CHAPMAN & HALL/CRC
TEXTBOOKS IN COMPUTING

Foundations of Semantic Web Technologies

Pascal Hitzler
Markus Krötzsch
Sebastian Rudolph

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

CHAPMAN & HALL/CRC
TEXTBOOKS IN COMPUTING

Foundations of Semantic Web Technologies

Pascal Hitzler
Markus Krötzsch
Sebastian Rudolph

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group an **informa** business

A CHAPMAN & HALL BOOK

Chapman & Hall/CRC
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Chapman & Hall/CRC is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number: 978-1-4200-9050-5 (Hardback)

Library of Congress Cataloging-in-Publication Data

Hitzler, Pascal.
Foundations of Semantic Web technologies / Pascal Hitzler, Sebastian Rudolph, Markus Krötzsch.
p. cm. -- (Chapman & Hall/CRC textbooks in computing)
Includes bibliographical references and index.
ISBN 978-1-4200-9050-5 (hardcover : alk. paper)
1. Semantic Web. I. Rudolph, Sebastian, Dr. II. Krötzsch, Markus. III. Title.

TK5105.88815.H57 2009
025.042'7--dc22 2009024576

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

List of Figures

Chapter 1

The Quest for Semantics

In this chapter, we explain the basic motivations and ideas underlying Semantic Web technologies as presented in this book, along with some of the history of these ideas. This gentle introduction prepares the stage for the more technical parts that are covered in subsequent chapters.

From its very beginnings, the development of Semantic Web technologies has been closely related to the World Wide Web. This is is not surprising, given that the inventor of the WWW – Sir Tim Berners-Lee – has originally coined the term "Semantic Web" and has inspired much research in this area. Important goals of the approaches that are described in this book are indeed very similar to the goals of the Web in general: to make knowledge widely accessible and to increase the utility of this knowledge by enabling advanced applications for searching, browsing, and evaluation. And, similar to the traditional Web, the foundation of Semantic Web technologies are data formats that can be used to encode knowledge for processing (relevant aspects of it) in computer systems, although the focus is on different forms of knowledge.

However, viewing the WWW as the only origin and inspiration for the technologies that are described in this book would not do justice to their true history. More importantly, it would also hide some of the main motivations that have led to the technologies in their present form. To avoid such a narrow perspective in this chapter, two further strands of closely related endeavors are explained here. One is the general approach of *building abstract models* that capture the complexities of the world in terms of simpler ideas. Modeling in this sense pervades human history – a comprehensive historic account is beyond the scope of this book – but underlying methods and motivations are highly relevant for the semantic technologies that are available for us today.

A second, more recent approach is the idea of *computing with knowledge*. The vision of representing knowledge in a way that allows machines to automatically come to reasonable conclusions, maybe even to "think," has been a driving force for decades of research and development, long before the WWW was imagined. Again, a brief look at this line of development helps us to understand some of the motivations and ideas behind the technologies presented in this book. Thus we arrive at the following three main topics that provide conceptual underpinnings for the Semantic Web:

- Building models: the quest for describing the world in abstract terms to allow for an easier understanding of a complex reality.

- Computing with knowledge: the endeavor of constructing reasoning machines that can draw meaningful conclusions from encoded knowledge.

- Exchanging information: the transmission of complex information resources among computers that allows us to distribute, interlink, and reconcile knowledge on a global scale.

Within this introductory chapter, we briefly outline the ideas underlying each of these topic areas within Sections 1.1, 1.2, and 1.3. Thereafter, in Section 1.4, we explain how these ideas come together in what is often described as *Semantic Web technologies* today. As in every chapter, we conclude with a brief guide to useful references for further reading that is given in Section 1.5.

1.1 Building Models

Generally speaking, a model is a simplified description of certain aspects of reality, used for understanding, structuring, or predicting parts of the real world. In a most general sense, forming models of the world is part of our very ability to reason and to communicate. In this section, we are interested in *scientific modeling*, and especially in those developments that influence today's semantic technologies. Numerical models, such as those described by physical formulae, are less relevant for our considerations and will not be discussed in detail.

Beginnings of scientific modeling can be traced back to ancient philosophy. The Greek philosopher Plato (429–347 BC) proposed answers to some of the most fundamental questions that arise during modeling: What is reality? Which things can be said to "exist"? What is the true nature of things? This marks the first major contribution to a philosophical field now known as *ontology* – the study of existence and being *as such*, and of the fundamental classes and relationships of existing things. Interestingly, the term "ontology" has become very important in today's semantic technologies, but with a rather different meaning: in computer science, an ontology is a description of knowledge about a domain of interest, the core of which is a machine-processable specification with a formally defined meaning.[1]

Ontology in the philosophical sense was further advanced by Plato's student Aristotle (384–322 BC). In contrast to his teacher, Aristotle held the view that models are not given as universal ideas that are merely reflected by reality, but rather that they should be *derived* from careful observations of reality – a view that had great influence on the development of science

[1] Ontology is not the first field of study to experience such terminological (ab)use, as is apparent when speaking of "Alaska's impressive geography" or "the biology of Dragonflies."

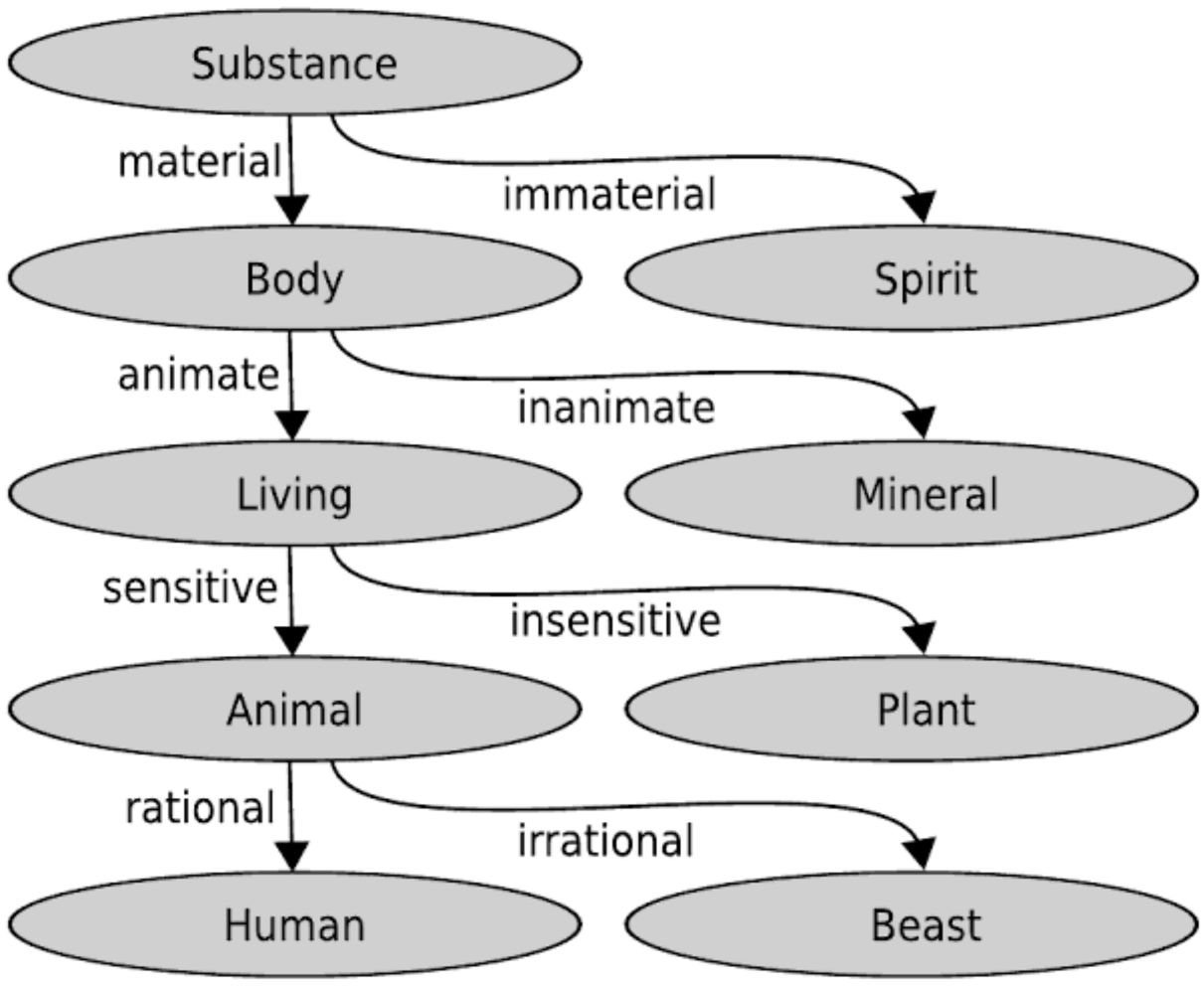

FIGURE 1.1: The *Tree of Porphyry*, an early tree structure in knowledge modeling; diagram based on a translation in [Sow00]

in centuries to come. Applying this approach, Aristotle developed ten categories to classify all things that may exist, and he described subcategories to further specify each of them. For example, Aristotle's category of animals is composed of rational ones (humans) and irrational ones (beasts). Typical for traditional classifications, subcategories in Aristotle's model are exhaustive, i.e. each thing in a category belongs to one of its subcategories, and mutually exclusive, i.e. each thing in a category can belong only to one of its subcategories.

These early approaches toward scientific classification also introduce the use of *structure* in modeling. The philosopher Porphyry (circa 234–305) developed the Tree of Porphyry, a small tree-shaped model that captures the hierarchical relationships of some of Aristotle's categories in a graphical form (see Fig. 1.1). Tree structures, concept hierarchies, and inheritance of properties are notions that are essential for numerous modeling tasks, and that are still found in many applications today.

Numerous influential scientific models have been developed in later centuries, often building upon the basic idea of classification that is found in the works of Aristotle. Carolus Linnaeus (1707–1778) laid the basis for modern biological classification by introducing *Linnaean taxonomy* as a means to classify all life forms. The term *taxonomy* – composed of the Greek words *taxis* (order) and *nomos* (law, science) – has since become the name of the science of classification, but it is also used to refer to individual hierarchical classification schemes. Other important classifications include the WHO's International Classification of Diseases (ICD), the Köppen classification of cli-

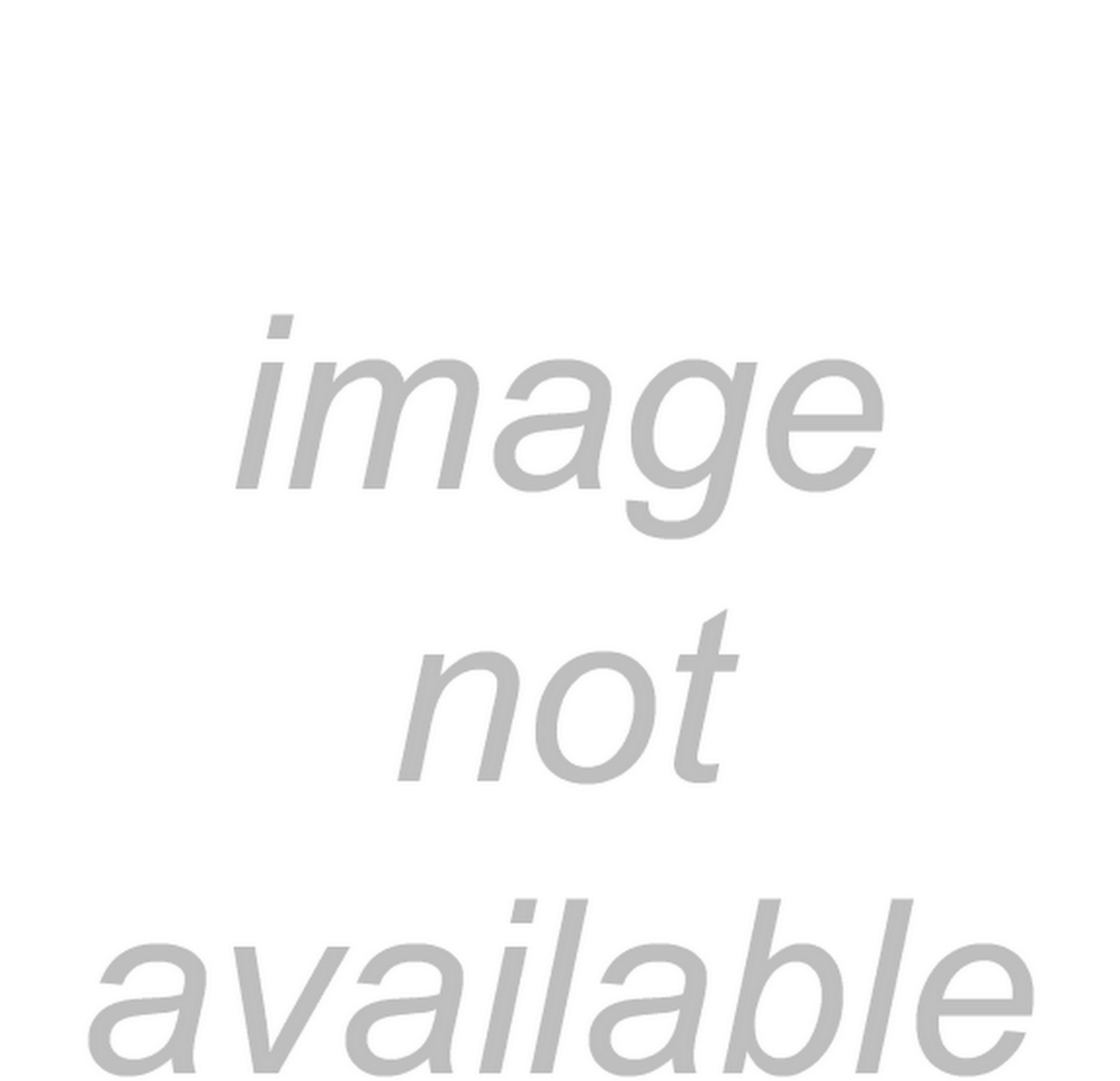
image
not
available

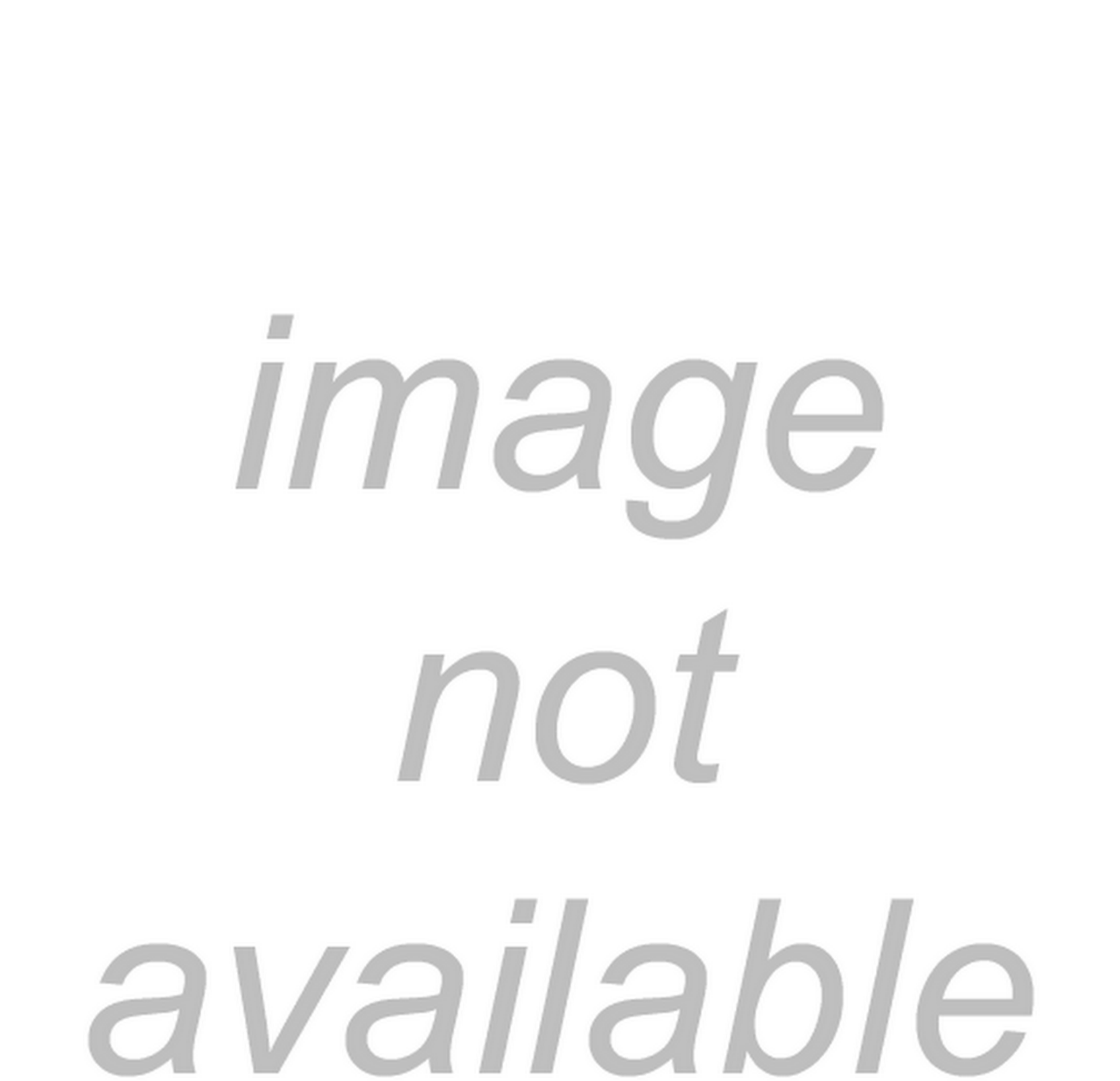
image
not
available

Part I

Resource Description Language RDF

topic, although Tim O'Reilly registered the term as a trademark and published various characterizations at `http://oreilly.com/`.

The seminal 2001 article about the Semantic Web is [BLHL01], and an update is [SBLH06]. The central website on the W3C Semantic Web activity is `http://www.w3.org/2001/sw/` and includes many pointers to further online resources. A research-oriented account of recent semantic technologies is given in [SS09]. For reading up on the most recent results concerning Semantic Web research and practice, we recommend the following sources.

- the Elsevier Journal of Web Semantics: Science, Services and Agents on the World Wide Web,[13]
- the IGI Global International Journal on Semantic Web and Information Systems,[14]
- the proceedings of the annual International Semantic Web Conferences (ISWC),[15]
- the proceedings of the annual International World Wide Web Conferences,[16]
- the proceedings of the annual Semantic Technology Conferences.[17]

Specific references to the mentioned Semantic Web standards are given in the respective chapters. This excludes two standards that have been mentioned but are not treated within this book: GRDDL ("Gleaning Resource Descriptions from Dialects of Languages") which is specified in [Con07], and RDFa (embedding RDF into XHTML) which is specified in [ABMP08]. Documentation about microformats can be found at `http://microformats.org/`.

[13] `http://www.websemanticsjournal.org/`
[14] http://www.ijswis.org/
[15] `http://iswc.semanticweb.org/`
[16] `http://www.iw3c2.org/.`
[17] `http://www.semantic-conference.com/`

Chapter 2

Simple Ontologies in RDF and RDF Schema

The *Resource Description Framework* RDF is a formal language for describing structured information. The goal of RDF is to enable applications to exchange data on the Web while still preserving their original meaning. As opposed to HTML and XML, the main intention now is not to display documents correctly, but rather to allow for further processing and re-combination of the information contained in them. RDF consequently is often viewed as the basic representation format for developing the Semantic Web.

The development of RDF began in the 1990s, and various predecessor languages have influenced the creation process of RDF. A first official specification was published in 1999 by the W3C, though the emphasis at this time still was clearly on the representation of *metadata* about Web resources. The term metadata generally refers to data providing information about given data sets or documents. In 1999, the latter were mainly expected to be Web pages, for which RDF could help to state information on authorship or copyright. Later the vision of the Semantic Web was extended to the representation of semantic information in general, reaching beyond simple RDF data as well as Web documents as primary subjects of such descriptions. This was the motivation for publishing a reworked and extended RDF specification in 2004.

As of today, numerous practical tools are available for dealing with RDF. Virtually every programming language offers libraries for reading and writing RDF documents. Various RDF stores – also called *triple stores* for reasons that shall become clear soon – are available for keeping and processing large amounts of RDF data, and even commercial database vendors are already providing suitable extensions for their products. RDF is also used to exchange (meta) data in specific application areas. The most prominent example of this kind of usage is likely to be RSS 1.0 for syndicating news on the Web.[1] But also metadata belonging to files of desktop applications are sometimes encoded using RDF, such as in the case of Adobe's RDF format XMP for embedding information in PDF files, or as annotations in the XML-based vector graphics format SVG. We will say more about such applications in Chapter 9.

[1] RSS 1.0 and 2.0 are different formats, which pursue the same goal but which, confusingly, are not based on each other. RSS 1.0 stands for *RDF Site Summary*, whereas RSS 2.0 is usually interpreted as *Really Simple Syndication*. See also Section 9.1.2.

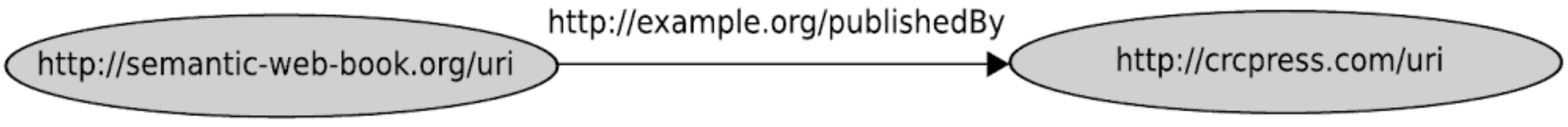

FIGURE 2.1: A simple RDF graph describing the relationship between this book and the publisher, CRC Press

This chapter introduces the basics of RDF. Initially, the representation of simple data is our main concern. In the subsequent sections, we have a closer look at the various syntactic formats available for exchanging RDF, and we address some further questions regarding the usage of RDF. Thereafter we consider some specific expressive features that go beyond the description of simple data. RDF is extended to the language *RDF Schema* (RDFS) for this purpose, allowing us to express also general information about a data set. The official formal semantics as used for properly interpreting RDF and RDFS in computer programs is explained in detail in Chapter 3.

2.1 Introduction to RDF

We begin by giving a very basic introduction to the RDF format that also highlights major differences to XML. As we shall see, RDF is based on a very simple graph-oriented data schema.

2.1.1 Graphs Instead of Trees

An RDF document describes a *directed graph*, i.e. a set of *nodes* that are linked by *directed edges* ("arrows"). Both nodes and edges are labeled with identifiers to distinguish them. Figure 2.1 shows a simple example of a graph of two nodes and one edge. In contrast, as recalled in Appendix A, information in XML is encoded in tree structures. Trees are perfectly suited for organizing information in electronic documents, where we are often confronted with strictly hierarchical structures. In addition, information in trees can often be fetched directly and be processed rather efficiently. Why then is RDF relying on graphs?

An important reason is that RDF was not conceived for the task of structuring documents, but rather for describing general relationships between objects of interest (in RDF one usually speaks of "resources"). The graph in Fig. 2.1, e.g., might be used to express that the book "Foundations of Semantic Web Technologies" was published by "CRC Press" if we interpret the given labels to refer to those objects. The relationship between book and publishing house in this case is information which does not in any obvious sense belong hierar-

chically below either of the resources. RDF therefore considers such relations as basic building blocks of information. Many such relationships together naturally form graphs, not hierarchical tree structures.

Another reason for choosing graphs is the fact that RDF was intended to serve as a description language for data on the WWW and other electronic networks. Information in these environments is typically stored and managed in decentralized ways, and indeed it is very easy to combine RDF data from multiple sources. For example, the RDF graphs from the website of this book could simply be joined with graphs from `http://semanticweb.org` – this would merely lead to a bigger graph that may or may not provide interesting new information. Note that we generally allow for graphs to consist of multiple unconnected components, i.e. of sub-graphs without any edges between them. Now it is easy to see why such a straightforward approach would not be feasible for combining multiple XML documents. An immediate problem is that the simple union of two tree structures is not a tree anymore, so that additional choices must be made to even obtain a well-formed XML document when combining multiple inputs. Moreover, related information items in trees might be separated by the strict structure: even if two XML files refer to the same resources, related information is likely to be found in very different locations in each tree. Graphs in RDF are therefore better suited for the composition of distributed information sources.

Note that these observations refer to the *semantic* way in which RDF structures information, not to the question of how to encode RDF data *syntactically*. We will see below that XML is still very useful for the latter purpose.

2.1.2 Names in RDF: URIs

We have claimed above that RDF graphs enable the simple composition of distributed data. This statement so far refers only to the graph structure in general, but not necessarily to the intended information in the composed graphs. An essential problem is that resources, just like in XML, may not have uniform identifiers within different RDF documents. Even when two documents contain information on related topics, the identifiers they use might be completely unrelated. On the one hand, it may happen that the same resource is labeled with different identifiers, for instance, since there is no globally agreed identifier for the book "Foundations of Semantic Web Technologies." On the other hand, it may occur that the same identifiers are used for different resources, e.g., "CRC" could refer to the publishing house as well as to the official currency of Puerto Rico. Such ambiguity would obviously be a major problem when trying to process and compose information automatically.

To solve the latter problem, RDF uses so-called *Uniform Resource Identifiers* (*URIs*) as names to clearly distinguish resources from each other. URIs are a generalization of URLs (Uniform Resource Locators), i.e. of Web addresses as they are used for accessing online documents. Every URL is also a

valid URI, and URLs can indeed be used as identifiers in RDF documents that talk about Web resources. In numerous other applications, however, the goal is not to exchange information about Web pages but about many different kinds of objects. In general this might be any object that has a clear identity in the context of the given application: books, places, people, publishing houses, events, relationships among such things, all kinds of abstract concepts, and many more. Such resources can obviously not be retrieved online and hence their URIs are used exclusively for unique identification. URIs that are not URLs are sometimes also called *Uniform Resource Names* (URNs).

Even if URIs can refer to resources that are not located on the Web, they are still based on a similar construction scheme as common Web addresses. Figure 2.2 gives an overview of the construction of URIs, and explains their relevant parts. The main characteristic of any URI is its initial scheme part. While schemes like `http` are typically associated with a protocol for transmitting information, we also find such schemes in many URIs that do not refer to an actual Web location. The details of the protocol are obviously not relevant when using a URI only as a name. The book "Foundations of Semantic Web Technologies" could, e.g., use the URI `http://semantic-web-book.org/uri` and it would not matter whether or not a document can be retrieved at the corresponding location, and whether this document is relevant in the given context. As we shall see later on, RDF makes use of various mechanisms of XML to abbreviate URIs when convenient.

As shown in Fig. 2.1, nodes and edges in RDF graphs both are labeled with URIs to distinguish them from other resources. This rule has two possible exceptions: RDF allows for the encoding of data values which are not URIs, and it features so-called *blank nodes* which do not carry any name. We will take a closer look at both cases next. Later we will also return to the question of finding good URIs in practice, in a way that ensures maximal utility and reliability in semantic applications. For now we are satisfied with the insight that URIs, if they are well-chosen, provide us with a robust mechanism for distinguishing different entities, thus avoiding confusion when combining RDF data from distributed sources.

2.1.3 Data Values in RDF: Literals

URIs allow us to name abstract resources, even those that cannot be represented or processed directly by a computer. URIs in this case are merely references to the intended objects (people, books, publishers, ...). While URIs can always be treated as names, the actual "intended" interpretation of particular URIs is not given in any formal way, and specific tools may have their own way of interpreting certain URIs. A certain Web Service, e.g., may recognize URIs that refer to books and treat them in a special way by displaying purchasing options or current prices. This degree of freedom is useful and in fact unavoidable when dealing with arbitrary resources. The situation is different when dealing with concrete data values such as numbers, times,

The general construction scheme of URIs is summarized below, where parts in brackets are optional:

scheme`:[//`*authority*`]`*path*`[?`*query*`][#`*fragment*`]`

The meaning of the various URI parts is as follows:

scheme The name of a URI scheme that classifies the type of URI. Schemes may also provide additional information on how to handle URIs in applications. Examples: `http`, `ftp`, `mailto`, `file`, `irc`

authority URIs of some URI schemes refer to "authorities" for structuring the available identifiers further. On the Web, this is typically a domain name, possibly with additional user and port details. The authority part of a URI is optional and can be recognized by the preceding `//`. Examples: `semantic-web-book.org`, `john@example.com`, `example.org:8080`

path The path is the main part of many URIs, though it is possible to use empty paths, e.g., in email addresses. Paths can be organized hierarchically using `/` as separator. Examples: `/etc/passwd`, `this/path/with/-:_~/is/../okay` (paths without initial `/` are only allowed if no authority is given)

query The query is an optional part of the URI that provides additional non-hierarchical information. It can be recognized by its preceding `?`. In URLs, queries are typically used for providing parameters, e.g., to a Web Service. Example: `q=Semantic+Web+book`

fragment The optional fragment part provides a second level of identifying resources, and its presence is recognized by the preceding `#`. In URLs, fragments are often used to address a sub-part of a retrieved resource, such as a section in an HTML file. URIs with different fragments are still different names for the purpose of RDF, even if they may lead to the same document being retrieved in a browser. Example: `section1`

Not all characters are allowed in all positions of a URI, and illegal symbols are sometimes encoded by specific means. For the purpose of this book it suffices to know that basic Latin letters and numbers are allowed in almost any position. Moreover, the use of non-Latin characters that abound in many languages is widely allowed in all current Semantic Web formats as well. URIs that are extended in this way are known as *International Resource Identifiers* (IRIs), and they can be used in any place where URIs are considered in this book.

FIGURE 2.2: The basic construction scheme for URIs

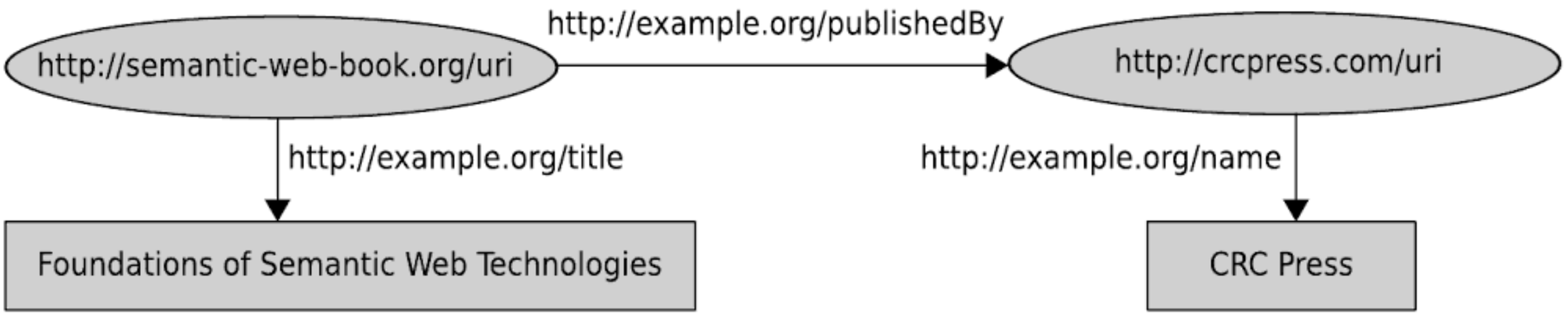

FIGURE 2.3: An RDF graph with literals for describing data values

or truth values: in these cases, we would expect every application to have a minimal understanding of the concrete meaning of such values. The number 42, e.g., has the same numeric interpretation in any context.

Data values in RDF are represented by so-called *literals*. These are reserved names for RDF resources of a certain datatype. The value of every literal is generally described by a sequence of characters, such as the string consisting of the symbols "4" and "2" in the above example. The interpretation of such sequences is then determined based on a given *datatype*. Knowing the datatype is crucial for understanding the intended meaning: the character sequences "42" and "042", e.g., refer to the same natural number but to different text strings.

For the time being, we will consider only literals for which no datatype has been given. Such *untyped* literals are always interpreted as text strings. The slightly more complex form of literals that contains an additional datatype identifier will be explained later on.

As can be seen in Fig. 2.3, rectangular boxes are used to distinguish literals from URIs when drawing RDF graphs. Another special trait of literals is that they may never be the origin of edges in an RDF graph. In practice, this means that we cannot make direct statements about literals.[2] This constraint needs to be taken into account when modeling data in RDF. Moreover, it is not allowed to use literals as labels for edges in RDF graphs – a minor restriction since it is hard to see what could be intended with such a labeling. Note that it is still allowed to use the same URI for labeling both nodes and edges in a graph, so at least in RDF there is no principle separation between resources used for either purpose.

[2]The reason for this restriction is in fact historic, and an official resolution of the RDF-Core working group notes that it could be waived in future Semantic Web languages; see `http://www.w3.org/2000/03/rdf-tracking/#rdfms-literalsubjects`.

2.2 Syntax for RDF

Up to this point, we have described RDF graphs by means of drawing diagrams. This way of representing RDF is easy to read and still precise, yet it is clearly not suitable for processing RDF in computer systems. Even for humans, understanding visual graphs works without much effort only if the graphs are very small – practically relevant data sets with thousands or millions of nodes do obviously not lend themselves to being stored and communicated in pictures. This section thus introduces ways of representing RDF by means of character strings that can easily be kept in electronic documents. This requires us to split the original graph into smaller parts that can be stored one by one. Such a transformation of complex data structures into linear strings is called *serialization*.

2.2.1 From Graphs to Triples

Computer science has various common ways of representing graphs as character strings, e.g., by using an adjacency matrix. RDF graphs, however, are typically very sparse graphs within which the vast majority of possible relationships do not hold. In such a case it makes sense to represent the graph as the set of edges that are actually given, and to store each edge on its own. In the example of Fig. 2.1 this is exactly one edge, uniquely determined by its start `http://semantic-web-book.org/uri`, label `http://example.org/publishedBy`, and endpoint `http://crcpress.com/uri`. Those three distinguished parts are called *subject*, *predicate*, and *object*, respectively.

It is easy to see that every RDF graph can, in essence, be completely described by its edges. There are of course many ways for drawing such graphs, but the details of the visual layout clearly have no effect on the information the graph conveys. Now every such edge corresponds to an *RDF triple* "subject-predicate-object." As we have seen above, each part of a triple can be a URI, though the object might also be a literal. Another special case is *blank nodes* that we will consider later.

2.2.2 Simple Triple Syntax: N3, N-Triples and Turtle

Our earlier observations suggest that one denotes RDF graphs simply as a collection of all their triples, given in arbitrary order. This basic idea has indeed been taken up in various concrete proposals for serializing RDF. A realization that dates back to 1998 is Tim Berners-Lee's *Notation 3* (N3), which also includes some more complex expressions such as paths and rules. The RDF recommendation of 2004 therefore proposed a less complicated part of N3 under the name *N-Triples* as a possible syntax for RDF. N-triples in

turn was further extended to incorporate various convenient abbreviations, leading to the RDF syntax *Turtle* which is hitherto not described in an official standardization document. Both N-Triple and Turtle are essentially parts of N3, restricted to covering only valid RDF graphs. Here we consider the more modern Turtle syntax.

The graph of Fig. 2.3 is written in Turtle as follows:

```
<http://semantic-web-book.org/uri>
    <http://example.org/publishedBy>  <http://crcpress.com/uri> .
<http://semantic-web-book.org/uri>
    <http://example.org/title>
        "Foundations of Semantic Web Technologies" .
<http://crcpress.com/uri>
    <http://example.org/name>         "CRC Press" .
```

URIs are thus written in angular brackets, literals are written in quotation marks, and every statement is terminated by a full stop. Besides those specific characteristics, however, the syntax is a direct translation of the RDF graph into triples. Spaces and line breaks are only relevant if used within URIs or literals, and are ignored otherwise. Our lengthy names force us to spread single triples over multiple lines. Due to the hierarchical structure of URIs, the identifiers in RDF documents typically use similar prefixes. Turtle offers a mechanism for abbreviating such URIs using so-called *namespaces*. The previous example can be written as follows:

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri .
book:uri  ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press" .
```

URIs are now abbreviated using prefixes of the form "prefix:" and are no longer enclosed in angular brackets. Without the latter modification it would be possible to confuse the abbreviated forms with full URIs, e.g., since it is allowable to use a prefix "http:" in namespace declarations. The prefix text that is used for abbreviating a particular URI part can be chosen freely, but it is recommended to select abbreviations that are easy to read and that refer the human reader to what they abbreviate. Identifiers of the form "prefix:name" are also known as *QNames* (for *qualified names*).

It frequently happens that RDF descriptions contain many triples with the same subject, or even with the same subject and predicate. For those common cases, Turtle provides further shortcuts as shown in the following example:

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri ;
          ex:title        "Foundations of Semantic Web Technologies" .
crc:uri   ex:name         "CRC Press", "CRC" .
```

The semicolon after the first line terminates the triple, and at the same time fixes the subject `book:uri` for the next triple. This allows us to write many triples for one subject without repeating the name of the subject. The comma in the last line similarly finishes the triple, but this time both subject and predicate are re-used for the next triple. Hence the final line in fact specifies two triples providing two different names. The overall RDF graph therefore consists of four edges and four nodes. It is possible to combine semicolon and comma, as shown in the next example with four triples:

```
@prefix book: <http://semantic-web-book.org/> .
@prefix ex: <http://example.org/> .

book:uri  ex:author book:Hitzler, book:Krötzsch, book:Rudolph ;
          ex:title  "Foundations of Semantic Web Technologies" .
```

The above abbreviations are not contained in the official (normative) W3C syntax N-Triples which allows neither namespaces, nor comma or semicolon. Yet, Turtle's syntactic shortcuts are frequently encountered in practice, and they have influenced the triple syntax of W3C's more recent SPARQL specification, introduced in Chapter 7.

2.2.3 The XML Serialization of RDF

The Turtle representation of RDF can easily be processed by machines but is still accessible for humans with relatively little effort. Yet, triple representations like Turtle are by far not the most commonly used RDF syntax in practice. One reason for this might be that many programming languages do not offer standard libraries for processing Turtle syntax, thus requiring developers to write their own tools for reading and writing to files. In contrast, essentially every programming language offers libraries for processing XML files, so that application developers can build on existing solutions for storage and pre-processing. As of today, the main syntax for RDF therefore is the XML-based serialization RDF/XML that is introduced in this section. This syntax also offers a number of additional features and abbreviations that can be convenient to represent advanced features which we will encounter later

on, but at the same time it imposes some additional technical restrictions. Readers who are not familiar with the basics of XML may wish to consult Appendix A for a quick introduction.

The differences of the data models of XML (trees) and RDF (graphs) are no obstacle, since XML only provides the syntactic structure used for organizing an RDF document. Since XML requires hierarchic structures, the encoding of triples now must as well be hierarchical. The space efficient Turtle descriptions of the previous section have illustrated that it is often useful to assign multiple predicate-object pairs to a single subject. Accordingly, triples in RDF/XML are grouped by their subjects. The following example encodes the RDF graph from Fig. 2.1:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/">

  <rdf:Description rdf:about="http://semantic-web-book.org/uri">
    <ex:publishedBy>
      <rdf:Description rdf:about="http://crcpress.com/uri">
      </rdf:Description>
    </ex:publishedBy>
  </rdf:Description>

</rdf:RDF>
```

After an optional specification of XML version and encoding, the document starts with a first node of type `rdf:RDF`. This element is generally used as the root of any RDF/XML document. At this place, we also declare the global (XML-)namespaces for `ex:` and `rdf:`. Just as in Turtle, namespaces allow us to abbreviate URIs with QNames, this time building upon the existing XML namespace mechanism. While abbreviations for namespaces are still mostly arbitrary, it is a convention to use the prefix `rdf:` for the RDF namespace as given in the above example. In the following, elements that have a special meaning in the RDF serialization are recognized by that prefix.

Nested within the element `rdf:RDF`, we find the encoding of the sole triple of the above example. Subject and object are described by elements of the type `rdf:Description`, where the XML attribute `rdf:about` defines the identifier of the resource. The predicate of the encoded triple is represented directly as the element `ex:publishedBy`.

Multiple triples can be encoded by representing each of them by a separate element of type `rdf:Description`, which may lead to multiple such elements referring to the same subject. Likewise, the order of the triples is of course not important. However, it is also possible to nest elements of type `rdf:Description`, possibly leading to a more concise serialization. The

following example encodes the graph from Fig. 2.3:[3]

```
<rdf:Description rdf:about="http://semantic-web-book.org/uri">
  <ex:title>Foundations of Semantic Web Technologies</ex:title>
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri">
      <ex:name>CRC Press</ex:name>
    </rdf:Description>
  </ex:publishedBy>
</rdf:Description>
```

Here we can see how literals are represented simply as the contents of a predicate-element. The name "CRC Press" is given directly by nesting XML elements instead of creating a second top-level subject element for describing `http://crcpress.com/uri`. Some further abbreviations are allowed:

```
<rdf:Description rdf:about="http://semantic-web-book/uri"
                 ex:title= "Foundations of Semantic Web Technologies">
  <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
</rdf:Description>
<rdf:Description rdf:about="http://crcpress.com/uri"
                 ex:Name="CRC Press" />
```

This syntax requires some explanation. First of all, all predicates with literal objects have been encoded as XML attributes. This abbreviation is admissible only for literals – objects referred to by URIs cannot be encoded in this way, since they would then be misinterpreted as literal strings.

Moreover, the element `ex:publishedBy` makes use of the special attribute `rdf:resource`. This directly specifies the object of the triple, such that no further nested element of type `rdf:Description` is necessary. This is the reason why `ex:publishedBy` has no content so that it can be written as an empty-element tag, as opposed to giving separate start and end tags. This shortcut notation is only allowed for URIs, i.e., every value of `rdf:resource` is considered as a URI.

Since we thus avoid a nested description of `http://crcpress.com/uri`, another description appears at the outer level. The predicate `ex:Name` is again encoded as an XML attribute and the otherwise empty description can be closed immediately.

[3] In many cases, we show only the interesting parts of an RDF/XML document in examples. The declaration of `rdf:RDF` can always be assumed to be the same as in the initial example on page 28.

We see that RDF/XML provides a multitude of different options for representing RDF. Some of those options stem from the underlying XML syntax. As an example, it is not relevant whether or not an element without contents is encoded by an empty-element tag instead of giving both start and end tags. A larger amount of freedom, however, is provided by RDF since the same triples can be encoded in many different ways. Our previous two examples certainly do not describe the same XML tree, yet they encode the same RDF graph.

W3C Validator The *W3C Validator* is a Web Service that can be employed to check the validity of RDF/XML documents with respect to the official specification. A simple online form is provided to upload XML-encoded RDF which is then validated. Valid documents are processed to extract individual triples and a visualization of the corresponding graph, whereas invalid documents lead to error messages that simplify the diagnosis of problems. This Web Service can also be used to investigate RDF/XML examples given within this book, though it should not be forgotten that many examples are only partial and must be augmented with a suitable `rdf:RDF` declaration to become valid.
The W3C Validator is found at `http://www.w3.org/RDF/Validator/`

2.2.4 RDF in XML: URIs and Other Problems

Namespaces in RDF/XML have been introduced above as a way of abbreviating URIs in the RDF/XML serialization. The truth, however, is that namespaces in RDF/XML are an indispensable part of the encoding rather than an optional convenience. The reason is that RDF/XML requires us to use resource identifiers as names of XML elements and attributes. But all URIs necessarily contain a colon – a symbol that is not allowed in XML names! Using namespaces, we can "hide" a URI's own colon within the declared prefix.

On the other hand, namespaces can only be used for abbreviating XML tags and attributes, but are not allowed within attribute values and plain text contents between XML tags. This is the reason why we used the complete URI `http://semantic-web-book/uri` in all previous examples, instead of employing a QName `book:uri` as in our earlier Turtle examples. An attribute assignment of the form `rdf:about="book:uri"` is not correct, and `book` in this case would be interpreted as the scheme part of a URI but not as an XML namespace prefix.

Thus we are in the unfortunate situation of having to write the same URI differently in different positions of an RDF/XML document. The next section introduces a method that still allows us to at least abbreviate URIs in cases where namespaces cannot be used. XML has a number of further syntactic restrictions that may complicate the encoding of arbitrary RDF graphs. It

is, e.g., not allowed to use a hyphen directly after a colon in XML tags, even though hyphens might occur within URIs. It may thus become necessary to declare auxiliary namespaces merely for the purpose of exporting single elements in a valid way.

Another practical problem is that the percent sign % occurs frequently within URLs since it is used to escape forbidden characters. The string %20, e.g., encodes the space character. Just like colon, the percent sign is not allowed in XML tags, and it can happen that existing URLs cannot be used as URIs in RDF. Fortunately, many problems of this kind are already addressed by existing RDF programming libraries, so that application developers do not need to focus on such serialization issues. Yet they should be aware that there are valid URIs that cannot be encoded at all in XML.

2.2.5 Shorter URIs: XML Entities and Relative URIs

In the above examples, we have always used absolute URIs as values of the attributes `rdf:about` and `rdf:resource`, as the use of namespaces would not be admissible in this context. This section discusses two methods for abbreviating such values as well. While these abbreviations are of course optional additions to the basic syntax, they are very widely used and thus indispensable for understanding most of today's RDF documents.

A simple method to abbreviate values in XML is the use of so-called *XML entities*. An entity in XML is a kind of shortcut that can be declared at the beginning of a document, and referred to later in the document instead of giving its complete value. The following is a concrete example of an XML document using this feature:

```
<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE rdf:RDF[
        <!ENTITY book 'http://semantic-web-book.org/'>
]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/">

  <rdf:Description rdf:about="&book;uri">
    <ex:title>Foundations of Semantic Web Technologies</ex:title>
  </rdf:Description>

</rdf:RDF>
```

An obvious novelty in this example is the initial entity declaration enclosed in `<!DOCTYPE rdf:RDF[` and `]>`. This part of the XML document constitutes its *document type declaration* which might provide a so-called *Document Type Definition* (DTD). A DTD can be used to declare entities as above, but also

to define a number of further restrictions on the contents of the XML document. All document type declarations used in this book, however, are plain entity definitions, so that we can content ourselves with knowing that entities in RDF/XML are defined as above. In our example, the entity defined is called `book` and its value is `http://semantic-web-book.org/`. Further entities could easily be defined by providing additional lines of the form `<!ENTITY name 'value'>`.

We may now refer to our newly defined entity by writing `&book;` as in the value of `rdf:about` above, and the XML document is interpreted just as if we had written the declared value of the entity at this position. Such entity references are allowed in XML attribute values and within plain text data contained in an element, such as the texts used for serializing data literals in Section 2.2.3. Entities cannot be used within names of XML elements and attributes – there we have to stick to the use of namespaces. In our current example, defining a new entity does not actually shorten the document, but usually entities for common URI prefixes lead to much more concise serializations and may also increase readability. XML also provides a small number of pre-defined entities that are useful for encoding certain symbols that would otherwise be confused with parts of the XML syntax. These entities are `<` (<), `>` (>), `&` (&), `&apos;` ('), and `&quot;` (").

There is another common case in which URIs might be abbreviated: in many RDF documents, URIs primarily stem from a common *base namespace*. A website that exports data in RDF, e.g., is likely to use many URIs that begin with the site's domain name. XML has the concept of a *base URI* that can be set for elements in a document using the attribute `xml:base`. Other attributes in the XML document then may, instead of full URIs, use so-called *relative references*. Such entries refer to a full URIs which are obtained by preceding the entries with the given base URI, as illustrated by the following example:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/"
         xml:base ="http://semantic-web-book.org/" >

  <rdf:Description rdf:about="uri">
    <ex:publishedBy rdf:resource="http://crcpress.com/uri" />
  </rdf:Description>

</rdf:RDF>
```

The relative reference `rdf:about="uri"` is thus interpreted as the URI `http://semantic-web-book/uri`. Values of `rdf:resource` or `rdf:datatype` (explained later) can be abbreviated in the same fashion. Relative references are distinguished from full URIs by lacking a scheme part; see Fig. 2.2. It is

possible to use relative references even without declaring the intended base URI beforehand: in this case, the base URI of the document – based on the URL it was retrieved from – is used. This mechanism is less robust since locations of documents may change; hence it is suggested to provide explicit base URIs whenever needed.

A second common use of `xml:base` for abbreviating URIs relates to the attribute `rdf:ID`. This attribute can be used just like `rdf:about`, but it always expects a single fragment identifier as its value, whereas complete URIs are not allowed. The full URI is then constructed by using the given value as a fragment for the base URI (which thus should not contain any fragment), i.e. we can obtain the URI by extending the base URI with the symbol # followed by the value of `rdf:ID`.

Thus we find that `rdf:ID="name"` has essentially the same meaning as `rdf:about="#name"`. The most relevant difference of both ways of writing URIs is that every value of `rdf:ID` must be used only once for a given base URI. An RDF/XML document may thus contain one element with a given ID, but it may still contain further elements that refer to the same URI by means of `rdf:about` and `rdf:resource`.

The Turtle syntax for RDF provides a similar support for relative references, which are resolved by using the base URI (URL) of the document. Setting the base URI explicitly is not encompassed by the current specification, even though the syntax `@base` was proposed for this purpose. Overall, relative references in Turtle are of only minor importance since namespace declarations can be used without the restrictions of the XML syntax.

Figure 2.4 provides an overview of the various forms of abbreviation mechanisms that we have introduced for RDF/XML. Note that a principal difference between XML entities and (base) namespaces is that the former can be declared only once for the whole document, whereas the latter may be declared in arbitrary XML start tags or empty-element tags. The namespaces then apply to the element within which they were declared, and to all subelements thereof. Moreover, entities can be used not only for abbreviating URIs but provide shortcuts for arbitrary text content, even within literal values.

2.2.6 Where Do URIs Come From? What Do They Mean?

Does the use of URIs, which is strictly required throughout RDF, allow for a semantically unambiguous interpretation of all RDF-encoded information? The answer is clearly no. It is still possible to use different URIs for the same resource, just as it is still possible to use the same URI for different things. A possible solution for this problem is the use of well-defined *vocabularies*. As in XML, the term vocabulary in RDF is most commonly used to refer to collections of identifiers with a clearly defined meaning. A typical example is provided by RDF itself: the URI

`http://www.w3.org/1999/02/22-rdf-syntax-ns#Description`,