
ISBN: 0-596-00263-7
[C] [11/03]

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.



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.

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.

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.

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.

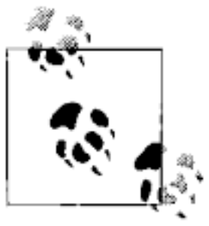
When one considers that much of the cost of a manufactured item resides in the management of the supply chain and within the manufacturing process, not in the raw material used to manufacture the item, I would expect to see considerable progress from industry efforts such as POSC and PDES in RDF/XML.

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.



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.

Now, time to get started.

CHAPTER 2

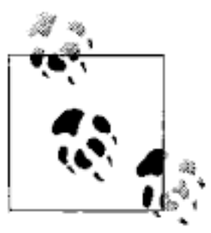
RDF: Heart and Soul

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.



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.

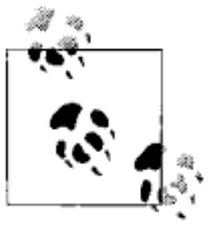
- A related article is ...
- The article is about the giant squid and its place in the legends.

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.



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:

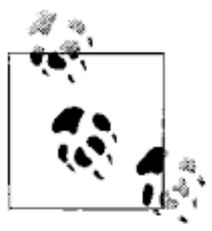
```
{http://burningbird.net/articles/monsters3.htm,  title, "Architeuthis Dux"}
```

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.

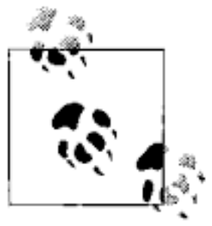


N3 notation does have some major fans as an approach to serializing RDF graphs, including Tim Berners-Lee. However, the W3C has officially sanctioned RDF/XML as the method to use for serializing RDF. One overwhelming advantage of RDF/XML is the wide acceptance of and technical support for XML. Though 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.



The addition of `rdf:nodeIDs`, 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.

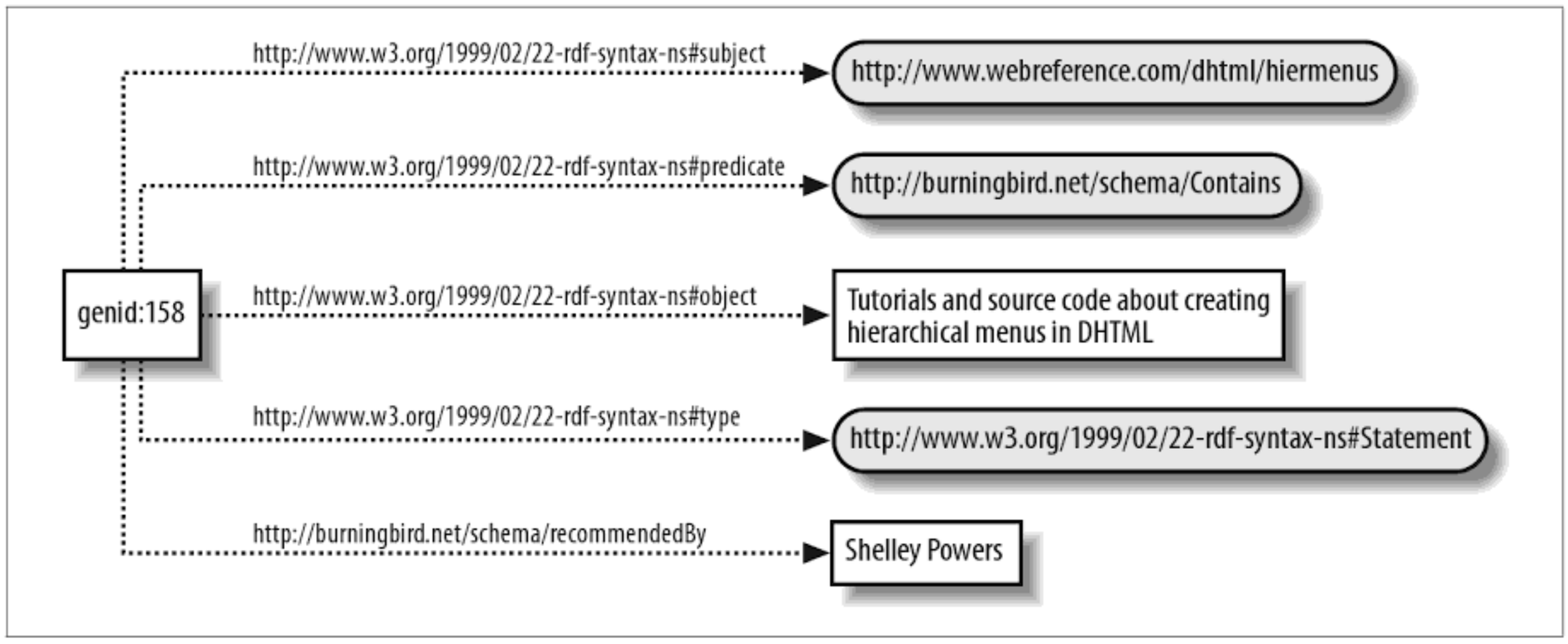
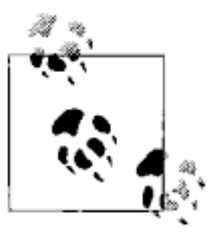


Figure 2-2. Example of autogenerated identifier representing blank node

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.



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
```

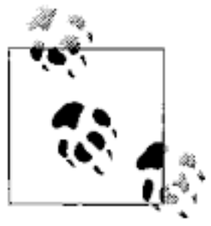
A relative URI is just as it sounds—relative to an absolute path. A relative reference to the Monsters article could be:

```
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.

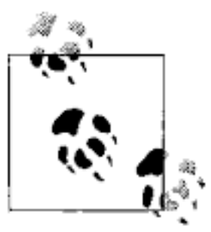


Though other techniques for serialization exist, as has been previously discussed, the only serialization technique officially adopted by the RDF specifications is RDF/XML.

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.



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.

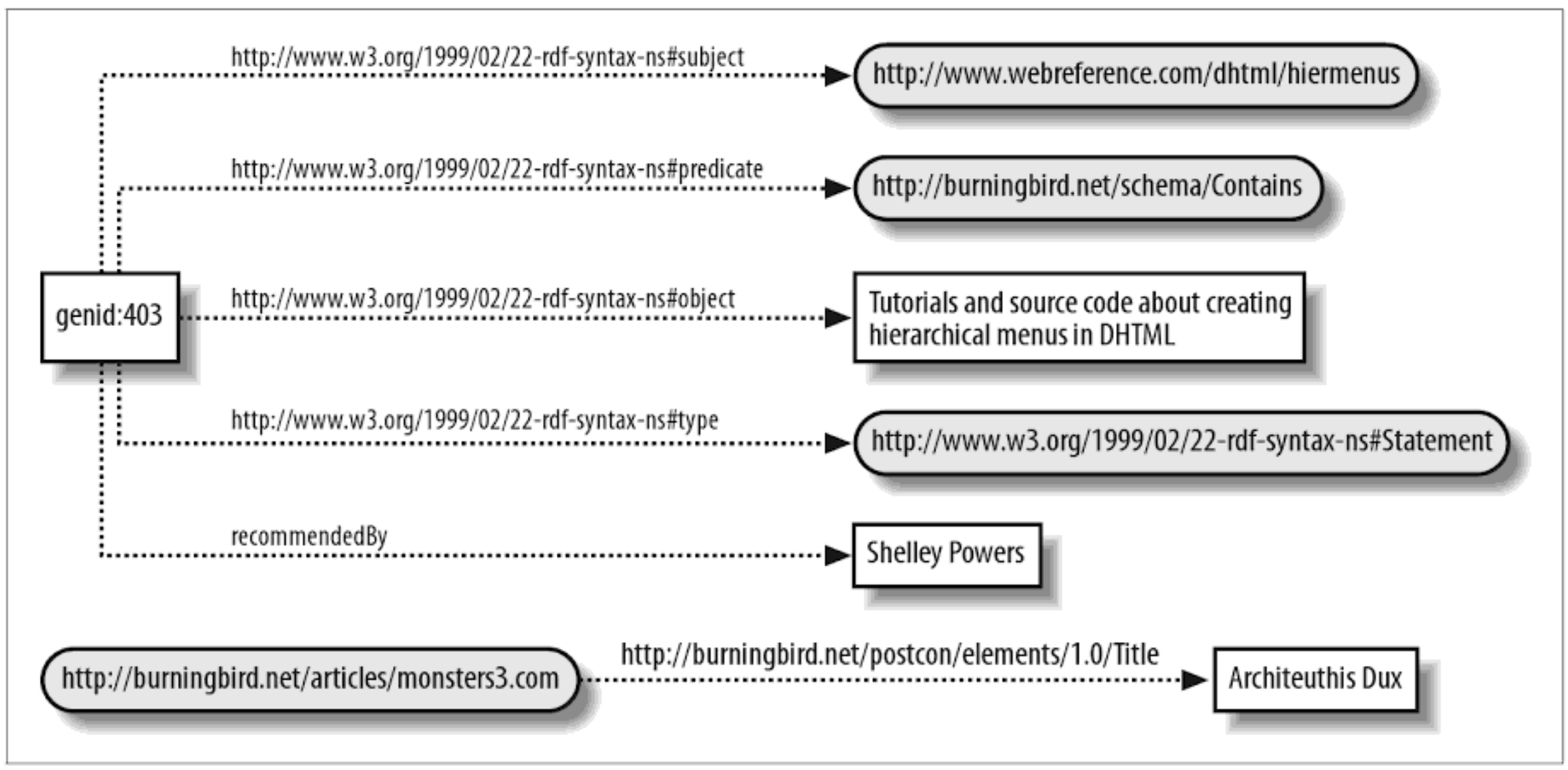


Figure 2-5. Merged RDF graph with disconnected nodes

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.

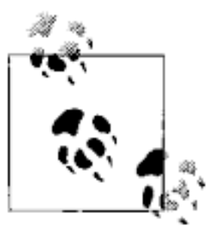
Since this is a bit confusing, for the rest of the book when I refer to an RDF vocabulary, I'm referring to a schema of a particular vocabulary, rather than any one particular implementation or document derived from it.

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.



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."

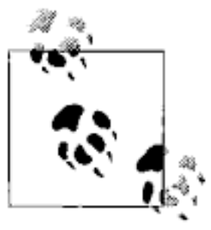
Oversimplification aside, what's important to realize about entailment is that it's not the same thing as equality. Equality is basically two graphs that are identical, even down to the same named nodes. Entailment implies something a little more sophisticated—that the semantics of an RDF construct as 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.

CHAPTER 3

The Basic Elements Within the RDF/XML Syntax

The usability of RDF is heavily dependent on the portability of the data defined in the RDF models and its ability to be interchanged with other data. Unfortunately, recording the RDF data in a graph—the default RDF documentation format—is not the most efficient means of storing or retrieving this data. Instead, transporting RDF data, a process known as *serialization*, usually occurs with RDF/XML.

Originally, the RDF model and the RDF/XML syntax were incorporated into one document, the Resource Description Framework (RDF) Model and Syntax Specification. However, when the document was updated, the RDF model was separated from the document detailing the RDF/XML syntax. Chapter 2 covered the RDF abstract model, graph, and semantics; this chapter provides a general introduction to the RDF/XML model and syntax (RDF M&S).



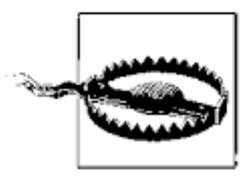
The original RDF M&S Specification can be found at *http://www.w3.org/TR/REC-rdf-syntax/*. The updated RDF/XML Syntax Specification (revised) can be found at *http://www.w3.org/TR/rdf-syntax-grammar/*.

Some RDF-specific aspects of RDF/XML at first make it seem overly complex when compared to non-RDF XML. However, keep in mind that RDF/XML is nothing more than well-formed XML, with an overlay of additional constraints that allow for easier interchange, collection, and mergence of data from multiple models. In most implementations, RDF/XML is parsable with straight XML technology and can be manipulated manually if you so choose. It's only when the interchangeability of the data is important and the data can be represented only by more complex data structures and relationships that the more formalized elements of RDF become necessary. And in those circumstances, you'll be glad that you have the extra capability.

All examples listed in the chapter are validated using the W3C's RDF Validator, located at *http://www.w3.org/RDF/Validator/*.

Serializing RDF to XML

Serialization converts an object into a persistent form. The RDF/XML syntax provides a means of documenting an RDF model in a text-based format, literally *serializing* the model using XML. This means that the content must both meet all requirements for well-formed XML and the additional constraints of RDF. However, before showing you some of these constraints, let's walk through an example of using RDF/XML.



RDF doesn't require XML-style validity, just well-formedness. RDF/XML parsers and validators do not use DTDs or XML Schemas to ensure that the XML used is valid. Norman Walsh wrote a short article for *xml.com* on what it means for an XML document to be well formed and/or valid; it explains the two concepts in more detail. See it at *http://www.xml.com/pub/a/98/10/guide3.html*.

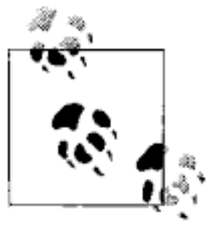
In Chapter 2, I discussed an article I wrote on the giant squid. Now, consider attaching context to it. Among the information that could be exposed about the article is that it explores the idea of the giant squid as a legendary creature from myths and lore; it discusses the current search efforts for the giant squid; and it provides physical characteristics of the creature. Putting this information into a paragraph results in the following:

```
The article on giant squids, titled "Architeuthis Dux," at
http://burningbird.net/articles/monsters3.htm, written by Shelley Powers, explores
the giant's squid's mythological representation as the legendary Kraken as well
as describing current efforts to capture images of a live specimen. In addition,
the article also provides descriptions of a giant squid's physical
characteristics. It is part of a four-part series, described at
http://burningbird.net/articles/monsters.htm and entitled "A Tale of Two
Monsters."
```

Reinterpreting this information into a set of statements, each with a specific predicate (property or fact) and its associated value, I come up with the following list:

- The article is uniquely identified by its URI, `http://burningbird.net/articles/monsters3.htm`.
- The article was written by Shelley Powers—predicate is *written by*, value is *Shelley Powers*.
- The article's title is "Architeuthis Dux"—predicate is *title*, value is *Architeuthis Dux*.
- The article is one of a four-part series—predicate is *series member*, value is *http://burningbird.net/articles/monsters.htm*.
- The series is titled "A Tale of Two Monsters"—series predicate is *title*, value is *A Tale of Two Monsters*.

- The article associates the giant squid with the legendary Kraken—predicate is *associates*, value is *Kraken and giant squid*.
- The article provides physical descriptions of the giant squid—predicate is *provides*, value is *physical description of giant squid*.



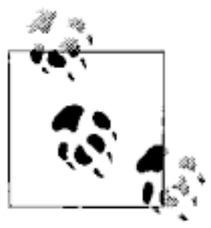
You'll notice in this chapter and elsewhere in the book that I tend to use *RDF statement* and *RDF triple* seemingly interchangeably. However, I primarily use *RDF statement* when referring to the particular fact being asserted by an RDF triple and use *RDF triple* when referring to the actual, physical instantiation of the statement in RDF triple format.

Starting small, we'll take a look at mapping the article and the author and title, only, into RDF. Example 3-1 shows this RDF mapping, wrapped completely within an XML document.

Example 3-1. Preliminary RDF of giant squid article

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
  </rdf:Description>
</rdf:RDF>
```

Tracing the XML from the top, the first line is the traditional XML declaration line. Following it is the RDF element, `rdf:RDF`, used to enclose the RDF-based content.



If the fact that the content is RDF can be determined from the context of the XML, the containing RDF element isn't necessary and can be omitted. In addition, the RDF content can be embedded within another document, such as an XML or HTML document, as will be discussed later in the section titled "RDF/XML: Separate Documents or Embedded Blocks."

Contained as attributes within the RDF element is a listing of the namespaces that identify the vocabulary for each RDF element. The first, with an `rdf` prefix, is the namespace for the RDF syntax; the second, with a prefix of `pstcn`, identifies elements I've created for the example RDF in this book. The namespace references an existing schema definition (see more on RDF Schemas in Chapter 5), but the schema itself doesn't have to exist on the Web, because it's not used for validation. However, as you will see in Chapter 5, there is good reason to physically create the RDF Schema document in the location given in the namespace URI.

In the example, after the enclosing `rdf:RDF` element is the RDF *Description*. An RDF Description begins with the opening RDF Description tag, `rdf:Description`, which in this case includes an attribute (`rdf:about`) used to identify the resource (the subject). The resource used within the specific element could be an identifier to a resource defined elsewhere in the document or the URI for the subject itself. In the example, the resource identifier is the URI for the giant squid article page.

The RDF Description wraps one or more resource predicate/object pairs. The predicate objects (the values) can be either literals or references to another resource. Regardless of object type, each RDF statement is a complete triple consisting of subject-predicate-object. Figure 3-1 shows the relationship between the RDF syntax and the RDF trio from the example.

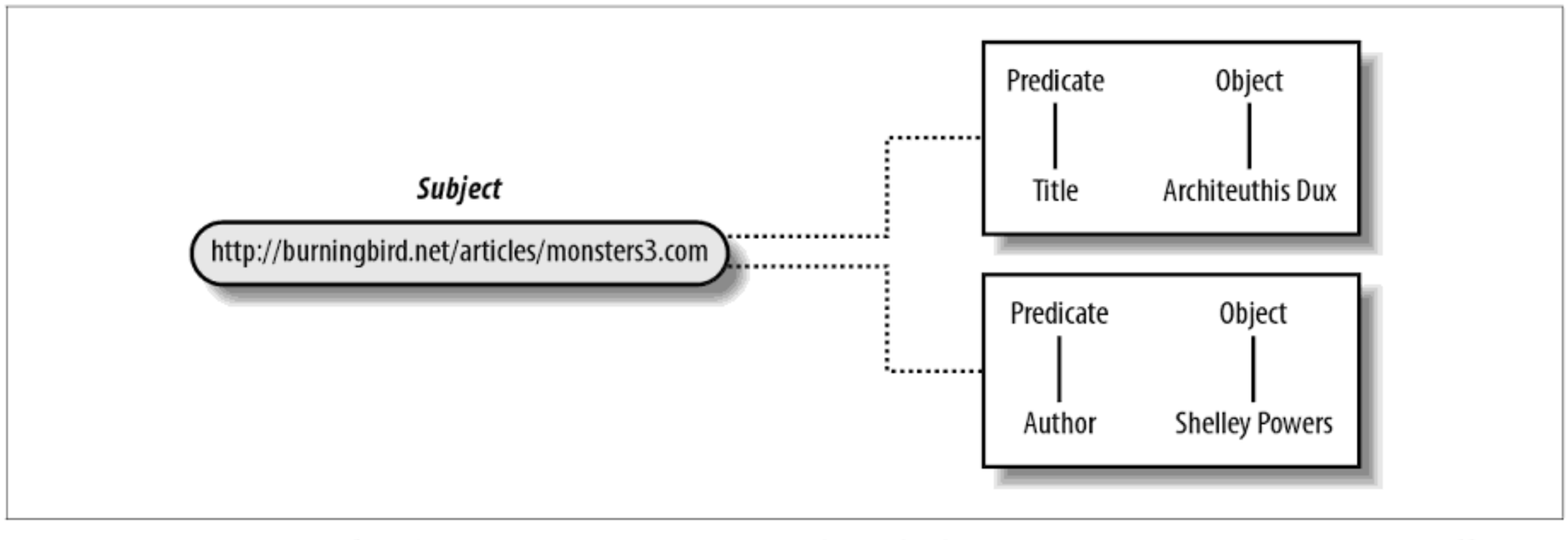


Figure 3-1. An example of two RDF statements, each with the same subject (resource), as well as a mapping between statement elements and values

As you can see, a complete RDF statement consists of the resource, a predicate, and its value. In addition, as the figure shows, resources can be described by more than one property (in RDF parlance, the subject can participate in more than one RDF statement within the document).

Running Example 3-1 through the RDF Validator results in a listing of N-Triples in the form of subject, predicate, and object:

```
<http://dynamicearth.com/articles/monsters3.htm>
              <http://burningbird.net/postcon/elements/1.0/author> "Shelley Powers"
.
<http://dynamicearth.com/articles/monsters3.htm>
              <http://burningbird.net/postcon/elements/1.0/title> "Architeuthis Dux"
.
```

The N-Triples representation of each RDF statement shows the formal identification of each predicate, as it would be identified within the namespace schema.

The validator also provides a graphic representation of the statement as shown in Figure 3-2. As you can see, the representation matches that shown in Figure 3-1—offering validation that the model syntax used does provide a correct representation of the statements being modeled.

Figure 3-2. RDF Validator–generated directed graph of Example 3-1

In Example 3-1, the objects are literal values. However, there is another resource described in the original paragraph in addition to the article itself: the series the article is a part of, represented with the URI `http://burningbird.net/articles/monsters.htm`. The series then becomes a new resource in the model but is still referenced as a property within the original article description.

To demonstrate this, in Example 3-2 the RDF has been expanded to include the information about the series, as well as to include the additional article predicate/object pairs. The modifications to the original RDF/XML are boldfaced.

Example 3-2. Expanded version of the giant squid RDF

```
<?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/">

<rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
    <pstcn:series rdf:resource="http://burningbird.net/articles/monsters.htm" />
    <pstcn:contains>Physical description of giant squids</pstcn:contains>
    <pstcn:alsoContains>Tale of the Legendary Kraken</pstcn:alsoContains>
</rdf:Description>

<rdf:Description rdf:about="http://burningbird.net/articles/monsters.htm">
    <pstcn:seriesTitle>A Tale of Two Monsters</pstcn:seriesTitle>
</rdf:Description>
</rdf:RDF>
```

The `rdf:resource` attribute within the `pstcn:series` predicate references a resource object, in this case one that's defined later in the document and which has a predicate of its own, `pstcn:seriesTitle`. Though the statements for the linked resource are separate from the enclosed statements in the original resource within the RDF/XML, the RDF graph that's generated in Figure 3-3 shows the linkage between the two.

The linked resource could be nested directly within the original resource by enclosing it within the original resource's `rdf:Description` element, in effect nesting it within the original resource description. Example 3-3 shows the syntax for the example after this modification has been applied. As you can see with this XML, the second resource being referenced within the original is more apparent using this approach, though the two result in equivalent RDF models.

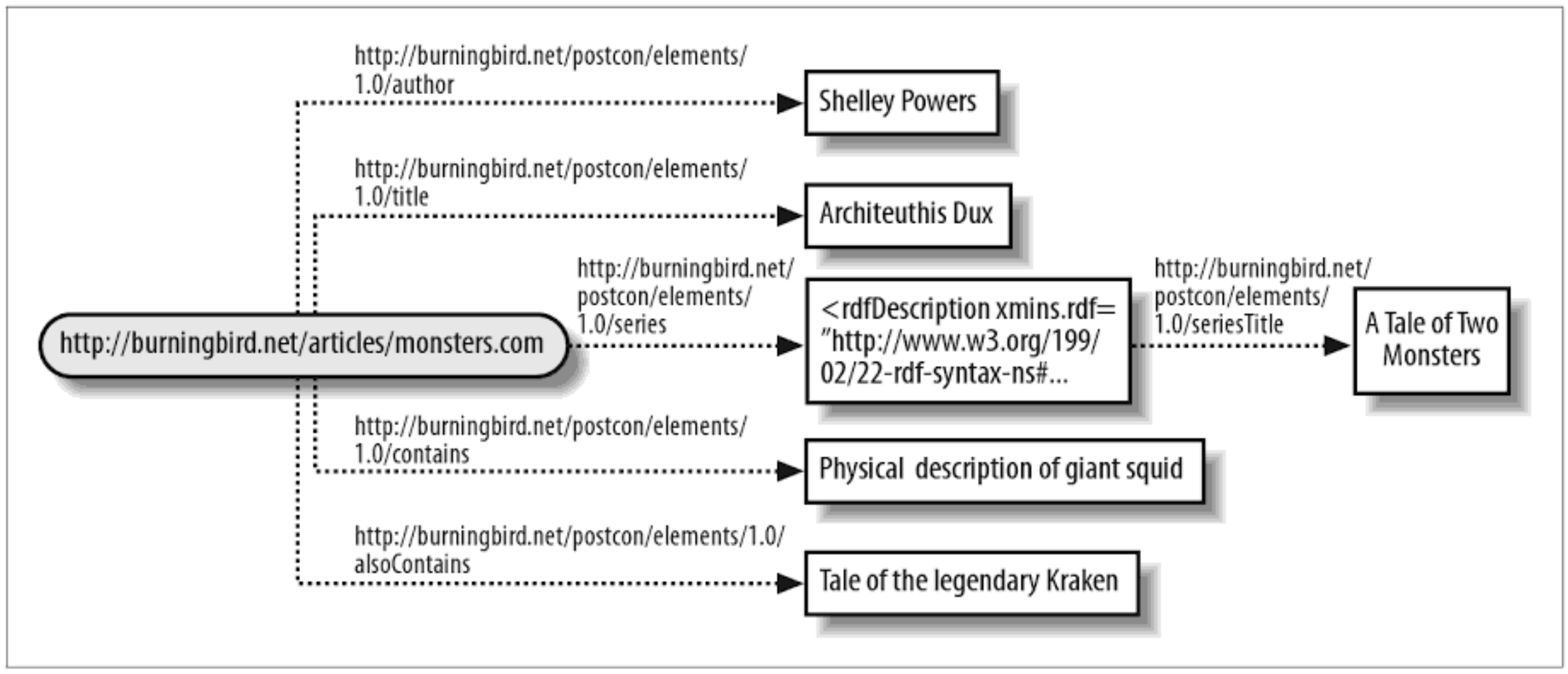


Figure 3-3. Using rdf:resource to set an object to another resource

Example 3-3. Expanded RDF modified to use nested resources

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
    <pstcn:series>
      <rdf:Description rdf:about=
                    "http://burningbird.net/articles/monsters.htm">
        <pstcn:SeriesTitle>A Tale of Two Monsters</pstcn:SeriesTitle>
      </rdf:Description>
    </pstcn:series>
    <pstcn:contains>Physical description of giant squids</pstcn:contains>
    <pstcn:alsoContains>Tale of the Legendary Kraken</pstcn:alsoContains>
  </rdf:Description>
</rdf:RDF>
```

Though nesting one resource description in another shows the connection between the two more clearly, I prefer keeping them apart—it allows for cleaner RDF documents in my opinion. If nesting becomes fairly extreme—a resource is an object of another resource, which is an object of another resource, and so on—trying to represent all of the resources in a nested manner soon becomes unreadable (though automated processes have no problems with it).

Example 3-3 demonstrates a fundamental behavior with RDF/XML: subjects and predicates occur in layers, with subjects separated from other subjects by predicates and predicates separated from other predicates by subjects. Subjects are never nested directly within subjects, and predicates are never nested directly within predicates. This RDF/XML *striping* is discussed next.

Striped Syntax

In a document titled "RDF: Understanding the Striped RDF/XML Syntax" (found at *http://www.w3.org/2001/10/stripes/*), the author, Dan Brickley, talks about a specific pattern of node-edge-node that forms a striping pattern within RDF/XML. This concept has been included in the newer Syntax document as a method of making RDF/XML a little easier to read and understand.

If you look at Figure 3-3, you can see this in the thread that extends from the subject (`http://burningbird.net/articles/monsters3.htm`) to the predicate (`pstcn:series`) to the object, which is also a resource (`http://burningbird.net/articles/monsters.htm`) to another predicate (`pstcn:seriesTitle`) to another object, a literal in this case (`A Tale of Two Monsters`). In this thread, no two predicates are nested directly within each other. Additionally, all nodes (subject or object) are separated by an arc—a predicate—providing a node-arc-node-arc-node... pattern.

Within RDF/XML this becomes particularly apparent when you highlight the predicates and their associated objects within the XML. Example 3-3 is replicated in Example 3-4, except this time the predicate/objects are boldfaced to make them stand out.

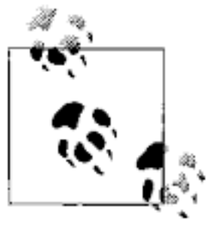
Example 3-4. Expanded RDF modified to use nested resources, predicates bolded to make them stand out

```
<?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/">

  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
    <pstcn:series>
      <rdf:Description rdf:about=
                        "http://dynamicearth.com/articles/monsters.htm">
                  <pstcn:seriesTitle>A Tale of Two Monsters</pstcn:seriesTitle>
      </rdf:Description>
    </pstcn:series>
    <pstcn:contains>Physical description of giant squids</pstcn:contains>
    <pstcn:alsoContains>Tale of the Legendary Kraken</pstcn:alsoContains>
  </rdf:Description>

</rdf:RDF>
```

Viewed in this manner, you can see the striping effect, whereby each predicate is separated by a resource, each resource by a predicate. This maps to the node-arc-node pattern established in the abstract RDF model based on directed graphs. This visualization clue can help you read RDF/XML more easily and allow you to differentiate between predicates and resources.



Another convention, though it isn't a requirement within the RDF specifications, is that all predicates (properties) start with lowercase (such as `title`, `author`, and `alsoContains`), and all classes start with an uppercase. However, in the examples just shown, other than the classes defined within the RDF Schema (such as `Description`), there is no implementation-specific class. Most of the XML elements present are RDF/XML properties. Later we'll see how to formally specify the PostCon classes within the RDF/XML.

Predicates

As you've seen in the examples, a predicate value (object) can be either a resource or a literal. If the object is a resource, an oval is drawn around it; otherwise, a rectangle is drawn. RDF parsers (and the RDF Validator) know which is which by the context of the object itself. However, there is a way that you can specifically mark the type of property—using the `rdf:parseType` attribute.

By default, all literals are plain literals and can be strings, integers, and so on. Their format would be the string value plus an optional `xml:language`. However, you can also embed XML within an RDF document by using the `rdf:parseType` attribute set to a value of `"Literal"`. For instance, Example 3-5 shows the RDF/XML from Example 3-4, but in this case the `pstcn:alsoContains` predicate has an XML-formatted value.

Example 3-5. RDF/XML demonstrating use of rdf:parseType

```
<?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/">

  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
    <pstcn:series>
      <rdf:Description rdf:about=
                        "http://dynamicearth.com/articles/monsters.htm">
        <pstcn:seriesTitle>A Tale of Two Monsters</pstcn:seriesTitle>
      </rdf:Description>
    </pstcn:series>
    <pstcn:contains>Physical description of giant squids</pstcn:contains>
    <pstcn:alsoContains rdf:parseType="Literal">
                                <h1>Tale of the Legendary Kraken
                                </h1></pstcn:alsoContains>
  </rdf:Description>

</rdf:RDF>
```

Without the rdf:parseType="Literal" attribute, the RDF/XML wouldn't be valid. Running the text through the RDF Validator results in the following error:

```
Error: {E202} Expected whitespace found: &apos;Tale of the Legendary Kraken&apos;.
[Line = 17, Column = 69
```

Specifically, rdf:parseType="Literal" is a way of embedding XML directly into an RDF/XML document. When used, RDF processors won't try to parse the element for additional RDF/XML when it sees the XML tags. If you used rdf:parseType="Literal" with series, itself, the RDF parser would place the literal value of the rdf:Description block within a rectangle, rather than parse it out. You'd get a model similar to that shown in Figure 3-4

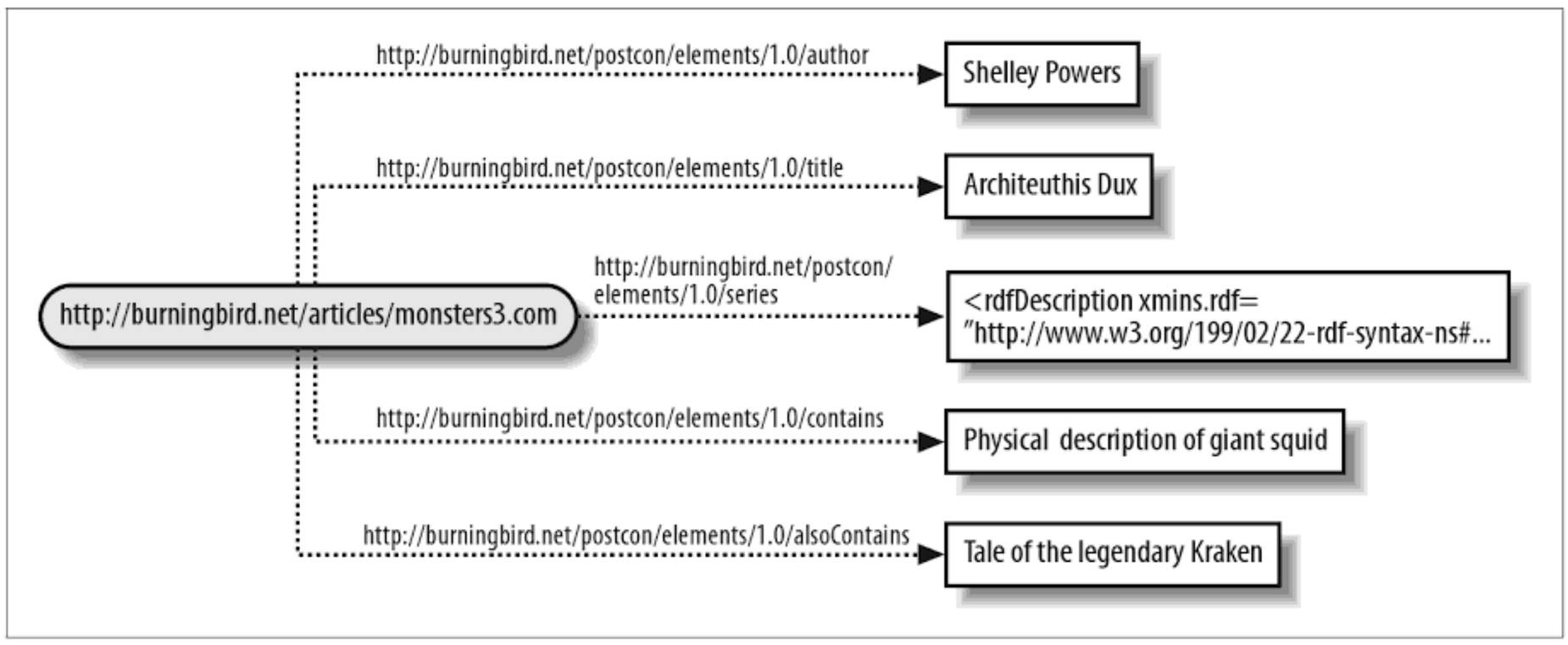


Figure 3-4. Using rdf:parseType of "Literal" for a property surrounding an RDF:Description block

Another rdf:parseType option, "Resource", identifies the element as a resource without having to use rdf:about or rdf:ID. In other words, the surrounding rdf:Description tags would not be necessary:

```
<rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
   <pstcn:series rdf:parseType="Resource">
      <pstcn:seriesTitle>A Tale of Two Monsters</pstcn:seriesTitle>
   </pstcn:series>
...
</rdf:Description>
```

The RDF/XML validates, and the RDF Validator creates an oval for the property. However, it would add a generated identifier in the oval, because the resource is a blank node. There is no place to add a URI for the object in the bubble, because there is no resource identifier for the series property. You can list the seriesTitle directly within the series property, and the property would be attached to it in the RDF graph. But there would be no way to attach a URI to the resource—it would remain as a blank node.

The rdf:parseType property can be used to mark a property as "Resource", even if there is no property value given yet. For instance, in Example 3-6, the property is marked as "Resource", but no value is given.

Example 3-6. RDF/XML demonstrating use of rdf:parseType

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author rdf:parseType="Resource" />
  </rdf:Description>

</rdf:RDF>
```

This approach can be used to signify that the object value isn't known but is nonetheless a valid property. Within the RDF directed graph resulting from this RDF/XML, an oval with a generated identifier is drawn to represent the object, as shown in Figure 3-5.

Figure 3-5. RDF directed graph of model containing "Resource" object with no value provided

Namespaces and QNames

An important goal of RDF is to record knowledge in machine-understandable format and then provide mechanisms to facilitate the combination of the data. By allowing combinations of multiple models, additions can be incorporated without necessarily impacting an existing RDF Schema. To ensure that RDF/XML data from different documents and different specifications can be successfully merged, namespace support has been added to the specification to prevent element collision. (Element collision occurs when an element with the same name is identified in two different schemas used within the same document.)

Read more on XML namespaces in the document "Namespaces in XML" at *http://www.w3.org/TR/1999/REC-xml-names-19990114/*. You may also want to explore the commentary provided in "XML Namespace Myths Exploded," available at *http://www.xml.com/pub/a/2000/03/08/namespaces/index.html*.

To add namespace support to an RDF/XML document, a namespace attribute can be added anywhere in the document; it is usually added to the RDF tag itself, if one is used. An example of this would be:

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

In this XML, two namespaces are declared—the RDF/XML syntax namespace (a requirement) and the namespace for the PostCon vocabulary. The format of namespace declarations in RDF/XML usually uses the following format:

```
xmlns:name="URI of schema"
```

The name doesn't have to be provided if the namespace is assumed to be the default (no prefix is used) within the document:

```
xmlns="URI of schema"
```

The namespace declaration for RDF vocabularies usually points to the URI of the RDF Schema document for the vocabulary. Though there is no formalized checking of this document involved in RDF/XML—it's not a DTD—the document should exist as documentation for the schema. In particular, as we'll see in later chapters, this schema is accessed directly by tools and utilities used to explore and view RDF/XML documents.

An element that has been known to generate a great deal of conversation within the RDF/XML and XML community is the *QName*—a namespace prefix followed by a colon (:) followed by an XML local name. In the examples shown so far, all element and attribute names have been identified using the QName, a requirement within RDF/XML. An example use of a QName is:

```
<rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
  <pstcn:author rdf:parseType="Literal" />
</rdf:Description>
```

In this example, the QName for the RDF Description class and the `about` and `rdf:parseType` attributes is `rdf`, a prefix for the RDF syntax URI, given earlier. The QName for the author element is `pstcn`, the PostCon URI prefix.

The actual prefix used, such as `rdf` and `pstcn`, can vary between documents, primarily because automated processes replace the prefix with the full namespace URI when processing the RDF data. However, by convention, the creators of a vocabulary usually set the particular prefix used, and users of the vocabulary are encouraged to use the same prefix for consistency. This makes the RDF/XML documents easier for humans to read.

In particular, the prefix for the RDF Syntax Schema is usually given as `rdf`, the RDF Schema is given as `rdfs`, and the Dublin Core schema (described in Chapter 6) is usually abbreviated as `dc`. And of course, PostCon is given as `pstcn`.

Earlier I mentioned that the QName is controversial. The reason is twofold:

First, the RDF specification requires that all element and attribute types in RDF/XML must be QNames. Though the reason for this is straightforward—allowing multiple schemas in the same document—the rule was not established with the very first releases of RDF/XML, and there is RDF/XML in use today, such as in Mozilla, (described in Chapter 14), in which attributes such as `about` are not decorated with the namespace prefix.

In order to ensure that these pre-existing applications don't break, the RDF Working Group has allowed some attributes to be non-namespace annotated. These attributes are:

- `ID`
- `bagID` (removed from the specification based on last call comments)
- `about`
- `resource`
- `parseType`
- `type`

When encountered, RDF/XML processors are required to expand these attributes by concatenating the RDF namespace to the attribute. Though these nonannotated attributes are allowed for backward compatibility, the WG (and yours truly) strongly recommend that you use QNames with your attributes. In fact, RDF/XML parsers may give a warning (but not an error) when these are used in a document. The only reason I include these nonannotated attributes in the book is so that you'll understand why these still validate when you come upon them in older uses of RDF/XML.

Another controversy surrounding QNames is their use as attribute values: specifically, using them as values for `rdf:about` or `rdf:type`. Example 3-7 shows an earlier version of the RDF/XML vocabulary used for demonstrations throughout the book and uses a QName for a attribute value. QName formatting is boldfaced in the example.

Example 3-7. Demonstrations of QName attribute values

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:bbd="http://www.burningbird.net/schema#">
  <rdf:Description rdf:about="http://www.burningbird.net/identifier/tutorials/xul.htm">
     <bbd:bio rdf:resource="bbd:bio"/>
     <bbd:relevancy rdf:resource="bbd:relevancy" />
  </rdf:Description>

  <rdf:Description rdf:about="bbd:bio">
     <bbd:Title>YASD Does Mozilla/Navigator 6.0</bbd:Title>
     <bbd:Description>Demonstrations of using XUL for interface development
     </bbd:Description>
     <bbd:CreationDate>May 2000</bbd:CreationDate>
     <bbd:ContentAuthor>Shelley Powers</bbd:ContentAuthor>
     <bbd:ContentOwner>Shelley Powers</bbd:ContentOwner>
     <bbd:CurrentLocation>N/A</bbd:CurrentLocation>
   </rdf:Description>

  <rdf:Description rdf:about="bbd:relevancy">
     <bbd:CurrentStatus>Inactive</bbd:CurrentStatus>
     <bbd:RelevancyExpiration>N/A</bbd:RelevancyExpiration>
```

Example 3-7. Demonstrations of QName attribute values (continued)

```
    <bbd:Dependencies>None</bbd:Dependencies>
  </rdf:Description>

</rdf:RDF>
```

Running this example through the RDF Validator results in a perfectly good RDF graph and no errors or warnings. Many tools also have no problems with the odd use of QName. Apply this practice in your RDF/XML vocabulary, though, and you'll receive howls from the RDF community—this is a bad use of QNames, though not necessarily a specifically stated *invalid* use of them. The relationship between QNames and URIs is still not completely certain.

RDF Blank Nodes

It would be easy to extrapolate a lot of meaning about blank nodes but, bottom line, a blank node represents a resource that isn't currently identified. As with the infamous null value from the relational data model, there could be two reasons why the identifying URI is absent: either the value will never exist (isn't meaningful) or the value could exist but doesn't at the moment (currently missing).

Most commonly, a blank node—known as a *bnode*, or occasionally *anonymous node*—is used when a resource URI isn't meaningful. An example of this could be a representation of a specific individual (since most of us don't think of humans with URIs).

In RDF/XML, a blank node is represented by an oval (it is a resource), with either no value in the oval or a computer-generated identifier. The RDF/XML Validator generates an identifier, which it uses within the blank node to distinguish it from other blank nodes within the graph. Most tools generate an identifier for blank nodes to differentiate them.

In Example 3-8, `bio` attributes are grouped within an enclosing PostCon bio resource. Since the `bio` doesn't have its own URI, a blank node represents it within the model.

Example 3-8. Blank node within RDF model

```
<?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/">

  <rdf:Description rdf:about="monsters1.htm">
     <pstcn:bio>
       <rdf:Description>
         <pstcn:title>Tale of Two Monsters: Legends</pstcn:title>
```

Example 3-8. Blank node within RDF model (continued)

```
            <pstcn:description>
               Part 1 of four-part series on cryptozoology, legends,
               Nessie the Loch Ness Monster and the giant squid.
            </pstcn:description>
         <pstcn:created>1999-08-01T00:00:00-06:00</pstcn:created>
         <pstcn:creator>Shelley Powers</pstcn:creator>
      </rdf:Description>
     </pstcn:bio>
   </rdf:Description>
</rdf:RDF>
```

Running this example through the RDF Validator gives the directed graph shown in Figure 3-6 (modified to fit within the page).

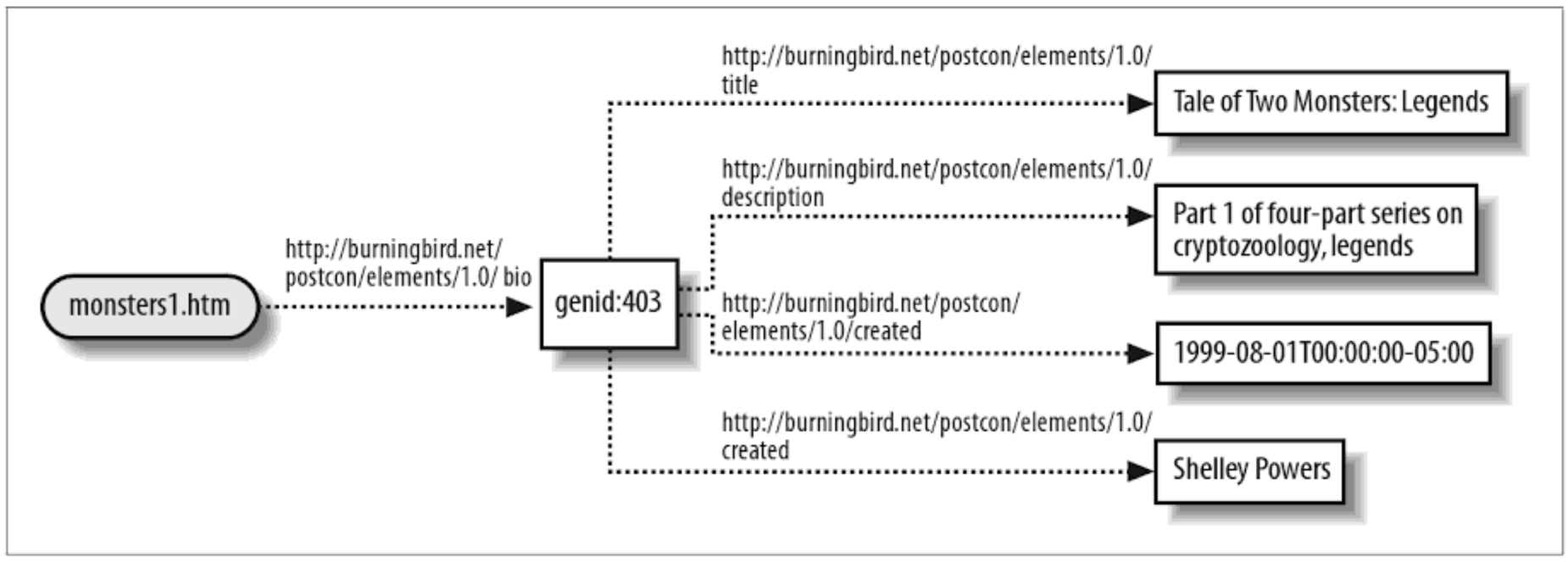
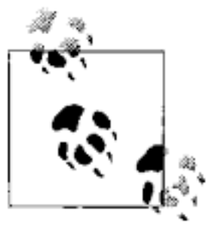


Figure 3-6. Directed graph demonstrating blank node

As you can see in the graph, the RDF Validator has generated a node identifier for the blank node, `genid:403`. This identifier has no meaning other than being a way to differentiate this blank node from other blank nodes, within the graph and within the generated N-Triples.



Example 3-8 also uses `xml:base` to establish a base URI for the other URIs in the document, avoiding a lot of repetition. This technique is described in more detail in the next section titled "URI References."

Instead of letting the tools provide a blank node identifier, you can provide one yourself. This is particularly useful if you want to reference a resource that's not nested within the outlying element but occurs elsewhere in the page as a separate RDF/XML triple. The `rdf:nodeID` is used to provide a specific identifier, as demonstrated in Example 3-9, when the embedded `bio` is pulled out into a separate triple. The `rdf:nodeID` attribute is used within the predicate of the original triple, as well as within the description of the newly created triple, as noted in bold type.

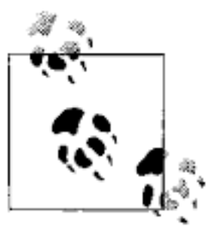
Example 3-9. Using rdf:nodeID to identify a unique blank node

```
<?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/">

  <rdf:Description rdf:about="monsters1.htm">
     <pstcn:bio rdf:nodeID="monsters1">
     </pstcn:bio>
   </rdf:Description>

   <rdf:Description rdf:nodeID="monsters1">
     <pstcn:title>Tale of Two Monsters: Legends</pstcn:title>
      <pstcn:description>
            Part 1 of four-part series on cryptozoology, legends,
            Nessie the Loch Ness Monster and the giant squid.
      </pstcn:description>
     <pstcn:created>1999-08-01T00:00:00-06:00</pstcn:created>
     <pstcn:creator>Shelley Powers</pstcn:creator>
  </rdf:Description>
</rdf:RDF>
```

The `rdf:nodeID` is unique to the document but not necessarily to all RDF/XML documents. When multiple RDF models are combined, the tools used could redefine the identifier in order to ensure that it is unique. The `rdf:nodeID` is *not* a way to provide a global identifier for a resource in order to process it mechanically when multiple models are combined. If you need this type of functionality, you're going to want to give the resource a formal URI, even if it is only a placeholder URI until a proper one can be defined.



As noted in the RDF Syntax Specification document, `nodeID` is unique to RDF/XML only, and does not have any representation within the RDF abstract model. It's a tool to help people work with RDF/XML; not part of the RDF model.

URI References

All predicates within RDF/XML are given as URIs, and most resources—other than those that are treated as blank nodes—are also given URIs. A basic grounding of URIs was given in Chapter 2, but this section takes a look at how URIs are used within the RDF/XML syntax.

Resolving Relative URIs and xml:base

Not all URI references in a document are full URIs. It's not uncommon for relative URI references to be given, which then need to be resolved to a base URI location. In the previous examples, the full resource URI is given within the `rdf:about` attribute.

Instead of using the full URI, the example could be a relative URI reference, which resolves to the base document concatenated with the relative URI reference. In the following, the relative URI reference `"#somevalue.htm"`:

```
<rdf:Description rdf:about="#somevalue">
```

then becomes `http://burningbird.net/articles/somedoc.htm#somevalue` if the containing document is `http://burningbird.net/articles/somedoc.htm`. To resolve correctly, the relative URI reference must be given with the format of pound sign (#) followed by the reference (`"#somevalue"`).

Normally, when a full URI is not provided for a specific resource, the owning document's URL is considered the base document for forming full URIs given relative URI references. So if the document is `http://burningbird.net/somedoc.htm`, the URI base is considered to be this document, and changes of the document name or URL change the URI for the resource.

With `xml:base`, you can specify a base document that's used to generate full URIs when given relative URI references, regardless of the URL of the owning document. This means that your URIs can be consistent regardless of document renaming and movement.

The `xml:base` attribute is added to the RDF/XML document, usually in the same element tag where you list your namespaces (though it can be placed anywhere). Redefining Example 3-6 with `xml:base` and using a relative URI reference would give you the RDF/XML shown in Example 3-10.

Example 3-10. Using xml:base to define the base document for relative URI references

```
<?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/">
  <rdf:Description rdf:about="monsters3.htm">
    <pstcn:author rdf:parseType="Literal" />
  </rdf:Description>
</rdf:RDF>
```

The URI for the article, given as relative `"monsters3.htm"`, is correctly expanded to the proper full URI of `http://burningbird.net/articles/monsters3.htm`.

Resolving References with rdf:ID

In the previous example, the `rdf:about` attribute was used to provide the URI reference. Other ways of providing a URI for a resource are to use the `rdf:resource`, `rdf:ID`, or `rdf:bagID` attributes. The `rdf:bagID` attribute is discussed in the next chapter, but we'll take a quick look at `rdf:ID` and `rdf:resource`.

Unlike the `rdf:about` attribute, which refers to an existing resource, `rdf:ID` generates a URI by concatenating the URI of the enclosing document (or the one provided by `xml:base`) to the identifier given, preceded by the relative URI # symbol. Rewriting Example 3-5 to use `rdf:ID` for the second resource results in the RDF/XML shown in Example 3-11.

Example 3-11. Using rdf:ID to provide identifier for resource

```
<?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/">

<rdf:Description rdf:ID="monsters3.htm">
  <pstcn:author>Shelley Powers</pstcn:author>
  <pstcn:title>Architeuthis Dux</pstcn:title>
  <pstcn:series>
    <rdf:Description rdf:ID="monsters.htm">
      <pstcn:seriesTitle>A Tale of Two Monsters
             </pstcn:seriesTitle>
    </rdf:Description>
  </pstcn:series>
  <pstcn:contains>Physical description of giant squids</pstcn:contains>
  <pstcn:alsoContains>Tale of the Legendary Kraken
             </pstcn:alsoContains>
</rdf:Description>

</rdf:RDF>
```

The generated RDF graph would show a resource giving the URI of the enclosing document, a pound sign (#), and the ID. In this case, if the enclosing document was at *http://burningbird.net/index.htm*, it would show a URI of `http://burningbird.net/index.htm#monsters3.htm`. This same effect can be given with the `rdf:about` by using a URI of `"#monsters"`.

As you can see, the URI of the resolved relative URI reference doesn't match that given previously: `http://burningbird.net/index.htm#monsters3.htm` does not match `http://burningbird.net/articles/monsters3.htm`. Based on this, I never use `rdf:ID` for actual resources; I tend to use it when I'm defining a resource that usually wouldn't have an actual URI but would have one primarily to support the required node-arc-node-arc-node nature of RDF/XML.

For example, the `pstcn:series` attribute given to the `http://burningbird.net/articles/monsters.htm` URI really doesn't exist—it's a way of showing a relationship between the article and a particular series, which has properties in its own right though it does not actually exist as a single object. Instead of using the full URI, what I could have done is use `ID`, as shown in Example 3-12.

Example 3-12. Using xml:base to identify the base document for all relative URI references

```
<?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/">
  <rdf:Description rdf:about="monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
    <pstcn:title>Architeuthis Dux</pstcn:title>
    <pstcn:series>
      <rdf:Description rdf:ID="monsters">
        <pstcn:seriesTitle>A Tale of Two Monsters
              </pstcn:seriesTitle>
      </rdf:Description>
    </pstcn:series>
    <pstcn:contains>Physical description of giant squids</pstcn:contains>
    <pstcn:alsoContains>Tale of the Legendary Kraken
              </pstcn:alsoContains>
  </rdf:Description>
</rdf:RDF>
```

The relative URI then resolves to `http://burningbird.net/articles/#monsters`, forming a representation of the URI as an identifier rather than an actual URL (a misunderstanding that can occur with URI references, since not all URIs are URLs). The `rdf:ID` is considered to have reified the statement (i.e., formally identified the statement within the model). The discussion about reification is continued in Chapter 4.

Representing Structured Data with rdf:value

Not all data relations in RDF represent straight binary connections between resource and object value. Some data values, such as measurement, have both a value and additional information that determines how you treat that value. In the following RDF/XML:

```
<pstcn:lastEdited>18</pstcn:lastEdited>
```

the statement is ambiguous because we don't know exactly what `18` means. Is it 18 days? Months? Hours? Did a person identified by the number 18 edit it?

To represent more structured data, you can include the additional information directly in the value:

```
<pstcn:lastEdited>18 days</pstcn:lastEdit>
```

However, this type of intelligent data then requires that systems know enough to split the value from its qualifier, and this goes beyond what should be required of RDF parsers and processors. Instead, you could define a second vocabulary element to capture the qualifier, such as:

```
<pstcn:lastEdited>18</pstcn:lastEdited>
<pstcn:lastEditedUnit>day</pstcn:lastEditedUnit>
```

This works, but unfortunately, there is a disconnect between the value and the unit because the two are only indirectly related based on their relationship with the resource. So the syntax is then refined, which is where `rdf:value` enters the picture. When dealing with structured data, the `rdf:value` predicate includes the actual value of the structure—it provides a signal to the processor that the data itself is included in this field, and all other members of the structure are qualifiers and additional information about the structure.

Redefining the data would then result in:

```
<pstcn:lastEdited rdf:parseType="Resource">
    <rdf:value>18</rdf:value>
   <pstcn:lastEditedUnit>day</pstcn:lastEditedUnit>
</pstcn:lastEdited>
```

Now, not only do we know that we're dealing with structured data, we know what the actual value, the kernel of the data so to speak, is by the use of `rdf:value`. You could use your own predicate, but `rdf:value` is global in scope—it crosses all RDF vocabularies—making its use much more attractive if you're concerned about combining your vocabulary data with other data.

The rdf:type Property

One general piece of information that is consistent about an RDF resource—outside of the URI to uniquely identify it—is the resource or class type. In the examples shown thus far, this value could implicitly be `"Web Resource"` to refer to all of the resources, or could be explicitly set to `"article"` for articles. All these would be correct, depending on how generically you want to define the resource and the other properties associated with the resource. To explicitly define the resource type, you would use the RDF `rdf:type` property.

Usually the `rdf:type` property is associated at the same level of granularity as the other properties. As the resources defined using RDF in this chapter all have properties associated more specifically with an article than a web resource, the RDF type property would be `"article"` or something similar.

In the next section, covering RDF containers, we will learn that the resource type for an RDF container would be the type of container rather than the type of the contained property or resource. Again, the type is equivalent to the granularity of the resource being described, and with containers, the resource is a canister (or group) of resources or properties rather than a specific resource or property.

The value of the RDF `rdf:type` property is a URI identifying an `rdfs:Class`-typed resource (`rdfs:Class` is described in detail in Chapter 5). To demonstrate how to attach an explicit type to a resource, Example 3-13 shows the resource defined in the RDF/XML for Example 3-1, but this time explicitly defining an RDF Schema element for the resource.

Example 3-13. Demonstrating the explicit resource property type

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:Author>Shelley Powers</pstcn:Author>
    <pstcn:Title>Architeuthis Dux</pstcn:Title>
    <rdf:type rdf:resource="http://burningbird.net/postcon/elements/1.0/Article" />
  </rdf:Description>
</rdf:RDF>
```

The type property includes a resource reference for the schema element, in this case for the `Article` class.

Rather than formally list out an `rdf:Description` and then attach the `rdf:type` predicate to it, you can cut through all of that using an RDF/XML shortcut. Incorporating the formal syntax of the type property directly into XML, as before, the type property is treated as an embedded element of the outer resource.

Within the shortcut, the type property is created directly as the element type rather than as a generic RDF Description element. This new syntax, demonstrated in Example 3-14, leads to correct interpretation of the RDF within an XML parser.

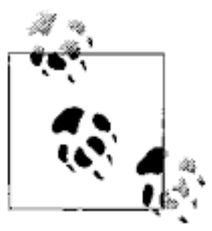
Example 3-14. Abbreviated syntax version of type property

```
<?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/">
  <pstcn:Article rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:Author>Shelley Powers</pstcn:Author>
    <pstcn:Title>Architeuthis Dux</pstcn:Title>
  </pstcn:Article>
</rdf:RDF>
```

Notice the capitalization of the first letter for `Article`. This provides a hint that the element is a resource, rather than a predicate type.

This shortcut approach is particularly effective in ensuring that there is no doubt as to the nature of the resource being described, especially since formally listing an `rdf:type` predicate isn't a requirement of the RDF/XML. As you'll see later, in Chapter 6, the PostCon vocabulary uses this shortcut technique to identify the major resource as a web document.

Other RDF/XML shortcuts that can help cut through some of the rather stylized RDF/XML formalisms and make the underlying model a little more opaque are described in the next section.



An RDF resource can have more than one `rdf:type` associated with it.

RDF/XML Shortcuts

An RDF/XML shortcut is just what it sounds like—an abbreviated technique you can use to record one specific characteristic of an RDF model within RDF/XML. In the last section, we looked at using a shortcut to embed a resource's type with the resource definition. Other RDF/XML shortcuts you can use include:

- Separate predicates can be enclosed within the same resource block.
- Nonrepeating properties can be created as resource attributes.
- Empty resource properties do not have to be formally defined with description blocks.

The first shortcut or abbreviated syntax—enclosing all predicates (properties) for the same subject within that subject block—is so common that it's unlikely you'll find RDF/XML files that repeat the resource for each property. However, the RDF/XML in Example 3-13 is equivalent to that shown in Example 3-15.

Example 3-15. Fully separating each RDF statement into separate XML block

```
<?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/">
  <rdf:Description rdf:about="http://dynamicearth.com/articles/monsters3.htm">
    <pstcn:author>Shelley Powers</pstcn:author>
  </rdf:Description>
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:title>Architeuthis Dux</pstcn:title>
  </rdf:Description>
</rdf:RDF>
```

If you try this RDF/XML within the RDF Validator, you'll get exactly the same model as you would with the RDF/XML from Example 3-1.

The RDF/XML from Examples 3-1 and 3-13 also demonstrates that you can generate an RDF graph from RDF/XML, but when you then convert it back into RDF/XML from the graph, you won't always get the same RDF/XML that you started with. In this example, the graph for both RDF/XML documents would most likely reconvert back to the document shown in Example 3-1, rather than the one shown in Example 3-13.

For the second instance of abbreviated syntax, we'll again return to RDF/XML in Example 3-1. Within this document, each of the resource properties is listed within a separate XML element. However, using the second abbreviated syntax—nonrepeating properties can be created as resource attributes—properties that don't repeat and are literals can be listed directly in the resource element, rather than listed out as separate formal predicate statements.

Rewriting Example 3-1 as Example 3-16, you'll quickly see the difference with this syntactic shortcut.

Example 3-16. Original RDF/XML document rewritten using an abbreviated (shortcut) syntax

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm"
    pstcn:author="Shelley Powers"
    pstcn:title="Architeuthis Dux" />
</rdf:RDF>
```

As you can see, this greatly simplifies the RDF/XML. RDF parsers interpret the XML in Examples 3-1 and 3-14 as equivalent, as you can see if you run this newer example through the RDF Validator.

There are actually two different representations of the third abbreviation type, having to do with formalizing predicate objects that are resources. In the examples, RDF resources have been identified within the `<rdf:Description>...</rdf:Description>` tags, using a formal striped XML syntax format, even if the resource is an object of the statement rather than the subject. However, the `rdf:Description` block doesn't have to be provided if the resource objects match one of two constraints.

The first constraint is that the resource object must have a URI but must not itself have predicates. It is an empty element. For instance, to record information about documents that are related to the document being described, you could use a related predicate with an `rdf:resource` value giving the document's URI, as shown in Example 3-17.

Example 3-17. Using rdf:resource to identify an empty resource object

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:Author>Shelley Powers</pstcn:Author>
    <pstcn:Title>Architeuthis Dux</pstcn:Title>
    <pstcn:related rdf:resource="http://burningbird.net/articles/monsters1.htm" />
   </rdf:Description>
</rdf:RDF>
```

You can also use the `rdf:resource` attribute to designate a resource that's described later in the document. This is an especially useful technique if a resource object is identified early on, but you didn't know if the object had properties itself. If you discover properties for the object at a later time, a separate `rdf:Description` can be defined for the resource object, and the properties added to it.

In Example 3-17, the related resource object is shown without properties itself. In Example 3-18, properties for this resource have been given, in this case a `reason` that the resource object is related to the original resource.

Example 3-18. Using rdf:resource to identify a resource that's defined later in the document

```
<?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/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:Author>Shelley Powers</pstcn:Author>
    <pstcn:Title>Architeuthis Dux</pstcn:Title>
    <pstcn:related rdf:resource="http://burningbird.net/articles/monsters1.htm" />
   </rdf:Description>

  <rdf:Description rdf:about="http://burningbird.net/articles/monsters1.htm">
    <pstcn:reason>First in the series</pstcn:reason>
  </rdf:Description>
</rdf:RDF>
```

Of course, this wouldn't be RDF if there weren't options in how models are serialized with RDF/XML. Another variation on using `rdf:resource` for an object resource is to identify the property object as a resource and then use the shortcut technique shown earlier—adding predicates as attributes directly. With this, you wouldn't need to define a separate `rdf:Description` block just to add the related property's reason. In fact, Example 3-19 shows all of the shortcut techniques combined to simplify one RDF/XML document.

Example 3-19. RDF/XML document demonstrating all RDF/XML shortcuts

```
<?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/">
  <pstcn:Article
    pstcn:author="Shelley Powers"
    pstcn:title="Architeuthis Dux"
    rdf:about="http://dynamicearth.com/articles/monsters3.htm" >
    <pstcn:related rdf:resource="http://burningbird.net/articles/monsters1.htm"
        pstcn:reason="First in the series" />
  </pstcn:Article>

</rdf:RDF>
```

Again, running this example through the validator results in a graph that's identical to that given if more formalized RDF/XML were used, as shown in Figure 3-7.

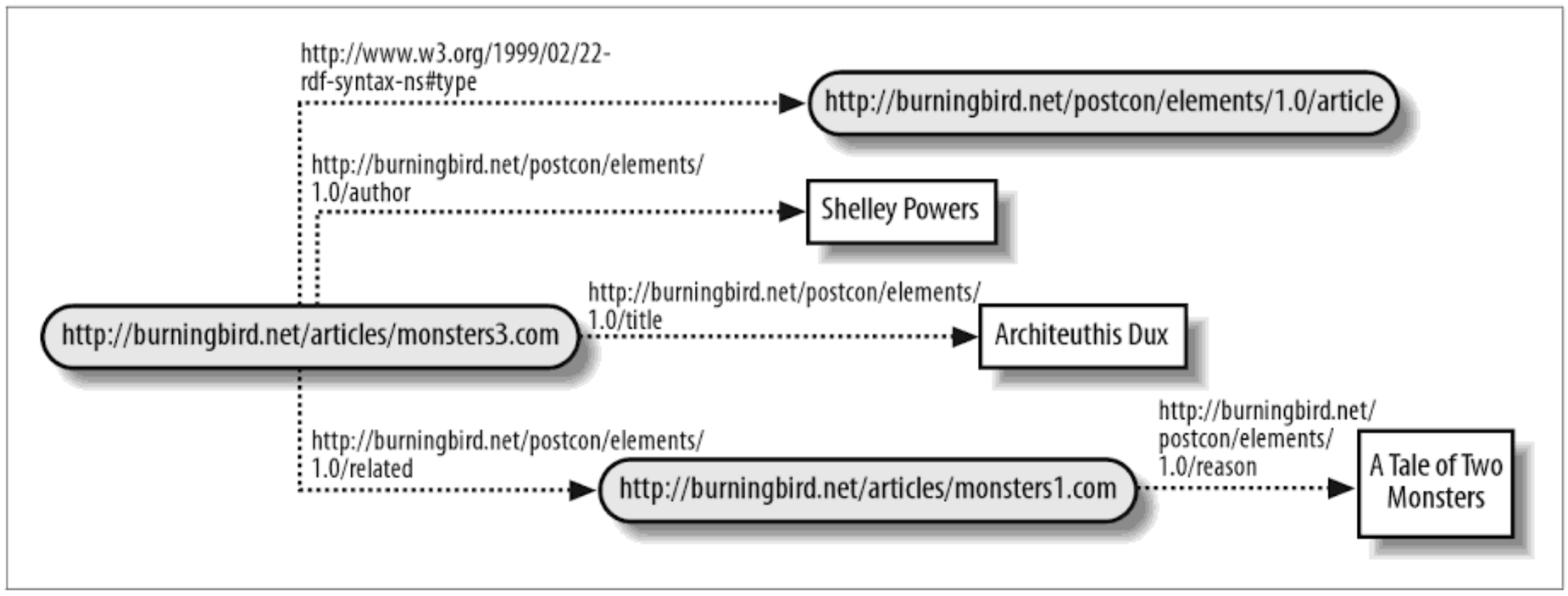


Figure 3-7. RDF directed graph of RDF/XML document created using shortcut techniques shown in Example 3-19

Which syntax should you use, formal or shortcut? According to the W3C Syntax Specification (revised), applications that can generate, query, or consume RDF are expected to support both the formal syntax and the abbreviated syntax, so you should be able to use both, either separately or together. The abbreviated syntax is less verbose, and the RDF model documented within the RDF is more clearly apparent. In fact, according to the specification, a benefit of using the abbreviated syntax is that the RDF model can be interpreted directly from the XML (with the help of some carefully designed DTDs).

What do I mean by that last statement? As an example, within the formal syntax, RDF properties are included as separate tagged elements contained within the outer RDF Description element. Opening an XML file such as this using an XML parser, such as in a browser, the properties would display as separate elements—connected to the description, true, but still showing, visibly, as separate elements.

Using the second form of the abbreviated syntax, the properties are included as attributes within the description tag and therefore don't show as separate elements. Instead, they show as descriptive attributes of the element being described, the resource. With rules and constraints enforced through a DTD, the attributes can be interpreted, directly and appropriately, within an XML document using an XML parser (not a specialized RDF parser) as a resource with given attributes (properties)—not an element with embedded, nested elements contained within an outer element.

This same concept of direct interpretation of the RDF model applies to nested resources. Using the formal syntax, a property that's also a resource is listed within a separate Description element and associated to the original resource through an identifier. An XML parser would interpret the two resources as separate elements without any visible association between the two. Using the abbreviated syntax, the resource property would be nested within the original resource's description; an XML parser would show that the resource property is a separate element, but associated with the primary resource by being embedded within the opening and closing tags of this resource.

More on RDF Data Types

RDF data types were discussed in Chapter 2, but their impact extends beyond just the RDF abstract model and concepts. RDF data types have their own XML constructs within the RDF/XML specification.

For instance, you can use the `xml:lang` attribute to specify a language for each RDF/XML element. In the examples in this English-language book, the value would be `"en"`, and would be included within an element as follows:

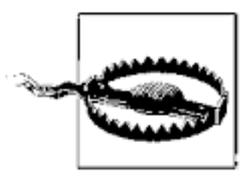
```
<pstcn:reason xml:lang="en">First in the series</pstcn:reason>
```

You can find out more about `xml:lang` at *http://www.w3.org/TR/REC-xml#sec-lang-tag*.

You can also specify a general type for a predicate object with `rdf:parseType`. We've seen `rdf:parseType` of `"Resource"`, but you can also use `rdf:parseType` of `"Literal"`:

```
<pstcn:reason xml:lang="en" rdf:parseType="Literal"><h1>Reason</h1></pstcn:reason>
```

By using `rdf:parseType="Literal"`, you are telling the RDF/XML parser to treat the contents of a predicate as a literal value rather than parse it out for new RDF/XML elements. This allows you to embed XML into an element that is not parsed.



Some implementations of RDF/XML specifically recommend using `rdf:parseType="Literal"` as a way of including unparsed XML within a document, to bypass having to formalize the XML into an RDF/XML valid syntax. This attribute was never intended to bypass best practices. If the data contained in the attribute is recurring, best practice would be to formalize the XML into RDF/XML and incorporate it into the vocabulary or create a new vocabulary.

RDF also allows for typed literals, which contain a reference to the data type of the literal compatible with the XML Schema data types. In the N3 notation, the typed literal would look similar to the following, as pulled from the RDF Primer:

```
ex:index.html  exterms:creation-date  "1999-08-16"^^xsd:date .
```

The format of the literal string is value (`1999-08-16`), data type URI (`^^` in this example), and XML Schema data type (`xsd:date`).

As interesting as this format is, one could see how this approach lacks some popularity, primarily because of the intelligence built directly into the string, which can be missed depending on the XML parser that forms the basis of the RDF/XML parser. Luckily, within RDF/XML, the data type is specified as an attribute of the element, using the `rdf:datatype` attribute, as demonstrated in Example 3-20, which is a copy of Example 3-1, but with data typing added.

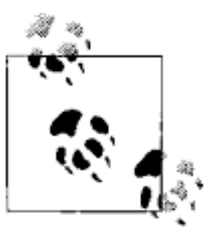
Example 3-20. Demonstration of typed literal in RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF
```

Example 3-20. Demonstration of typed literal in RDF/XML (continued)

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:pstcn="http://burningbird.net/postcon/elements/1.0/">
  <rdf:Description rdf:about="http://burningbird.net/articles/monsters3.htm">
    <pstcn:author rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                                                  Shelley Powers</pstcn:author>
    <pstcn:title rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                                                  Architeuthis Dux</pstcn:title>
  </rdf:Description>
</rdf:RDF>
```

There is no requirement to use data types with literals—it is up to not only the vocabulary designer but also those who generate instances of the vocabulary to decide if they wish to use typed literals. No implicit semantics are attached to typed literals, by which I mean toolmakers are not obliged to double-check the validity of a particular literal against its type. Additionally, there's no requirement that toolmakers even have to differentiate between the types or ensure that typed literals used in an instance map to the same typed literals for the RDF Schema of the vocabulary. Typed literals are more of a way to communicate data types between vocabulary users than between vocabulary-automated processes.



You can read more about XML Schema built-in data types at *http://www.w3.org/TR/xmlschema-2/*. *XML.com* also has a number of articles covering XML Schema and data typing in general.

RDF/XML: Separate Documents or Embedded Blocks

By convention, RDF/XML files are stored as separate documents and given the extension of *.rdf* (just *rdf* for Mac systems). The associated MIME type for an RDF/XML document is: *application/rdf+xml*.

There's been considerable discussion about embedding RDF within other documents, such as within non-RDF XML and HTML. I've used RDF embedded within HTML pages, and I know other applications that have done the same.

The problem with embedding, particularly within HTML documents, is that it's not a simple matter to separate the RDF/XML from the rest of the content. If the RDF/XML used consists of a resource and its associated properties listed as attributes of the resource, this isn't a problem. An example of this would be:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description
    about="http://burningbird.net/cgi-bin/mt-tb.cgi?tb_id=121"
    dc:title="Good RSS"
    dc:identifier="http://weblog.burningbird.net/archives/000619.php"
```

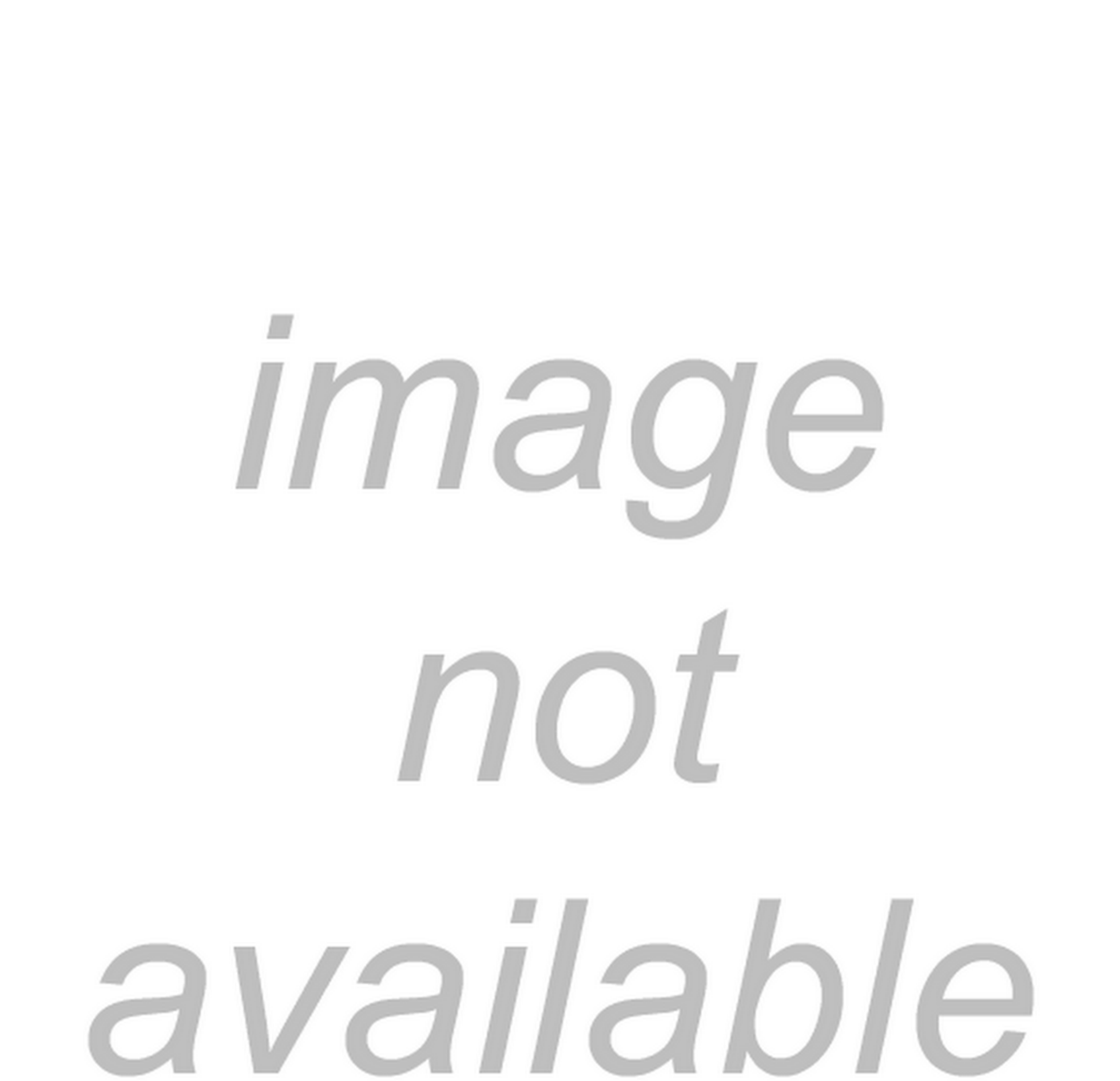


image
not
available

Example 3-21. Embedding RDF in HTML script elements (continued)

```
</rdf:RDF>
</script>
```

The HTML parser ignores the script contents, assuming that the `text/rdf` content will be processed by some application geared to this data type. This approach works rather well except for one thing: it doesn't allow an HTML page to validate as XHTML. And many organizations insist that web pages validate as XHTML.

To allow the page with the embedded RDF to validate, you can then surround the contents with HTML comments:

```
<!--
RDF/XML
-->
```

Unfortunately, HTML comments are also XML comments, and any content within them tends to be ignored by most XML parsers, including RDF/XML parsers.

Until XML can be embedded into an XHTML document in such a way that allows the page to be validated, the only approach you can take for the RDF data is to include it in an external RDF document and then link the document into the XHTML page using the link element:

```
<link rel="meta" type="application/rdf+xml" title="RSS"
href="http://burningbird.net/index.rdf" />
```

Another approach is to embed the RDF/XML into the XHTML using comments but to pull this data out and feed it directly to an RDF/XML parser. It's a bit cumbersome, but doable, especially since most screen-scraping technologies such as Perl's LWP provide for finding specific blocks of data and grabbing them directly.

Example 4-14. Using reification to attach the originator of trust (continued)

```
   <pstcn:recommendedBy rdf:resource="#s1" />
  </rdf:Description>

</rdf:RDF>
```

With this modification, the search engine results would be:

```
Some Bookstore, found at http://www.somebookstore.com/, is an online bookstore.
Trust in this store is high. The assertion about the type of store and the trust
in the store is provided by Shelley Powers.
```

Now the person shopping for an online bookstore has the information necessary to verify the source of the level of trust. Of course, the person would then have to determine if the source of the information is someone who can *also* be trusted. (Trust me. I can be trusted.)

Metadata about statements

Another use of reification is to record metadata information about a specific statement. For instance, if the statement about the resource (not the resource itself) is valid only after a specific date or only within a specific area or use, this type of information can be recorded using reification. Reification should be used because statement properties would associate the information directly to the resource, rather than to the statement.

One of the problems with the web today is that so many links to sites are obsolete, primarily because the original resource has been removed or moved to a new location. Web pages can have an expiration date attached to them, but that's not going to help when adding a link to the web resource among your own pages. It's the link or reference that needs to age gracefully, not the original resource.

To solve this, valid date information can be attached to the reference to the web resource, rather than being attached directly to the resource itself.

In Example 4-15, very simple RDF is used to describe a resource, an article, containing vacation and travel spot information. Attached to this recommendation is a constraint that the reference to this article is valid only for the year 2002.

Example 4-15. Providing a valid date for an article

```
<?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/">
  <rdf:Description>
    <rdf:subject rdf:resource="http://burningbird.net/somearticle.htm" />
    <rdf:predicate rdf:resource=
                         "http://burningbird.net/schema/Recommendations" />
    <rdf:object>Vacation and Travel Spots</rdf:object>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
```

Example 4-15. Providing a valid date for an article (continued)

```
    <pstcn:validFor>2002</pstcn:validFor>
  </rdf:Description>
</rdf:RDF>
```

By using reification, we've attached a valid date range to the *reference* to the article rather than directly to the article. We're saying that this reference (link) is valid only in the year 2002, rather than implying that the article the link is referencing is valid only in the year 2002.

CHAPTER 5

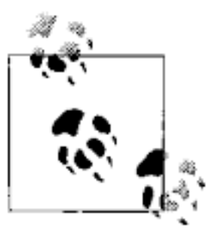
Important Concepts from the W3C RDF Vocabulary/Schema

When discussing the Resource Description Framework (RDF) specification, we're really talking about two different specifications—a Syntax Specification and a Schema Specification. As described in Chapters 3 and 4, the Syntax Specification shows how RDF constructs relate to each other and how they can be diagrammed in XML. For instance, elements such as `rdf:type` and `pstcn:bio` are used to describe a specific resource, providing information such as the resource's type and the author of the resource. The different namespace prefixes associated with each element (such as `rdf:` and `pstcn:`) represent the schema that particular element is defined within.

In the context of RDF/XML, a vocabulary or schema is a rules-based dictionary that defines the elements of importance to a domain and then describes how these elements relate to one another. It provides a type system that can then be used by domain owners to create RDF/XML vocabularies for their particular domains. For example, the `pstcn:bio` element is from a custom vocabulary created for use with this book while the `rdf:type` element is from the RDF vocabulary. These are different vocabularies and have different vocabulary owners, but both follow rules defined within the RDF Vocabulary Description Language 1.0: RDF Schema.

However, before getting into the details of the RDF Schema, consider the following: if RDF is a way of describing data, then the RDF Schema can be considered a domain-neutral way of describing the metadata that can then be used to describe the data for a domain-specific vocabulary.

If all this seems convoluted, then you'll appreciate reading more about the concept of metadata, its importance to existing applications, and how RDF fits into the concept, all discussed in the next section.



The material in this chapter references the RDF Vocabulary Description Language 1.0: RDF Schema. The most recent version of the document can be found at *http://www.w3.org/TR/rdf-schema/*.

Example 14-2. RDF/XML document used to provide data in template (continued)

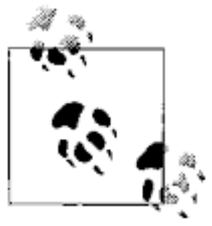
```
        </rdf:Seq>
    </rdf:li>
    <rdf:li>
        <rdf:Seq rdf:about="urn:weblog:writing">
          <rdf:li rdf:resource="urn:weblog:writing:rdfbook"/>
          <rdf:li rdf:resource="urn:weblog:writing:poetry"/>
          <rdf:li rdf:resource="urn:weblog:writing:review"/>
          <rdf:li rdf:resource="urn:weblog:writing:ebook"/>
          <rdf:li rdf:resource="urn:weblog:writing:online"/>
          <rdf:li>
              <rdf:Seq rdf:about="urn:weblog:writing:journal">
                 <rdf:li rdf:resource="urn:weblog:writing:journal:weblog"/>
                 <rdf:li rdf:resource="urn:weblog:writing:journal:paper"/>
              </rdf:Seq>
            </rdf:li>
          </rdf:Seq>
      </rdf:li>
      <rdf:li>
          <rdf:Seq rdf:about="urn:weblog:connecting">
            <rdf:li rdf:resource="urn:weblog:connecting:relationships"/>
            <rdf:li rdf:resource="urn:weblog:connecting:conferences"/>
          </rdf:Seq>
    </rdf:li>
  </rdf:Seq>

</rdf:RDF>
```

Note that the top-level `rdf:Seq` is given a URI of `urn:weblog:data`, matching the starting position given in the template. Each major category is given its own sequence and its own resource. Each title item is listed as an `rdf:li` and defined as a separate resource with both category and title.

When the data is processed, the rule attached in the template basically states that all category values are placed in the left column, and all titles in the right. Since the major categories don't have titles, the `treecells` for these values are blank. However, clicking on the drop-down indicator next to the categories displays both the minor category (subcategory) and titles for each row.

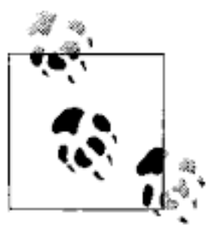
Earlier I mentioned there was a caveat about the validity of the RDF/XML used in the example. The RDF/XML document shown in Example 14-2 validates with the RDF Validator, but not all RDF/XML documents used in providing data for templates in Mozilla do. For instance, I separated out each `rdf:Seq` element, something that's not necessary with XUL but is necessary to maintain the RDF/XML striping (arc-node-arc-node). In addition, many of the XUL RDF/XML documents also don't qualify the `about` or `resource` attributes, which is discouraged in the RDF specifications. This doesn't generate an error, but does generate warnings. However, when you create your own RDF/XML documents, you can use the qualified versions without impacting on the XUL processing.



The Mozilla group wasn't the only organization to use RDF/XML to facilitate building a user interface. The Haystack project at MIT, *http://haystack.lcs.mit.edu/*, uses RDF as the primary data modeling framework.

Creative Commons License

The Creative Commons (CC) is an organization formed in 2002 to facilitate the movement of artists' work to the public domain. One of the outputs from the organization is the Creative Commons licenses: licenses that can be attached to a work of art such as a writing, a graphic, or a song, that provides information about how that material can be used and reused by others.



The Creative Commons web site is at *http://creativecommons.org*.

The CC licenses don't replace copyright and fair use laws; they primarily signal an artist's interest in licensing certain aspects of his copyright to the public, such as the right to copy a work, to derive new works from an original creation, and so on. The license is associated with the art in whatever manner is most expeditious, but if the art is digitized on the Web, the license is usually included with the art as RDF/XML.

The RDF/XML for use can be generated at the CC web site when you pick what particular license you want to apply. For instance, the following RDF/XML is generated when you pick a license that requires attribution and doesn't allow derivative works and/or commercial use:

```
<rdf:RDF xmlns="http://web.resource.org/cc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<Work rdf:about="">
<license rdf:resource="http://creativecommons.org/licenses/by-nd-nc/1.0" />
</Work>

<License rdf:about="http://creativecommons.org/licenses/by-nd-nc/1.0">
   <requires rdf:resource="http://web.resource.org/cc/Attribution" />
   <permits rdf:resource="http://web.resource.org/cc/Reproduction" />
   <permits rdf:resource="http://web.resource.org/cc/Distribution" />
   <prohibits rdf:resource="http://web.resource.org/cc/CommercialUse" />
   <requires rdf:resource="http://web.resource.org/cc/Notice" />
</License>

</rdf:RDF>
```

Normally this RDF/XML is included as part of a larger HTML block, and the RDF is enclosed in HTML comments to allow the page to validate as XHTML. Unfortunately, since HTML comments are also XML comments, this precludes accessing the

RDF/XML directly from the page for most parsers, which will ignore the data much as the HTML browsers do.

The CC RDF Schema makes use of several Dublin Core elements, such as `dc:title`, `dc:description`, `dc:subject` and so on. You can see the model breakdown at *http://creativecommons.org/learn/technology/metadata/implement#learn* and the schema itself at *http://creativecommons.org/learn/technology/metadata/schema.rdf*. The schema includes definitions for the CC elements, though it uses the `dc:description` and `dc:title` elements for this rather than the RDFS equivalents of `rdfs:comment` and `rdfs:label`. The namespace for the Creative Commons schema is `http://web.resource.org/cc/`, and the prefix usually used is `cc`.

Though CC makes use of Dublin Core elements, the data contained within these elements does differ from other popular uses of Dublin Core. A case in point is `dc:creator`. For the most part, `dc:creator` usually contains a string literal representing the name of the person who created the work. However, the CC folks, following from an earlier overly involved discussion in the RDF Interest Group surrounding the concept that "strings don't create anything," provided a bit more detail—in this case, that a `dc:creator` is an "agent," with a `dc:title` equivalent to the agent's name. In the following RDF/XML, the `dc:creator` field is boldfaced to demonstrate the structure of the data used by CC:

```
<rdf:RDF xmlns="http://web.resource.org/cc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<Work rdf:about="http://rdf.burningbird.net">
   <dc:title>Practical RDF</dc:title>
   <dc:date>2003-2-1</dc:date>
   <dc:description>Sample CC license for book</dc:description>
   <dc:creator><Agent>
      <dc:title>Shelley Powers</dc:title>
   </Agent></dc:creator>
   <dc:rights><Agent>
      <dc:title>O'Reilly</dc:title>
   </Agent></dc:rights>
   <dc:type rdf:resource="http://purl.org/dc/dcmitype/Text" />
<license rdf:resource="http://creativecommons.org/licenses/by-nd-nc/1.0" />
</Work>

<License rdf:about="http://creativecommons.org/licenses/by-nd-nc/1.0">
   <requires rdf:resource="http://web.resource.org/cc/Attribution" />
   <permits rdf:resource="http://web.resource.org/cc/Reproduction" />
   <permits rdf:resource="http://web.resource.org/cc/Distribution" />
   <prohibits rdf:resource="http://web.resource.org/cc/CommercialUse" />
   <requires rdf:resource="http://web.resource.org/cc/Notice" />
</License>

</rdf:RDF>
```

the search and to which you can add custom code as needed. Additionally, you can access the Seamark services through the Seamark API, a SOAP-based protocol.

The user interface for Seamark is quite simple, consisting of a main model/RDF document page, with peripheral pages to manage the application data. Once the application is installed, the first steps to take after starting the application are to create a model and then load one or more RDF/XML documents. Figure 15-2 shows the page form used to identify an internal RDF/XML document. Among the parameters specified is whether to load the document on a timed schedule, or manually, in addition to the URL of the file and the base URL used within the document. The page also provides space for an XSL stylesheet to transform non-RDF XML to RDF/XML.

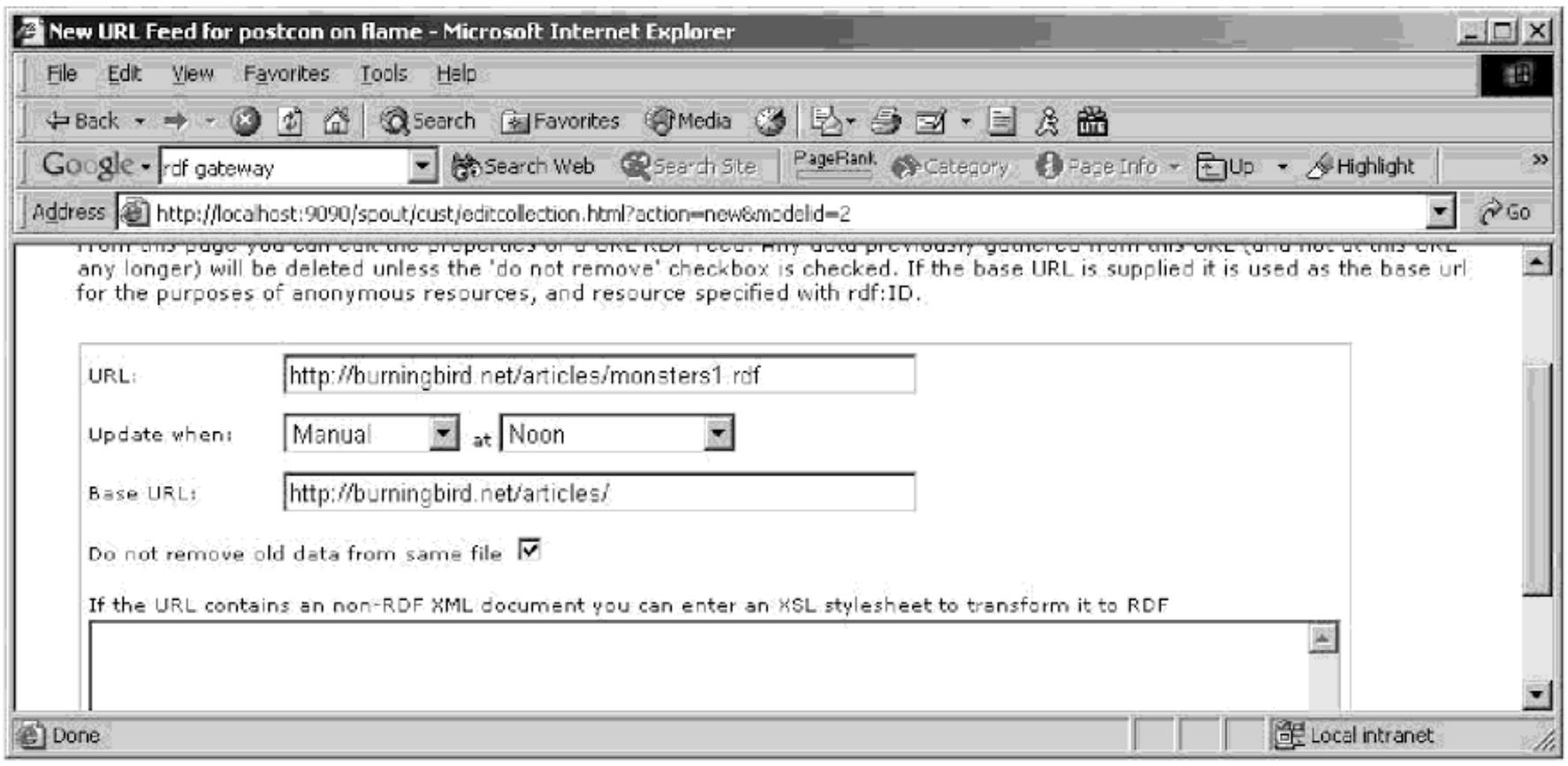


Figure 15-2. Adding a URL for an external RDF/XML data source

Once the external feed is defined, the data can then be loaded manually or allowed to load according to the schedule you defined.

After data is loaded into the Seamark repository, you can then create the search queries to access it. In the query page, Seamark lists out the RDFS classes within the document; you can pick among these and have the tool create the query for you or manually create the query.

For instance, the example RDF/XML used throughout the book, *http://burningbird.net/articles/monsters1.rdf*, has three separate classes:

`pstcn:Resource`
: Main object and any related resources

`pstcn:Movement`
: Resource movements

`rdf:Seq`
: The RDF sequence used to coordinate resource history

For my first query, I selected the Resource object, and had Seamark generate the query, as shown in Figure 15-3.

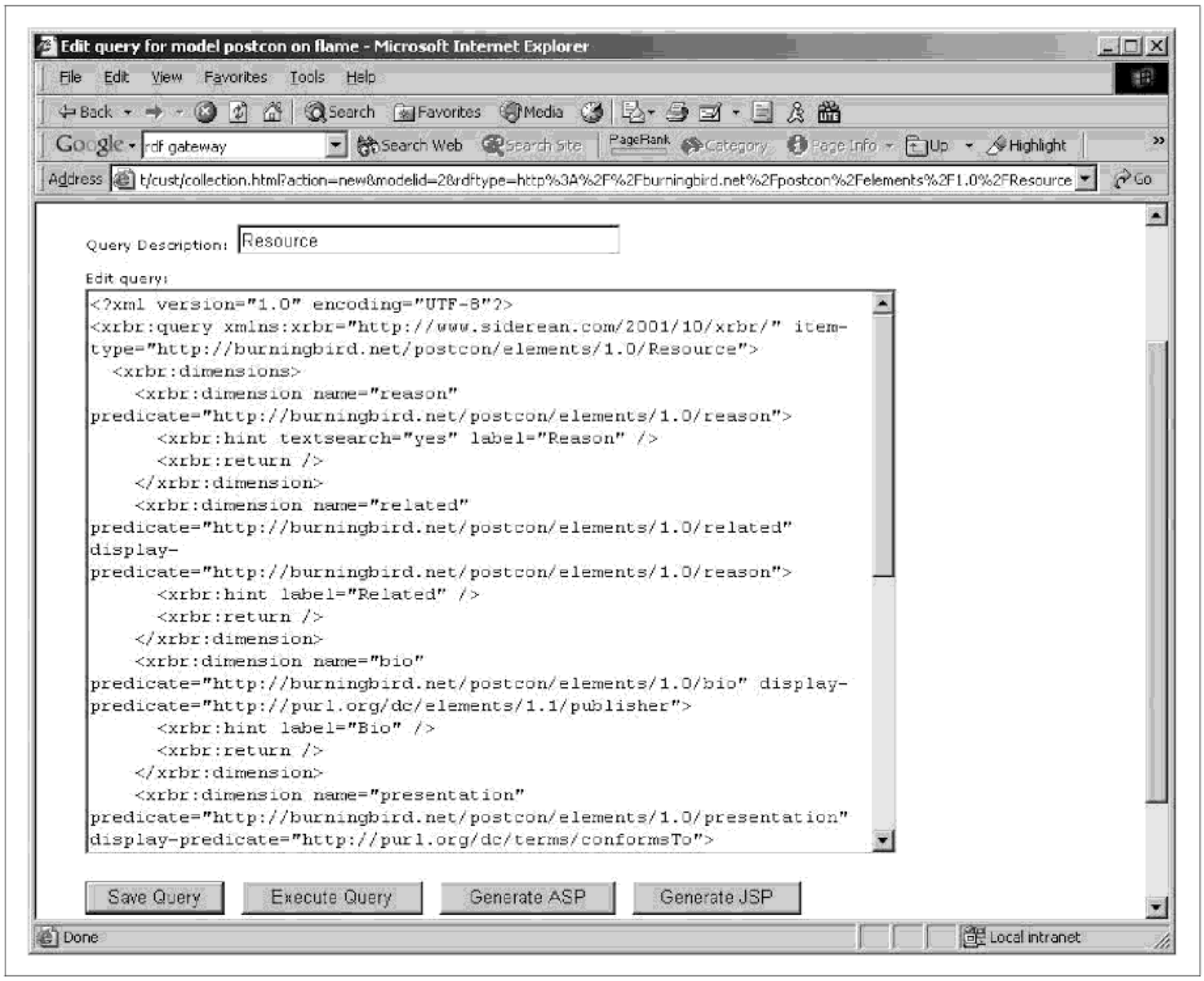


Figure 15-3. An automatically generated query in Seamark.

As you can see from the figure, XRBR isn't a trivial query language, though a little practice helps you work through the verbosity of the query. Once the initial XRBR is generated, you can customize the query, save it, execute it, or generate ASP or JSP to manage the query—or any combination of these options. Executing the query returns XRBR-formatted data, consisting of data and characteristics, or *facets* for all the Resource classes in the document. At this point, you can again customize the query or generate an ASP or JSP page.

When you add new RDF/XML documents to the repository, this new data is incorporated into the system, and running the query again queries the new data as well as the old. Figure 15-4 shows the page for the model with two loaded RDF/XML documents and one query defined.

Seamark comes with a default application called *bookdemo* that can be used as a prototype as well as a training tool. In addition, the application is easily installed and configured and comes with considerable documentation, most in PDF format. What

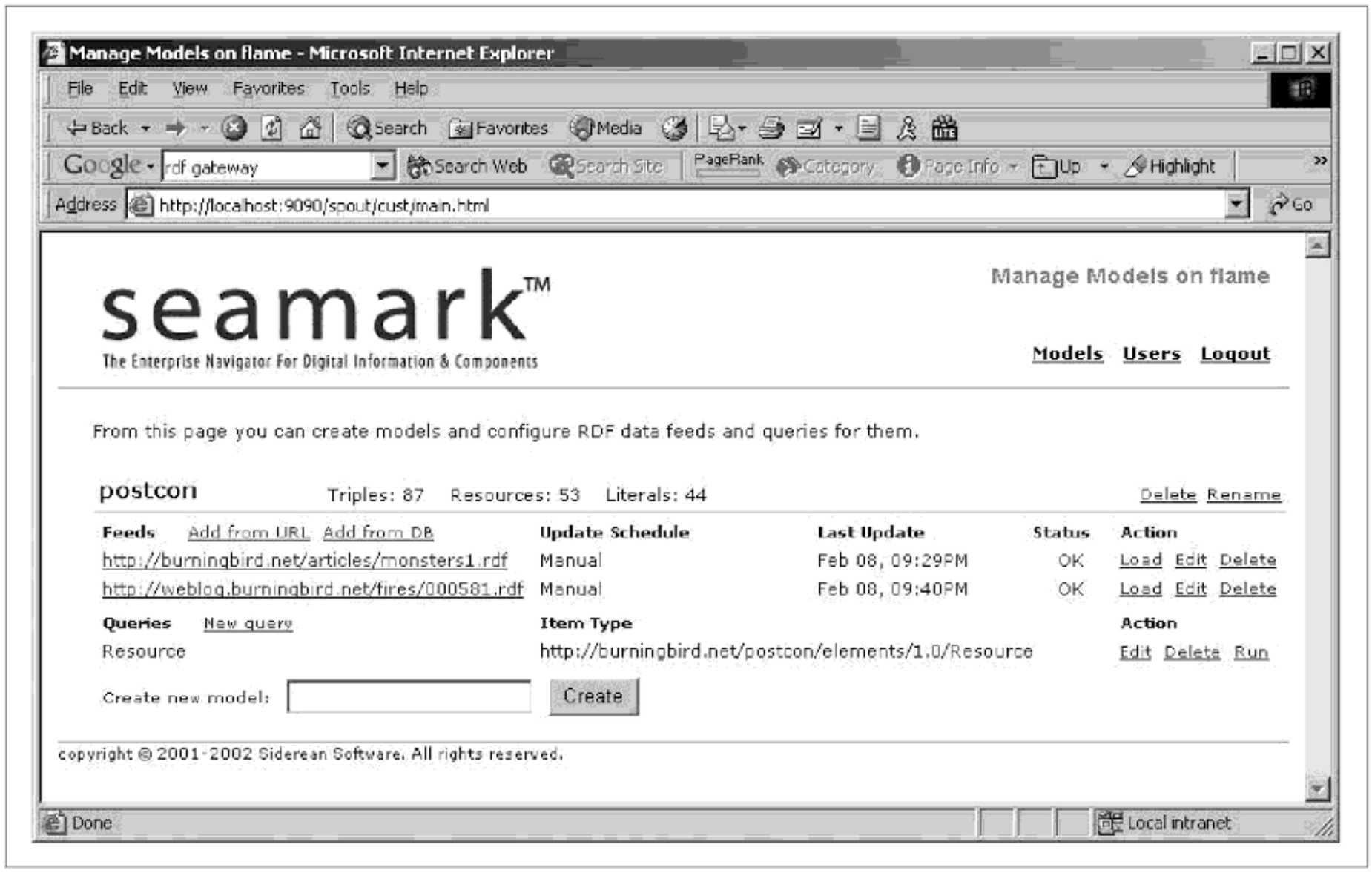


Figure 15-4. PostCon Seamark model with two data sources and one query

I was most impressed with, though, was how quickly and easily it integrated my RDF/XML data from the PostCon application into a sophisticated query engine with little or no effort. Few things prove the usefulness of a well-defined metadata structure faster than commercial viability.

Plugged In Software's Tucana Knowledge Store

Plugged In Software's Tucana Knowledge Store (TKS) enables storage and retrieval of data that's designed to efficiently scale to larger datastores. The scalability is assured because distributed data sources are an inherent part of the architecture, as is shown in the diagram in Figure 15-5.

You can download an evaluation copy of Tucana Knowledge Store at *http://www.pisoftware.com/index.html*. In addition, if you intend to use the application for academic purposes, you can download and use an academic copy of the application for free.

In situations with large amounts of potentially complex data, this distributed data repository may be the only effective approach to finding specific types of data. TKS has found a home in the defense industry because of the nature of its architecture and is being used within the intelligence as well as defense communities.

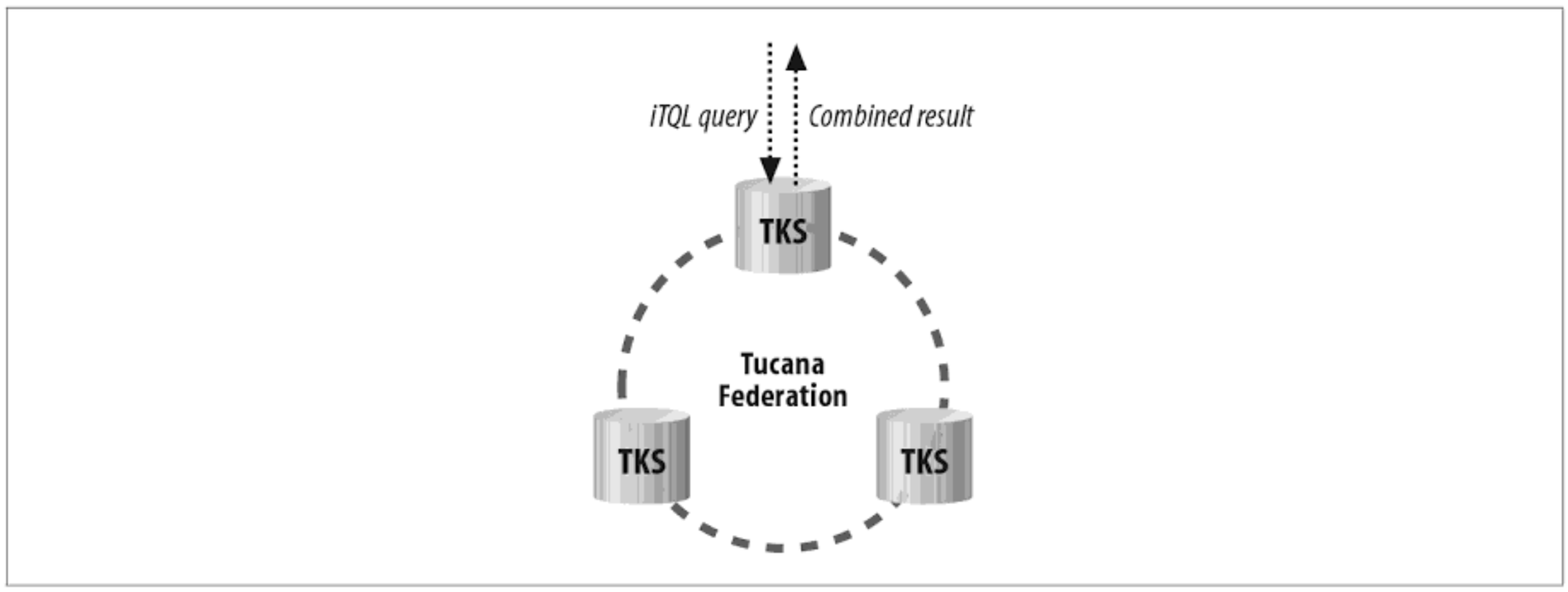


Figure 15-5. Demonstration of TKS distributed nature

TKS pairs the large-scale data storage and querying with a surprisingly simple interface. For instance, the query language support (iTQL) functionality can be accessed at the command line by typing in the following command:

```
java -jar itql-1.0.jar
```

This command opens an iTQL shell session. Once in, just type in the commands necessary. I found TKS to be as intuitively easy to use as it was to install. I followed the tutorial included with TKS, except using my example RDF/XML document, *http://burningbird.net/articles/monsters1.rdf*, as the data source. First, I created a model within TKS to hold the data:

```
iTQL> create <rmi://localhost/server1#postcon>;
Successfully created model rmi://localhost/server1#postcon
```

Next, I loaded the data from the external document:

```
iTQL> load <http://burningbird.net/articles/monsters1.rdf> into <rmi://localhost/
server1#postcon>;
Successfully loaded 58 statements from http://burningbird.net/articles/monsters1.rdf
into rmi://localhost/server1#postcon
```

After the data was loaded, I queried the two "columns" in the data—the predicate and the object—for the main resource, `http://burningbird.net/articles/monsters1.htm`:

```
iTQL> select $obj $pred from <rmi://localhost/server1#postcon> where <pstcn:release>
$pred $obj;
0 columns: (0 rows)
iTQL> select $obj $pred from <rmi://localhost/server1#postcon> where <http://
burningbird.net/articles/monsters1.htm> $pred $obj;
2 columns: obj pred (8 rows)
        obj=http://burningbird.net/articles/monsters2.htm       pred=http://burn
ingbird.net/postcon/elements/1.0/related
        obj=http://burningbird.net/articles/monsters3.htm       pred=http://burn
ingbird.net/postcon/elements/1.0/related
        obj=http://burningbird.net/articles/monsters4.htm       pred=http://burn
ingbird.net/postcon/elements/1.0/related
        obj=http://burningbird.net/postcon/elements/1.0/Resource        pred=htt
p://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

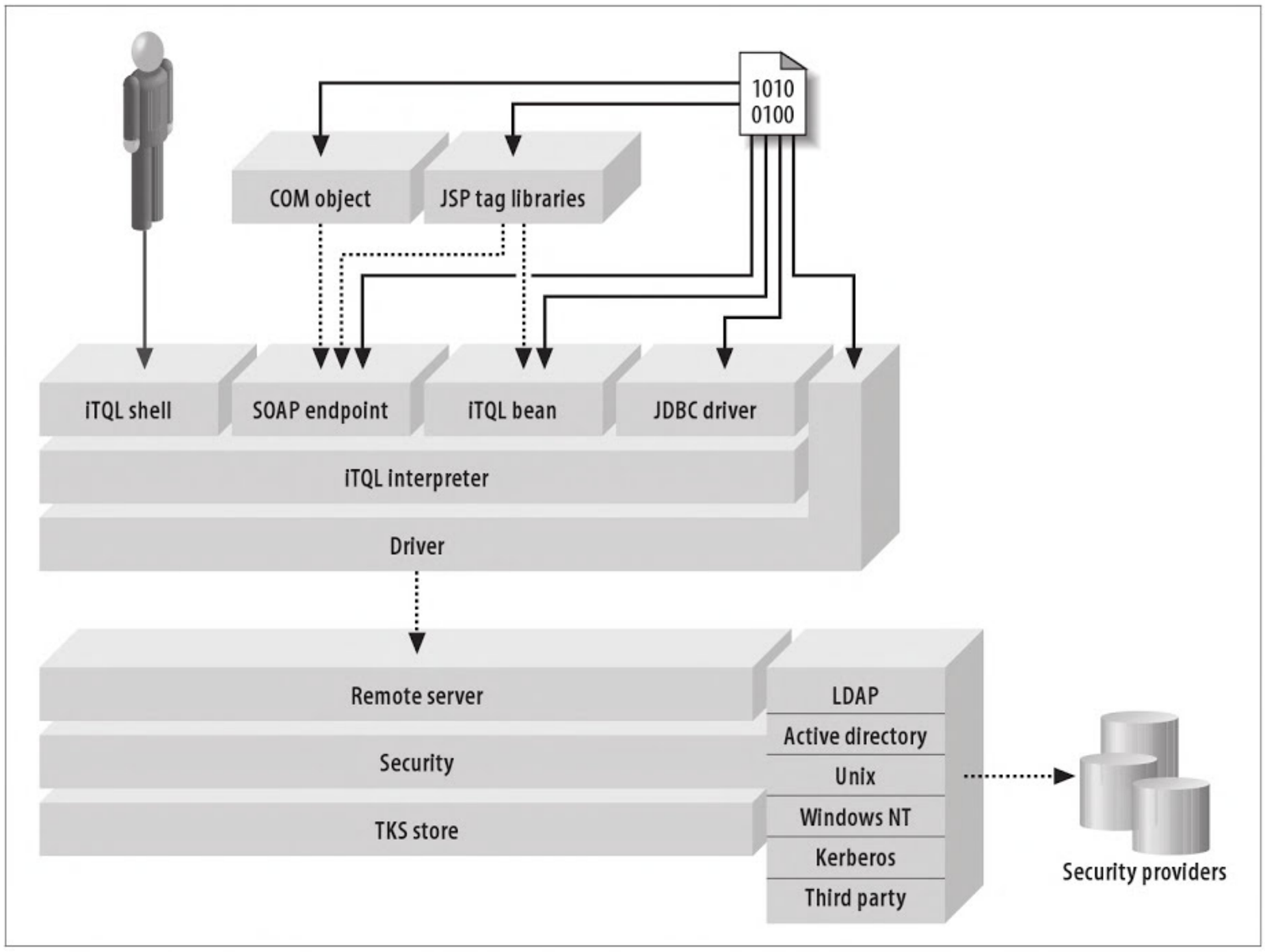


Figure 15-6. Client/Server architecture supported by TKS

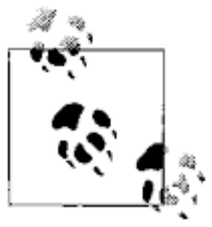
FrameMaker 7.0, GoLive 6.0, InCopy 2.0, InDesign 2.0, Illustrator 10, and LiveMotion 2.0.

The information included within the embedded labels can be from any schema as long as it's recorded in valid RDF/XML. The XMP source code is freely available for download, use, and modification under an open source license.

Read more about Adobe XMP at *http://www.adobe.com/products/xmp/*. Download the SDK at *http://partners.adobe.com/asn/developer/xmp/main.html*.

Unlike so much of the RDF/XML technology, which emphasizes Java or Python, the XMP Toolkit provides only support for C++. Specifically, the toolkit works with Microsoft's Visual C++ in Windows (or compatible compiler) and Metrowerks CodeWarrior C++ for the Mac.

Within the SDK is a subdirectory of C++ code that allows a person to read and write XMP metadata. Included in the SDK is a good set of documentation that provides samples and instructions on embedding XMP metadata into TIFF, HTML, JPEG, PNG, PDF, SVG/XML, Illustrator (*.ai*), Photoshop (*.psd*), and Postscript and EPS formats.



The SDK is a bit out of date in regard to recent activities with RDF and RDF/XML. For instance, when discussing embedded RDF/XML into HTML documents, it references a W3C note that was favorable to the idea of embedding of RDF/XML into HTML. However, as you read in Chapter 3, recent decisions discourage the embedding of metadata into (X)HTML documents, though it isn't expressly forbidden.

The SDK contains some documentation, but be forewarned, it assumes significant experience with the different data types, as well as experience working with C++. The document of most interest is the Metadata Framework PDF file, specifically the section discussing how XMP works with RDF, as well as the section on extending XMP with external RDF/XML Schemas. This involves nothing more than defining data in valid RDF and using a namespace for data not from the core schemas used by XMP. The section titled "XMP Schemas" lists all elements of XMP's built-in schemas.

The SDK also includes C++ and the necessary support files for the Metadata Library, as well as some other utilities and samples. I dusted off my rarely used Visual C++ 6.0 to access the project for the Metadata Toolkit, Windows, and was able to build the library without any problems just by accessing the project file, *XAPToolkit.dsw*. The other C++ applications also compiled cleanly as long as I remembered to add the paths for the included header files and libraries.

One of the samples included with the SDK was XAPDumper, an application that scans for embedded RDF/XML within an application or file and then prints it out. I compiled it and ran it against the *SDKOverview.pdf* document. An excerpt of the embedded data found in this file is:

```
<rdf:Description rdf:about=''
 xmlns:pdf='http://ns.adobe.com/pdf/1.3/'>
 <pdf:Producer>Acrobat Distiller 5.0.5 for Macintosh</pdf:Producer>
 <!–pdf:CreationDate is aliased–>
 <!–pdf:ModDate is aliased–>
 <!–pdf:Creator is aliased–>
 <!–pdf:Author is aliased–>
 <!–pdf:Title is aliased–>
</rdf:Description>
```

Embedding RDF/XML isn't much different than attaching a bar code to physical objects. Both RDF and bar codes uniquely identify important information about the object in case it becomes separated from an initial package. In addition, within a publications environment, if all of the files are marked with this RDF/XML-embedded information, automated processes could access this information and use it to determine how to connect the different files together, such as embedding a JPEG file into an HTML page and so on.

I can see the advantage of embedded RDF/XML for any source that's loaded to the Web. Eventually, web bots could access and use this information to provide more

intelligent information about the resources that they touch. Instead of a few keywords and a title as well as document type, these bots could provide an entire history of a document or picture, as well as every particular about it.

Other applications can also build in support for working with XMP. For instance, RDF Gateway, mentioned earlier, has the capability of reading in Adobe XMP. An example of how this application would access data from an Adobe PDF would be:

```
var monsters = new
DataSource("inet?url=http://burningbird.net/articles/monsters3.pdf&parse
type=xmp");
```

An important consideration with these embedded techniques is that there is no adverse impact on the file, nothing that impacts on the visibility of a JPEG or a PNG graphic or prevents an HTML file from loading into a browser. In fact, if you've read any PDF files from Adobe and other sites that use the newer Adobe products, you've probably been working with XMP documents containing embedded RDF/XML and didn't even know it.

What's It All Mean?

In my opinion, Adobe's use of RDF/XML demonstrates how RDF/XML will be integrated in other applications and uses in the future—quietly, behind the scenes. Unlike XML with its public exposure, huge fanfare, and claims of human and machine compatibility and interoperability, RDF was never meant to be anything more than a behind-the-scenes metadata model and an associated serialization format. RDF records statements so that they can be discovered mechanically—nothing more, nothing less. However, this simple act creates a great many uses of RDF/XML because of the careful analysis and precision that went into building the specification upon which RDF resides and which RDF/XML transcribes.

RDF assures us that any data stored in RDF/XML format in one application can be incorporated with data stored in RDF/XML format in another application, and moving the data from one to the other occurs without loss of information or integrity. While sharing and transmitting, merging and coalescing the data, we can attach meaning to objects stored on the Web—meaning that can be accessed and understood by applications and automated agents and APIs such as those covered in this book.

As the use of RDF grows, the dissemination of RDF/XML data on the Web increases and the processing of this data is incorporated into existing applications, the days when I'll search for information about the giant squid and receive information on how to cook giant squid steaks will fade into the past. I will be able to input parameters specific to my search about the giant squid into the computer and have it return exactly what I'm looking for, because the computer and I will have learned to understand each other.

This belief in the future of RDF and RDF/XML was somewhat borne out when I did a final search for information on the giant squid and its relation to the legends and to that other legendary creature, Nessie the Loch Ness Monster, as I was finishing this book. When I input the terms *giant squid legends Nessie* in Google, terms from my subject lists associated with the article that's been used for most of the examples in this book, the PostCon RDF/XML file for my giant squid article was the first item Google returned.

It's a start.

Index

Symbols

(pound sign), indicating relative URI, 44

Numbers

3-tuple representation of RDF triples, 18
4RDF, Python-based RDF API, 191

A

A Relational Model of Data for Large Shared Data Banks, 86
absolute URI, 22
addProperty method, Jena, 152
Adobe XMP (see XMP)
aggregators, RSS, 11, 268–274, 278–280
Alt (alternative) container, 60, 64, 65, 89
AmphetaDesk aggregator, 269–270
anonymous nodes (see blank nodes)
Apache, PHP support with, 183
APIs
  Drive, for C#, 215–218
  Informa RSS Library, for Java, 278
  Jena, for Java (see Jena)
  PerlRDF (see PerlRDF)
  RDF API for PHP, 183–187
  RDFLib, for Python, 187–191
  RDFStore, for Perl, 182
  Wilbur, for LISP, 218
  (see also frameworks; software)
applications based on RDF
  Chandler, 303
  Creative Commons Licenses, 295–297
  FOAF (Friend-of-a-Friend), 298–301
  MIT DSpace, 297
  Mozilla, 11, 286–294
  RDF Gateway, 304–309, 317
  Seamark, 193, 309–312
  TKS (Tucana Knowledge Store), 193, 312–314
  XMP (eXtensible Metadata Platform), 314–317
  (see also software)
arcs, in RDF graph, 20
ARP2 (Another RDF/XML Parser, second generation), 136–138
  (see also Jena)
autodiscovery
  of FOAF file, 299–300
  of RSS file, 268

B

Bag container, 58, 64, 65, 89
base document, resolving relative URIs with, 44
Beckett, Dave (Expressing Simple Dublin Core in RDF/XML), 122
Berkeley Database
  persisting RDF data to, with PerlRDF, 174
  Redland using, 222
Berners-Lee, Tim
  describing uses of Semantic Web, 2
  views on the term context, 16
blank nodes, 20, 41–43
  identifiers generated for, 42
  merging, 26
  none, indicating grounded graph, 26
BlankNode class, RDF API for PHP, 183

We'd like to hear your suggestions for improving our indexes. Send email to *index@oreilly.com*.

bnodes (see blank nodes)
books
  about Mozilla, 288
  about RSS, 254
  (see also specifications and documents)
Boswell, David (Creating Applications with Mozilla), 288
Brickley, Dan
  creator of RubyRDF, 218
  editor, RDF Schema specification, 2
  Expressing Simple Dublin Core in RDF/XML, 122
  RDF: Understanding the Striped RDF/XML Syntax, 35
BrownSauce, RDF/XML browser, 132–135
browser (BrownSauce), 132–135
business model (see ontology)

C

C#, API for (see Drive)
C, API for (see Redland)
CC licenses (see Creative Commons Licenses)
Chandler, 303
channel element, RSS, 257
circles, in RDF graph, 20
classes
  Drive, 216
  Jena, 150, 154–157
  OWL, 244–247
  PHP XML, 203–210
  RDF API for PHP, 183
  RDF Schema, 87–90, 118
  Redland, 222
CLOS, API for (see Wilbur)
CNRI Handle System, 297
Codd, E. F. (A Relational Model of Data for Large Shared Data Banks), 86
codepiction project, FOAF used by, 301
collections, 65
  compared to containers, 114
  history of, 57
Collins, Pete (Creating Applications with Mozilla), 288
complement classes, OWL, 246
constraints, RDF Schema, 95
containers, 57–65
  Alt (alternative), 60, 64, 65
  alternative to, 61
  Bag, 58, 64, 65
  compared to collections, 114
  creating with Jena, 161–164
  current specifications for, 64–65
  distributive referents of, 61
  history of, 57
  initial specifications for, 58–61
  PostCon example using, 114–116
  referents (items) of, 61
  semantics of, 66
  Seq (sequence), 59, 63, 64, 65
  typed node specification revisions for, 62–64
Content module, RSS, 265
Content Syndication with XML and RSS, 254
context
  meaning of, within RDF, 16
  searching Internet based on, 15
core modules, RSS, 264–267
Creating Applications with Mozilla, 288
Creative Commons Licenses, 295–297
CSS, RDF compared to, 7

D

DAML (DARPA Agent Markup Language), 8, 229
daml:Class element, compared to rdfs:Class, 230
DAMLModelImpl class, Jena, 150
DAML+OIL, 8
  compared to OWL, 234
  compared to RDFS, 230
  converting to OWL, 231
  history of, 229
  Jena support for, 151
  specifications for, 231
DARPA Agent Markup Language (see DAML)
data handshaking, 9
data types, 53
  built-in, XML Schema, 54
  of literal nodes, 20
  of literal predicates, 53
database (see relational database)
DataSource object, RDF Gateway, 304
DB_File object, PerlRDF, 174
DC (see Dublin Core Metadata Initiative)
DC-dot generator, 129
description element, RSS, 257
directed graph (see RDF graph)
disjoint classes, OWL, 247
distributive referents, 61
Document Type Definition (see DTD)
documents (see specifications and documents)

Domfest, Rael (creator of Meerkat), 271
Drive, 215–218
DSpace (see MIT DSpace)
DTD (Document Type Definition), 97
  compared to RDF vocabulary, 101
  using with RDF/XML, 12
Dublin Core Element Set, Version 1.1, 267
Dublin Core Metadata Initiative, 120
  compared to RDF, 120
  Creative Commons Licenses using, 296
  DC-dot generator for, 129
  Element set for, 121, 123
  elements, mixing with RDF vocabulary, 124–129
  Jena classes for, 154
  MIT DSpace using, 297
  qualifiers for, 123
  RDF/XML implementation of, 122
  RSS module for, 266
Dublin Core module, RSS, 266
Dumbill, Edd (creator of FOAFBot), 301

E

editors
  IsaViz, 142–146
  Protégé, 248–252
  RDF Editor in Java, 146–148
  SMORE, 248
Eisenzopf, Jonathan (creator of XML::RSS), 281
embedded RDF, 54–56
encoding element, RSS Content module, 266
entailment, 27
enumeration classes, OWL, 246
Expressing Simple Dublin Core in RDF/XML, 122
extended modules, RSS, 267
eXtensible Metadata Platform (see XMP)
eXtensible User interface Language (see XUL)

F

faceted metadata, Seamark, 309
Feature Synopsis for OWL, 235
find method, RDF API for PHP, 187
findRegex method, RDF API for PHP, 187
findVocabulary method, RDF API for PHP, 187
FOAF (Friend-of-a-Friend), 298–301
FOAF-A-Matic, 299–300
FOAFBot, 301
FOAFCorp project, 301
format element, RSS Content module, 266
4RDF, Python-based RDF API, 191
Fourthought 4RDF, 191
frameworks
  Redfoot, 225–227
  Redland, 218–225
  (see also APIs)
Friend-of-a-Friend (see FOAF)

G

get_rdf_document method, PHP XML, 208
Ginger Alliance PerlRDF (see PerlRDF)
Globally Unique Identifier (see GUID)
graph, directed (see RDF graph)
graphics, output by IsaViz, 146
grounded RDF graph, 26
Gruber, Tom (paper about ontologies), 228
Guha, R. V.
  creator of rdfDB, 193
  editor of RDF Schema specification, 2
GUID (Globally Unique Identifier), 93

H

Hammersley, Ben (Content Syndication with XML and RSS), 254
handshaking (see data handshaking)
Haystack project, 295
hearsay, reification compared to, 67
Hemenway, Kevin (creator of AmphetaDesk), 269
HTML
  embedding RDF in, 54–56
  generating RDF/XML from (see DC-dot generator)
Hypercode scripting language, Redfoot, 225

I

ICS-FORTH Validating RDF Parser (see VRP)
Iff, Morbus (see Hemenway, Kevin)
image element, RSS, 261
Informa RSS Library, 278
Inkling database, 194–197
instance lemma, 27
instance of RDF graph, 27
Internet, searching, 15
interpolation lemma, 28
intersection of classes, OWL, 244