CHAPTER 2

Knowledge Representation

To improve the automated processability of web sites, formal *knowledge representation* standards are required that can be used not only to annotate markup elements for simple machine-readable data but also to express complex statements and relationships in a machine-processable manner. After understanding the structure of these statements and their serialization in the Resource Description Framework (RDF), the structured data can be efficiently modeled as well as annotated in the markup, or written in separate, machine-readable metadata files. The formal definitions used for modeling and representing data make efficient data analysis and reuse possible. The three most common machine-readable annotations that are recognized and processed by search engines are RDFa (RDF in attributes), HTML5 Microdata, and JSON-LD, of which HTML5 Microdata is the recommended format. The machine-readable annotations extend the core (X)HTML markup with additional elements and attributes through external vocabularies that contain the terminology and properties of a knowledge representation domain, as well as the relationship between the properties in a machine-readable form. Ontologies can be used for searching, querying, indexing, and managing agent or service metadata and improving application and database interoperability. Ontologies are especially useful for knowledge-intensive applications, in which text extraction, decision support, or resource planning are common tasks, as well as in knowledge repositories used for knowledge acquisition. The schemas defining the most common concepts of a field of interest, the relationships between them, and related individuals are collected by semantic knowledge bases. These schemas are the de facto standards used by machine-readable annotations serialized in RDFa, HTML5 Microdata, or JSON-LD, as well as in RDF files of Linked Open Data datasets.

Vocabularies and Ontologies

Controlled vocabularies of the Semantic Web collect concepts and terms used to describe a field of interest or area of concern. *Ontologies* are more complex, very formal definitions of terms, individuals and their properties, object groups (classes), and relationships between individuals suitable to describe virtually any statement related to the field of interest in a machine-readable form.

For example, to declare a person in a machine-readable format, we need a vocabulary that has the formal definition of "Person." A straightforward choice is the Friend of a Friend (FOAF) vocabulary, which has a `Person` class that defines typical properties of a person, including, but not limited to, `name` and `homepage`. If we write this code in XML serialization, we would get the code in Listing 2-1.

Listing 2-1. Pseudocode for Defining the Class and a Property of a Resource

```
<Person>
  <name>Leslie Sikos</name>
</Person>
```

This code provides hierarchy, inferring that `Person` is a class and `name` is a property; however, it is out of context. We have to declare which external vocabulary defines this class and property, using the *namespace mechanism*. In RDF/XML serialization, this can be done using the `xmlns` attribute in the form `xmlns:vocabulary_prefix="vocabulary_namespace:web_address"`, in our case, `xmlns:foaf="http://xmlns.com/foaf/0.1/"`, which points to the FOAF namespace at `http://xmlns.com/foaf/0.1/`. The namespace mechanism makes it possible to abbreviate `http://xmlns.com/foaf/0.1/` as `foaf` (also known as prefix), so `foaf:Person` refers to `http://xmlns.com/foaf/0.1/Person`, `foaf:homepage` to `http://xmlns.com/foaf/0.1/homepage` and so forth (see Listing 2-2).

■ **Note** These links are often symbolic links that do not always point to a dedicated web page for each individual property and are sometimes forwarded to the domain of the namespace. Some vocabularies have a namespace address mechanism whereby all links point directly to the corresponding section of the machine-readable vocabulary file. The human-readable explanation of the properties of external vocabularies is not always provided. In case of FOAF, the web address of the individual property addresses point to the web site of the specification (`http://xmlns.com/foaf/spec/`), while the individual properties have their own fragment identifier, such as the `Person` property's address, `http://xmlns.com/foaf/spec/#term_Person`.

Listing 2-2. Describing the Name of a Person Using a Class and a Property from a Vocabulary

```
… xmlns:foaf="http://xmlns.com/foaf/0.1/"
…
<foaf:Person>
  <foaf:name>Leslie Sikos</foaf:name>
</foaf:Person>
```

The format and serialization of the structured data are independent of the vocabulary definitions, so, for example, the same schema.org reference can be used in RDF, RDFa, HTML5 Microdata, and JSON-LD. The vocabulary or ontology required depends on the area of interest you want to represent; however, some knowledge domains such as persons and books can be described with classes and properties from more than one vocabulary.

The schema.org Vocabulary Collection

Covering approximately 300 concept definitions, `https://schema.org` is one of the most frequently used collections of structured data markup schemas. Schema.org was launched by Google, Yahoo!, and Bing in 2011. Schema.org contains the machine-readable definitions of the most commonly used concepts, making it possible to annotate actions, creative works, events, services, medical concepts, organizations, persons, places, and products.

Analogously to the previous example, if we want to describe a book, we need a vocabulary with the definition of "Book" and typical book properties. If we want to add the book title with a more descriptive property than the `name` property of schema.org, we can use the `title` property from the *Dublin Core* (*DC*) vocabulary, resulting in two namespace declarations (see Listing 2-3).

Listing 2-3. Describing a Book

```
…
xmlns:schema="http://schema.org/"
xmlns:dc="http://purl.org/dc/terms/"
…
<schema:Book>
  <dc:title>Web Standards: Mastering HTML5, CSS3, and XML</dc:title>
</schema:Book>
```

Here, `schema:Book` abbreviates `http://schema.org/Book`, which is the machine-readable definition of the `Book` class, while `dc:title` abbreviates `http://purl.org/dc/terms/title`, which is the machine-readable definition of the `title` property.

The most common schema.org types (classes) and properties are collected at `http://schema.org/docs/gs.html#schemaorg_types`, while the full list of properties is available at `http://schema.org/docs/full.html`.

General, Access, and Structural Metadata

General metadata, such as abstract, creator, date, publisher, title, language of web resources (web sites, images, videos), physical resources (books, CDs), and objects such as artworks, can be described using Dublin Core. The Dublin Core elements are widely deployed in machine-readable annotations, used on DMOZ [1], one of the biggest multilingual open-content directories of web links, as well as in XMP metadata of JPEG photos. The namespace of Dublin Core Elements (`dc`) is `http://purl.org/dc/elements/1.1/`, and the namespace of Dublin Core terms (`dcterms`) is `http://purl.org/dc/terms/`.

Structured datasets can be described using terms from the *Vocabulary of Interlinked Datasets* (*VoID*). VoID covers general, access, and structural metadata definitions, as well as the description of links between structured datasets. The prefix of VoID is `void`, and the namespace is `http://rdfs.org/ns/void#`.

Person Vocabularies

The features of a person and relationships between people can be described by a variety of controlled vocabularies, as summarized in Table 2-1.

Table 2-1. *Person Vocabularies*

Vocabulary	Abbreviation	Namespace	Typical Use
`Person` class from schema.org	`schema:Person`	`http://schema.org/Person`	Given name, family name, gender, affiliation, award, nationality, honorific prefix or suffix, job title, etc.
Friend of a Friend	`foaf`	`http://xmlns.com/foaf/0.1/`	Person, name, gender, home page
Contact: Utility concepts for everyday life	`contact`	`http://www.w3.org/2000/10/swap/pim/contact`	Contact location, personal title, mother tongue, nearest airport to your residence
vcard	`vcard`	`http://www.w3.org/2001/vcard-rdf/3.0#`	Electronic business card
Bio	`bio`	`http://vocab.org/bio/0.1/`	Biographical information
Relationship vocabulary	`relationship`	`http://vocab.org/relationship/`	Relationships between people (`friendOf`, `parentOf`, `spouseOf`, etc.)

Book Vocabularies

Books can be precisely described using properties from http://schema.org/Book, defining book formats, the number of pages, the copyright holder, the genre, and other features of books. Dublin Core is often used to declare general metadata of books. The International Standard Book Number (ISBN) of books can be declared not only with the isbn property of the Book class of schema.org (defined at http://schema.org/isbn), but also using the isbn property from the URN vocabulary. Books you intend to read, books you've already read, or your favorite books can be described using the reading list schema through the http://www.ldodds.com/schemas/book/ namespace.

PRISM: A Publishing Vocabulary

The Publishing Requirements for Industry Standard Metadata (PRISM) describes many components of print, online, mobile, and multimedia content, including the following:

- Creator, contributor, copyright owner
- Locations, organizations, topics, people, and events, conditions of reproduction
- Publication date, including cover date, post date, volume, number
- Restrictions for republishing and reuse

PRISM is commonly used for describing partner syndication, content aggregation, content repurposing, resource discovery, multiple channel distribution, content archiving, capture rights usage information, RSS, XMP, and machine-readable annotations of web sites. The PRISM namespaces are http://prismstandard.org/namespaces/basic/2.1/ for PRISM 2.1 Basic (typical prefix: prism) and http://prismstandard.org/namespaces/prism-ad/3.0/ for PRISM 3.0 (usual prefix: prism-ad).

GoodRelations: An E-commerce Ontology

The de facto ontology for e-commerce is GoodRelations (gr), which is suitable for describing businesses, offerings, prices, features, payment options, opening hours, and so on. The namespace of GoodRelations is http://purl.org/goodrelations/v1#. GoodRelations is widely deployed and also used by Yahoo! and BestBuy.

Publication Ontologies

While the generic metadata of publications can be expressed in Dublin Core, there are ontologies specially written for describing publications and citations. Table 2-2 summarizes the four most deployed publishing ontologies (FaBiO, PRO, PSO, and PWO) and the four referencing ontologies (CiTO, BiRO, C4O, and DoCO) that are known as the *Semantic Publishing and Referencing Ontologies* (*SPAR*), as well as the Bibliographic Ontology (bibo).

Table 2-2. *Publication and Referencing Ontologies*

Ontology	Abbreviation	Namespace	Typical Use
Bibliographic Ontology	bibo	http://purl.org/ontology/bibo/	Citation, document classification, describe documents, distributors, editors, interviewers, performers, ISBN, etc.
Bibliographic Reference Ontology	biro	http://purl.org/spar/biro/	Bibliographic records, references, collections, and lists
Citation Counting and Context Characterization Ontology	c4o	http://purl.org/spar/c4o/	Number of citations, citation context
Citation Typing Ontology	cito	http://purl.org/spar/cito/	Factual and rhetorical type and nature of citations (e.g., shared authors, one publication confirms the conclusion of another one)
Document Components Ontology	doco	http://purl.org/spar/doco/	Chapter, section, paragraph, table, preface, glossary, etc.
FRBR-aligned Bibliographic Ontology	fabio	http://purl.org/spar/fabio/	Abstracts, articles, artistic works, theses, blog posts, conference proceedings
Publishing Roles Ontology	pro	http://purl.org/spar/pro	Roles of agents (e.g., author, editor, reviewer, publisher)
Publishing Status Ontology	pso	http://purl.org/spar/pso	Status of publication (e.g., submitted manuscript, accepted manuscript, proof)
Publishing Workflow Ontology	pwo	http://purl.org/spar/pwo	Stages of publication workflow (e.g., under review)

DOAP: A Project Management Vocabulary

Description of a project (DOAP) is a vocabulary to describe software projects, especially open source projects and their associated resources, including participants and web resources. The namespace of DOAP is http://usefulinc.com/doap/.

Licensing Vocabularies

ALicensing, such as copyright information, permissions and prohibition regarding the reproduction, distribution, and sharing of creative works, as well as creating derivative works, is best described using Creative Commons (cc) licenses. The namespace of Creative Commons is http://creativecommons.org/ns#.

Media Ontologies

There are ontologies dedicated to media resources, such as music and video files, as summarized in Table 2-3.

Table 2-3. *Media Ontologies*

Ontology	Abbreviation	Namespace	Typical Use
The Music Ontology	mo	http://purl.org/ontology/mo/	Artist, composer, conductor, discography, imdb, record, remixer, singer, tempo, etc.
VidOnt: The Video Ontology	vidont	http://vidont.org/	Movie properties (remake, sequel, narrator, etc.), video file properties (aspect ratio, audio codec, letterboxed, video bitrate, MAR, etc.)

Vocabularies for Online Communities

Posts, user roles, threads, user accounts, and user groups of online communities can be described using *Semantically-Interlinked Online Communities* (*SIOC*). The namespace of SIOC Core is http://rdfs.org/sioc/ns#.

Facebook uses the vocabulary of Facebook OpenGraph (og) to allow web pages the same functionality as any other object on Facebook. The namespace of OpenGraph is http://ogp.me/ns#.

Knowledge Management Standards

The most frequently used knowledge management standards are the Resource Description Framework (RDF), the Web Ontology Language (OWL), and the Simple Knowledge Organization System (SKOS).

Resource Description Framework (RDF)

On the Semantic Web, structured datasets are usually expressed in, or based on, the *Resource Description Framework* (*RDF*) [2]. RDF can be used to create a machine-interpretable description about any kind of web resource, because RDF files can be extended with an arbitrary number of external vocabularies. In fact, RDF and other core Semantic Web standards such as RDFS and OWL have their own vocabularies, which are usually combined with one another and extended using other vocabularies, in order to describe objects and their properties. Keep in mind, however, that RDF is far more than just a vocabulary, as it is a fully featured semantic data-modeling language. The namespace of RDF is http://www.w3.org/1999/02/22-rdf-syntax-ns#.

The RDF vocabulary defines classes for XML literal values (`rdf:XMLLiteral`), properties (`rdf:Property`), RDF statements (`rdf:Statement`), RDF lists (`rdf:List`), as well as containers of alternatives (`rdf:Alt`), unordered containers (`rdf:Bag`), and ordered containers (`rdf:Seq`). An instance of `rdf:List` is `rdf:nil`, which represents the empty list. The RDF vocabulary also defines properties such as `rdf:type` (an instance of `rdf:Property` used to express that a resource is an instance of a class), `rdf:first` (the first item in the subject RDF list), `rdf:rest` (the rest of the subject RDF list after `rdf:first`), `rdf:value` (structured value), `rdf:subject` (the subject of the RDF statement), `rdf:predicate` (the predicate of the RDF statement), and `rdf:object` (the object of the RDF statement).

The RDF data model is based on statements to describe and feature resources, especially web resources, in the form of subject-predicate-object (resource-property-value) expressions called *RDF triples* or *RDF statements*. The predicate (property) describes the relationship between the subject and the object. For example, the natural language sentence "Leslie's homepage is http://www.lesliesikos.com" can be expressed as shown in Table 2-4. All elements of the triple are resources defined by a unique URI (see Listing 2-4).

Table 2-4. *An RDF Triple*

	RDF Data Model	RDF Triple
Subject	Leslie	http://www.lesliesikos.com/metadata/sikos.rdf#lesliesikos
Predicate	The machine-readable definition of "homepage" from the Friend of a Friend (FOAF) external vocabulary	http://xmlns.com/foaf/0.1/homepage
Object	http://www.lesliesikos.com	http://www.lesliesikos.com

Listing 2-4. Describing a Person in RDF/XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ➥
 xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person rdf:about="http://www.lesliesikos.com/metadata/sikos.rdf#lesliesikos">
    <foaf:homepage rdf:resource="http://www.lesliesikos.com" />
    <foaf:family_name>Sikos</foaf:family_name>
    <foaf:givenname>Leslie</foaf:givenname>
  </foaf:Person>
</rdf:RDF>
```

The `about` attribute of RDF declares the subject of the RDF statement, which is, in this case, http://www.lesliesikos.com/metadata/sikos.rdf#lesliesikos. The fragment identifier `#lesliesikos` is used to identify an actual person rather than a document (`sikos.rdf`). Those objects whose value is a web address, such as the home page of a person, are declared using the `resource` attribute of RDF, in contrast to those that are string literals (character sequences), such as `Sikos` (the value of the `family_name` property from the FOAF vocabulary). The syntax of this example is known as the *RDF/XML serialization* (*RDF/XML*), which is the normative syntax of RDF [3], using the `application/rdf+xml` Internet media type and the `.rdf` or `.xml` file extension. Structured datasets can be written in RDF using a variety of other syntax notations and data serialization formats, for example, RDFa, JSON-LD, Notation3 (N3), Turtle, N-Triples [4], TRiG [5], and TRiX [6], so the syntax of RDF triples varies from format to format. The N3 syntax is, for example, less verbose than the RDF/XML serialization, where the namespace prefix is declared by the `@prefix` directive, the URIs are delimited by the less than (`<`) and greater than (`>`) signs, and the triples are separated by semicolons (`;`) (see Listing 2-5).

Listing 2-5. Describing a Person in N3

```
@prefix :     <http://www.lesliesikos.com/metadata/sikos.rdf#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:lesliesikos a foaf:Person ;
  foaf:givenname "Leslie" ;
  foaf:family_name "Sikos" ;
  foaf:homepage <http://www.lesliesikos.com> .
```

Shorthand notation can be used for the most common predicates (see Table 2-5).

Table 2-5. *Shorthand Notation for Common Predicates*

Predicate	Shorthand Notation
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	a
<http://www.w3.org/2002/07/owl#sameAs>	=
<http://www.w3.org/2000/10/swap/log#implies>	=> or <=

This is the reason why the RDF type of the person is declared using a. If the Notation 3 code is in an external file, the typical file extension is .n3. The MIME type and character encoding of N3 should be declared as text/n3; charset=utf-8. Tokenizing and whitespace handling are not specified in the N3 grammar. Base URIs to be used for the parsing of relative URIs can be set with the @base directive in the form @base <http://example.com/overview/>. Several N3 rules for string escaping are derived from Python, namely, stringliteral, stringprefix, shortstring, shortstringitem, longstring, longstringitem, shortstringchar, and longstringchar. Additionally, the \U extension, also used in another RDF serialization (N-Triples), can be applied. Legal escape sequences are \newline, \\ (backslash, \), \' (single quote, '), \" (double quote, "), \n (ASCII Linefeed, LF), \r (ASCII Carriage Return, CR), \t (ASCII Horizontal Tab, TAB), \uhhhh (Unicode character in BMP), and \U00hhhhhh (Unicode character in plane 1–16 notation). The escapes \a, \b, \f, and \v cannot be used, because the corresponding characters are not allowed in RDF.

A subset of N3 is the *Terse RDF Triple Language*, often referred to as *Turtle*. Turtle provides a syntax to describe RDF graphs in a compact textual form, which is easy to develop. It is a subset of Notation 3 (N3) and a superset of N-Triples. Turtle is popular among Semantic Web developers and considered an easy-to-read alternative to RDF/XML. The typical file extension of Turtle files is .ttl. The character encoding of Turtle files should be UTF-8. The MIME type of Turtle is text/turtle. Turtle is supported by many software frameworks that can be used for querying and analyzing RDF data, such as Jena, Redland, and Sesame. Turtle files consist of a sequence of directives, statements representing triples, and blank lines. Triples can be written in Turtle as a sequence of subject-predicate-object terms, separated by whitespace, and terminated by a period (.). URIs are written in angle brackets (<>), and string literals are delimited by double quotes ("") such as <http://www.lesliesikos.com/metadata/sikos.rdf#> <http://xmlns.com/foaf/0.1/homepage> <http://www.lesliesikos.com> . Using the URI prefix declaration @PREFIX foaf: <http://xmlns.com/foaf/0.1/> ., this can be abbreviated as <http://www.lesliesikos.com/metadata/sikos.rdf#> foaf:homepage <http://www.lesliesikos.com> ., where foaf:homepage declares the concatenation of http://xmlns.com/foaf/0.1/ with homepage, revealing the original URI http://xmlns.com/foaf/0.1/homepage.

Figure 2-1 represents the triples of Listing 2-5 as an *RDF graph*, which is a directed, labeled graph in which the *nodes* are the resources and values [7]. The nodes and predicate arcs of the RDF graph correspond to *node elements* and *property elements*. The default node element is rdf:Description, which is very frequently used as the generic container of RDF statements in RDF/XML. To add context to RDF statements

and make them globally interpretable, RDF triples are sometimes stored with the name of the graph, called *quads* (subject-predicate-object-graph name), which will be demonstrated in later chapters.

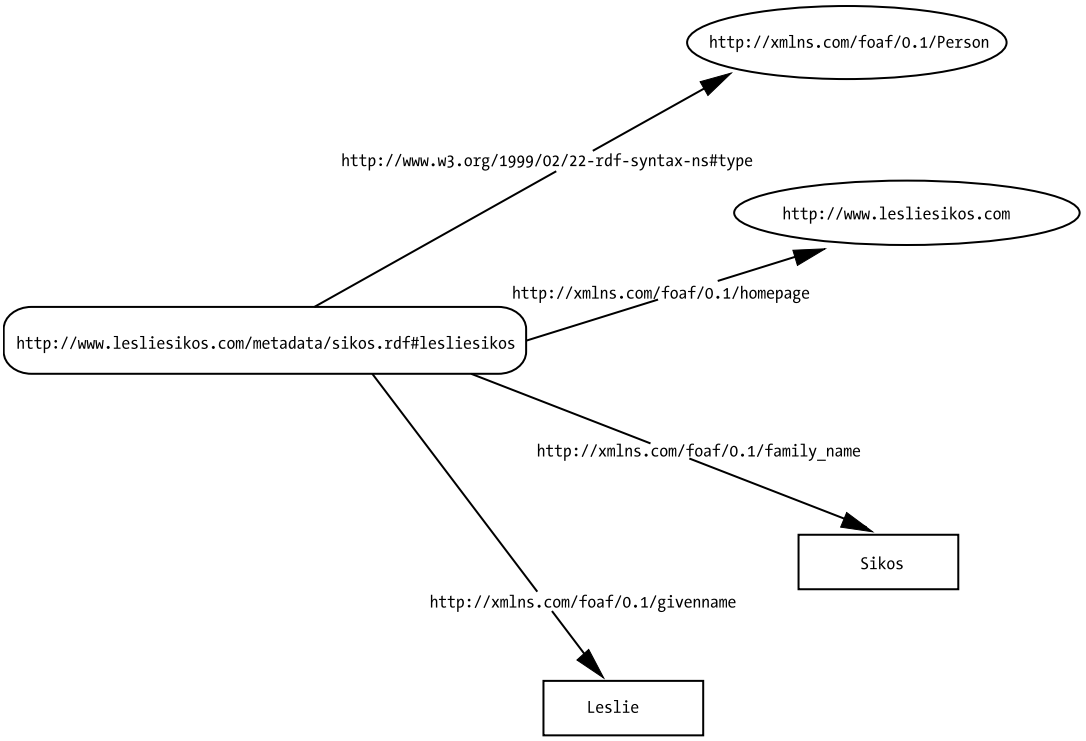

Figure 2-1. *A simple RDF graph*

The example can be extended with properties from other external vocabularies, but the concept remains the same. Once you've created your RDF file, you have a machine-readable metadata file you can upload to your web site. Semantic software agents can find and retrieve the information in such files (see Figure 2-2), display the human-readable part in a visually appealing manner (see Figure 2-3) and generate a scalable graph based on the RDF triples (see Figure 2-4), and infer new information.

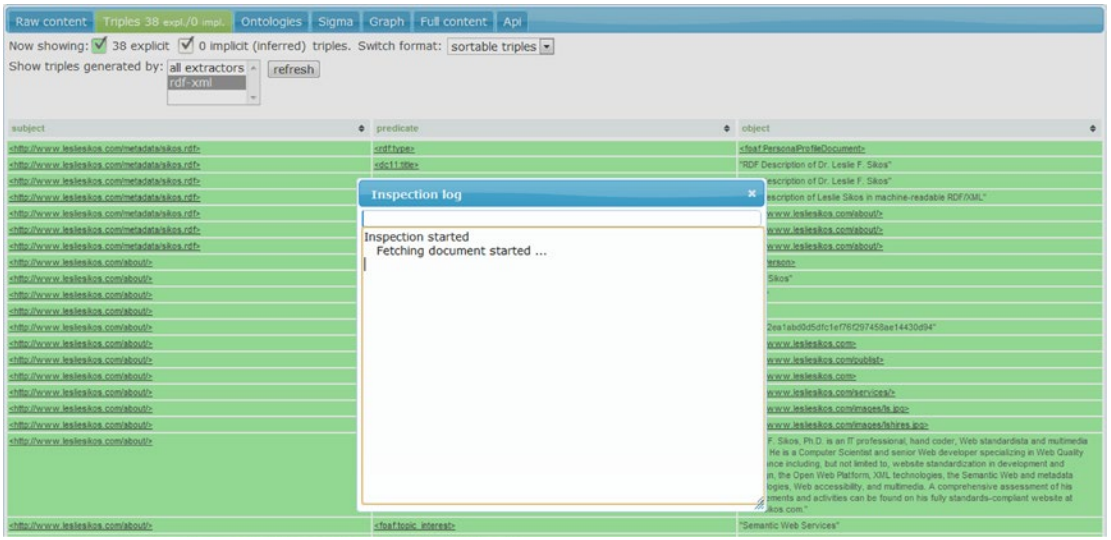

Figure 2-2. *RDF triples extracted by Sindice Web Data Inspector [8]*

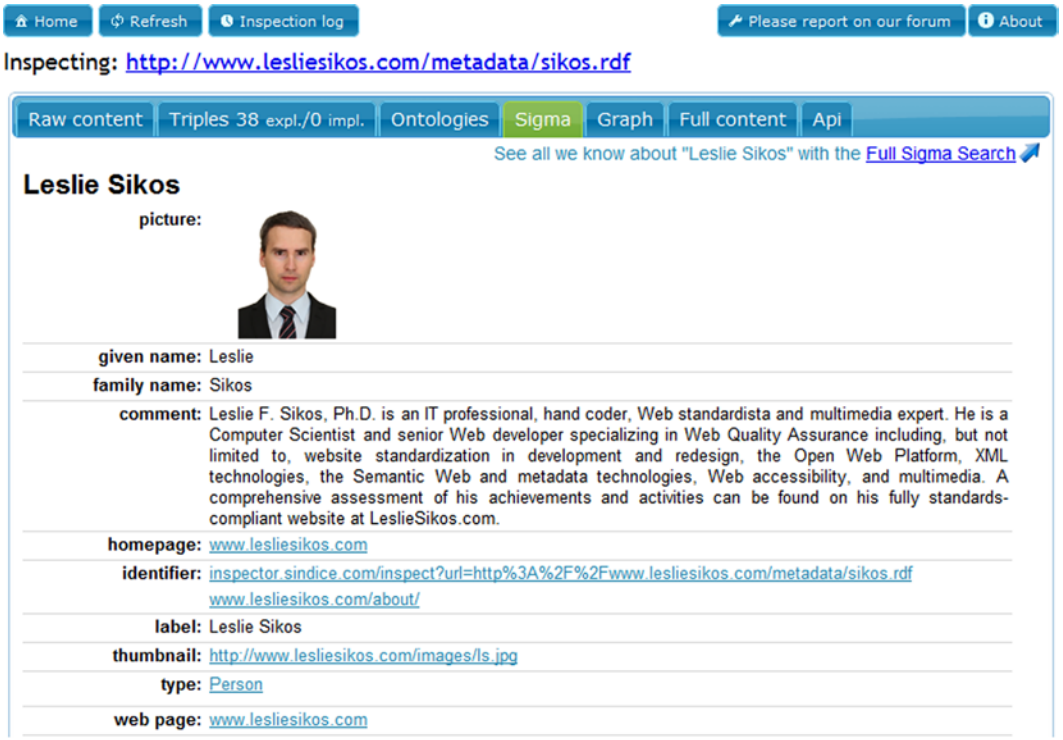

Figure 2-3. *A personal description extracted from RDF and displayed on a web page*

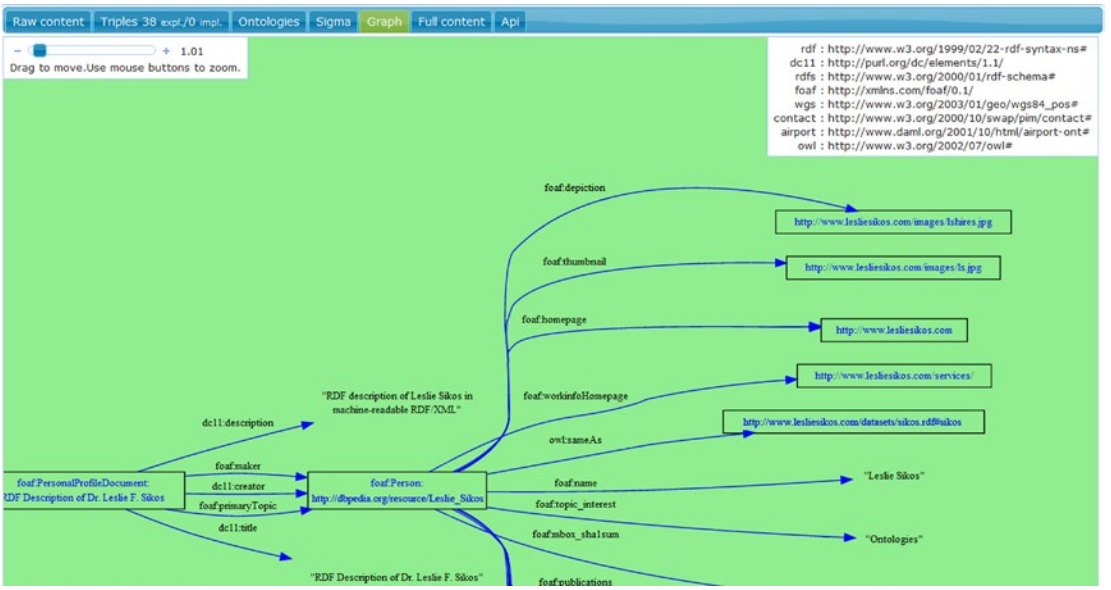

Figure 2-4. *A graph generated from an RDF file*

While such machine-readable RDF files are useful, their primary application is data modeling, so the RDF files are separate from the markup of your web site. You can add structured data directly to the markup, such as (X)HTML5, by using *machine-readable annotations*, which can be processed by semantic data extractors and, if needed, converted into RDF.

Machine-Readable Annotations

There are four machine-readable annotation formats for web sites (by order of introduction):

- *Microformats*, which publish structured data about basic concepts,[1] such as people, places, events, recipes, and audio, through core (X)HTML attributes
- *RDFa*, which expresses RDF in markup attributes that are not part of the core (X) HTML vocabularies
- *HTML5 Microdata*, which extends the HTML5 markup with structured metadata (a HTML5 Application Programming Interface)
- *JSON-LD*, which adds structured data to the markup as JavaScript code

RDFa and JSON-LD can be used in most markup language versions and variants, while HTML5 Microdata can be used in (X)HTML5 only. All these annotation formats have their own syntax. For example, the vocabulary is declared with the `vocab` attribute in RDFa, the `itemtype` attribute in Microdata, and `context` in JSON-LD (see Table 2-6).

[1]The other three formats are more advanced, as they can use concepts from any external vocabulary.

Table 2-6. *Data Represented as Structured Data in a Microformat, Microdata, RDFa, and JSON-LD*

Markup without Semantic Annotation	Leslie Sikos Leslie's web site: lesliesikos.com
Markup with the hCard microformat	**<link rel="profile" href="http://microformats.org/profile/hcard" />** … <div **class="vcard"**> <span **class="fn"**>Leslie Sikos Leslie's web site: <a **class="url"** href="http://www.lesliesikos.com">lesliesikos.com
Markup with HTML5 Microdata	<div **itemscope="itemscope" itemtype="http://schema.org/Person"**> <span **itemprop="name"**>Leslie Sikos Leslie's web site: lesliesikos.com
Markup with RDFa	<div **vocab="http://schema.org/" typeof="Person"**> <span **property="name"**>Leslie Sikos Leslie's web site: lesliesikos.com
Markup with JSON-LD	**<script type="application/ld+json">** **{** **"@context": "http://schema.org",** **"@type": "Person",** **"image": "lesliesikos.jpg",** **"name": "Leslie Sikos",** **"url": "http://www.lesliesikos.com"** **}** **</script>**

These syntaxes will be described in the next sections.

Microformats

The results of the very first approach to add machine-readable annotations to the (X)HTML markup are called *microformats* (μF). Some microformats apply and reuse features of existing technologies, such as the `rel` attribute of (X)HTML, while others, such as hCard, extend the core markup vocabulary the simplest way possible: based on *Plain Old Semantic HTML* (*POSH*). Microformats can be implemented not only in (X)HTML markup but also in XML, RSS, Atom, and so on. Microformats can express site structure, link weight, content type, and human relationships with the `class`, `rel`, and `rev` attribute values. They are very easy to write, and a great deal of software supports them (the Operator and Tails Export add-ons for Firefox, the Michromeformats Google Chrome extension, the microformats transformer Optimus, or the Microformats Bookmarklet for Safari, Firefox, and IE).

However, due to limitations and open issues, other machine-readable annotation formats gradually overtook microformats. Applying various microformats as multiple values on the same a element, such as `rel="nofollow"` and `rel="friend"`, cannot be used. The `rev` attribute used by the Vote Links microformat is not supported by HTML5. Profile URIs provided by the `profile` attribute cannot be used on the head element in HTML5, wherein the `profile` attribute values can be declared for the `rel` attribute on anchors (a) or link elements (`link`). As an example, a profile URI is presented for the hCalender microformat with all the three options. The hCalendar microformat is based on the iCalendar standard (RFC 2445). All contents that use hCalendar notation should refer to the hCalendar XMDP profile, in other words, `http://microformats.org/profile/hcalendar`, as shown in Listing 2-6 or Listing 2-7 for the document head or Listing 2-8 as part of the document body. These methods can also be combined.

Listing 2-6. Providing the hCalendar Head Profile in the Document Head (Cannot Be Used in HTML5)

```
<head profile="http://microformats.org/profile/hcalendar">
```

Listing 2-7. Linking to the hCalendar Profile in the Document Head

```
<link rel="profile" href="http://microformats.org/profile/hcalendar" />
```

Listing 2-8. Using the hCalendar Profile in the Document Body

```
<a rel="profile" href="http://microformats.org/profile/hcalendar">hCalendar</a>
```

■ **Note** New structural elements introduced by HTML5, such as `article` or `section`, are not recognized by all microformat parsers, so the preceding attributes on these elements might not be processed.

In the next sections, I will give you an overview of some of the most popular microformats, namely, hCalendar, hCard, `rel="license"`, `rel="nofollow"`, `rel="tag"`, Vote Links, and XFN.

hCalendar and h-event

You can use the hCalendar microformat to create calendar entries for sport events, anniversaries, reminders, meetings, workshops, conferences, and other events.

The root class name for hCalendar is `vcalendar`. The root class name for events is `vevent`, which is required for all event listings. The properties are represented by the elements of hCalendar. The required classes are `dtstart`, which should be provided in the ISO date format,[2] and `summary`. Listing 2-9 shows an hCalendar example.

Listing 2-9. A Three-Day Conference Represented in hCalendar

```
<link rel="profile" href="http://microformats.org/profile/hcalendar" />
…
<div class="vevent">
  <h1 class="summary">Semantic Web Conference 2015</h1>
  <div class="description">Semantic Web Conference 2015 was announced yesterday.</div>
  <div>Posted on: <abbr class="dtstamp" title="20150825T080000Z">Aug 25, 2015</abbr></div>
```

[2]Beyond microformats such as hAtom, hCalendar, hCard, and hReview, several web technologies apply the ISO 8601 date format for date-time representation, such as XML, XML schema datatypes, RDF, and Atom.

```
  <div class="uid">uid1@host.com</div>
  <div>Organized by: <a class="organizer" href="mailto:js@expl.com">js@expl.com</a></div>
  <div>Dates: <abbr class="dtstart" title="20151012T093000Z">October 12 2015, 9.30am ↩
   UTC</abbr> - <abbr class="dtend" title="20151014T200000Z">October 14 2015, 8.00pm ↩
   UTC</abbr></div>
  <div>Status: <span class="status">Confirmed</span></div>
  <div>Filed under:</div>
  <ul>
    <li class="category">Conference</li>
  </ul>
</div>
```

Optional properties include, but are not limited to, `location`, `url`, `dtend` (in ISO date format), `duration` (in ISO date duration format), `rdate`, `rrule`, `category`, `description`, `uid`, `geo`, `attendee`, `contact`, `organizer`, `attach`, and `status`. The geo property has the subproperties `latitude` and `longitude`, while `attendee` has the subproperties `partstat` and `role`. Those who have to publish new events regularly might find the hCalendar generator *hCalendar-o-matic* useful [9].

The specification has been superseded by the h-event specification, which supports the following properties inside a markup element with class `h-event`: `p-name` (event name or title), `p-summary` (short summary), `dt-start` (date and time when the event starts), `dt-end` (date and time when the event ends), `dt-duration` (duration of the event), `p-description` (verbose description), `u-url` (web site), `p-category` (event category or categories), and `p-location` (event location, which can include h-card, h-adr, or h-geo). All properties are optional. An example is shown in Listing 2-10.

Listing 2-10. A Three-Day Conference Annotated Using h-event

```
<div class="h-event">
  <h1 class="p-name"> Semantic Web Conference '15</h1>
  <p>From
    <time class="dt-start" datetime="2015-10-12 09:30">12<sup>th</sup> October 2015,
    9:30am</time>
    to <time class="dt-end" datetime="2015-10-14 20:00">14<sup>th</sup> October 2015,
    8:00pm</time>
    at <span class="p-location">Nice Conference Hall</span></p>
  <p class="p-summary">Semantic Web Conference 2015 was announced yesterday.</p>
</div>
```

hCard

The hCard microformat standard can be used to represent contact data of people, companies, and organizations by semantic markup. hCard metadata should be provided on the contact pages of web sites. In summer 2010, hCard crossed the 2 billion mark, according to the now-discontinued Yahoo! SearchMonkey, which made hCard the most popular metadata format for people and organizations up to 2010. Because hCard is based on the vCard standard (RFC 2426), existing vCards can be easily converted to hCard.[3]

[3]The vCard notation `BEGIN:VCARD` is `class="vcard"` in hCard, `N:` is `class="n"`, `FN:` is `class="fn"`, and so on.

■ **Tip** The vCard standard is widely used for storing electronic business cards. For example, Microsoft Outlook uses this format for the business cards available under Contacts. Also, many smartphones use the vCard format to store contacts in the phone memory (when you set up contacts not to be stored on the SIM card).

The hCard class names should be in lowercase.

■ **Caution** The root class name for an hCard is `vcard`. An element with a class name `vcard` is itself called an hCard.

The two required attributes in hCard are `fn` and `n`. However, the second one is optional if any *implied "N" optimization rules* are in effect.[4] The property `n` might have the subproperties `family-name`, `given-name`, `additional-name`, `honorific-prefix`, and `honorific-suffix`. All other properties are optional, including `adr`, `agent`, `bday`, `category`, `class`, `email`, `geo`, `key`, `label`, `logo`, `mailer`, `nickname`, `note`, `org`, `photo`, `rev`, `role`, `sort-string`, `sound`, `tel`, `title`, `tz`, `uid`, and `url`. Permissible subproperties are `post-office-box`, `extended-address`, `street-address`, `locality`, `region`, `postal-code`, `country-name`, `type`, and `value` for `adr`; `type` and `value` for `email`; `latitude` and `longitude` for `geo`; `organization-name` and `organization-unit` for `org`; and `type` and `value` for `tel`. A typical hCard code looks like Listing 2-11.

Listing 2-11. A Typical hCard

```
<link rel="profile" href="http://microformats.org/profile/hcard" />
…
<div id="hcard-John-Smith" class="vcard">
  <img src="http://www.example.com/jsmith.jpg" alt="Photo of John Smith" class="photo" />
  <a class="url fn" href="http://www.example.com">John Smith</a>
  <div class="org">Smith and Sons</div>
  <a class="email" href="mailto:smith@example.com">smith@example.com</a>
  <div class="adr">
    <div class="street-address">123 Nice Street</div>
    <span class="locality">Adelaide</span>,
    <span class="region">SA</span>,
    <span class="postal-code">5000</span>
    <span class="country-name">Australia</span>
  </div>
  <div class="tel">+61812345678</div>
</div>
```

The following hCard elements are singular and can be provided just once: `fn`, `n`, `bday`, `tz`, `geo`, `sort-string`, `uid`, `class`, and `rev`. All other properties are allowed to have multiple instances. Generally, the visible property values of markup elements represent the value of the hCard property. However, there are some exceptions. For hyperlinks that are represented by the `a` element for one or multiple hCard properties, the `href` attribute provides the property value for all properties with a URL value (for example, `photo`). In case the `img` element is used, the `src` attribute holds the property value for all properties with a URL value. For `object` elements, the data attribute provides the property value. The content of the element is the property

[4] If `n` is omitted but `fn` is present, the value of `n` will be equal to the value of `fn`.

value for all other properties. If the `title` attribute is provided for `abbr` elements with hCard notation, its value is considered as the hCard property instead of the element contents used otherwise.

Although it is easy to create it manually, hCard metadata can be generated by the hCard creator *hCard-o-matic* on the web site of the authors of the specification [10]. You simply fill in a form about the name, organization, country, e-mail, and other contact data, and the software generates the hCard.

To provide additional information, microformats can also be nested. For example, a sport event review might contain not only the review (annotated in hReview) but also personal information (in hCard) at the same time (see Listing 2-12).

Listing 2-12. A Combination of hReview and hCard

```
<link rel="profile" href="http://microformats.org/profile/hreview" />
<link rel="profile" href="http://microformats.org/profile/hcard" />
…
<div class="hreview">
  <h1 class="summary">The Winner Takes It All Review</h1>
  <span class="reviewer vcard">
  by <span class="fn">John Smith</span>, <span class="title">Editor</span> ↩
  at <span class="org">Sport Reviews</span>
  </span>
  Rating: <span class="rating">4.5</span> out of 5.
  <span class="description">A fascinating performance.</span>
</div>
```

The review is described by the hReview microformat (`class="hreview"`). The name of the reviewer is revealed by `span class="reviewer"`. The hCard microformat is nested inside the hReview microformat in order to provide additional information about him/her (a space-separated list of attribute values in `<span class="reviewer vcard">`). The hCard properties describe the name (`fn`), job title (`title`), and organization (`org`) of the reviewer.

rel="license"

There are millions of web resources with some or all rights reserved. Many licenses associated with documents and objects are sophisticated, and users cannot be expected to know them. The `rel="license"` microformat can be added to hyperlinks that point to the description of the license. This is especially useful for images but can be used for any resources. Basic image embeddings apply only the `src` and `alt` attributes on the `img` element, such as in Listing 2-13.

Listing 2-13. A Basic Image Embedding

```
<img src="hotel.jpg" alt="The Palace Hotel" />
```

To declare the image license, the `rel` and `href` attributes should also be used. In the case of the *Creative Commons Attribution-ShareAlike license*, for example, it should be in the form shown in Listing 2-14.

Listing 2-14. Declaring an Image License

```
<link rel="profile" href="http://microformats.org/profile/rel-license" />
…
<img src="hotel.jpg" alt="The Palace Hotel" rel="license" ↩
 ref="http://creativecommons.org/licenses/by-sa/4.0/" />
```

The value of the `href` attribute provides the associated URI of the resource in which the license is described. Some of the most commonly used *license deeds* are [11] as follows:

- Creative Commons Attribution (cc by)

  http://creativecommons.org/licenses/by/4.0/
- Creative Commons Attribution Share Alike (cc by-sa)

  http://creativecommons.org/licenses/by-sa/4.0/
- Creative Commons Attribution No Derivatives (cc by-nd)

  http://creativecommons.org/licenses/by-nd/4.0/
- Creative Commons Attribution Non-Commercial (cc by-nc)

  http://creativecommons.org/licenses/by-nc/4.0/
- Creative Commons Attribution Non-Commercial Share Alike (cc by-nc-sa)

  http://creativecommons.org/licenses/by-nc-sa/4.0/
- Creative Commons Attribution Non-Commercial No Derivatives (cc by-nc-nd)

  http://creativecommons.org/licenses/by-nc-nd/4.0/

You should select a license that matches what you let others do with your work (distribute commercially or noncommercially, remix, tweak, share with proper crediting, alter, and so on).

rel="nofollow"

One value of the `rel` attribute deserves extended attention, because it is often used in *search engine optimization* (SEO). When `rel="nofollow"` is added to a hyperlink, the link destination should not be considered for additional ranking by search engines. This attribute value can be applied if document owners require hyperlinks, without affecting the ranking of their web pages or links to external web sites. For example, if a hyperlink is vital on the web page but its destination page has a very low PageRank (PR), the hyperlink should be provided with `rel="nofollow"`, to avoid search engine penalty.

■ **Note** PageRank is a link analysis algorithm used to assign a numerical weighting to each web page, in order to express its relative importance on a 0–10 scale.

For example, if the index page of `lowprsite.com` has a low PR but you have to link to it because of the content presented there, you can use the `rel="nofollow"` microformat, as shown in Listing 2-15.

Listing 2-15. A Link That Will Not Be Considered by Search Engines While Indexing a Page

```
<link rel="profile" href="http://microformats.org/profile/rel-nofollow" />
…
<a href="http://www.lowprsite.com" rel="nofollow">Low PR site</a>
```

Although it is widely used, there are several open issues about this microformat. The `rel="nofollow"` microformat indicates a behavior rather than a relationship, so the definition is illogical. The name of the microformat does not reflect the real meaning, and it is not a noun. While `rel="nofollow"` was originally introduced to stop comment spam in blogs, using it alone does not prevent spamming attempts to add marketing to a page (only prevent target pages from benefiting through an increased page rank). Finally, many legitimate non-spam links provided as the attribute value of `rel="nofollow"` might be ignored or given reduced weight by search engines, which is an undesirable side effect.

rel="tag"

Unlike other microformats and general `meta` keywords, the `rel="tag"` microformat can be used for visible links. It can be applied on hyperlink elements to indicate that the destination of the link is a general author-designated tag (keyword) for the current page. An example is shown in Listing 2-16.

Listing 2-16. Using `rel="tag"`

```
<link rel="profile" href="http://microformats.org/profile/rel-tag" />
…
<a href="http://www.lesliesikos.com/category/textbooks/" rel="tag">Textbooks</a>
```

Vote Links

Vote Links is an elemental microformat with three possible values on the `rev` attribute of the `a` element: `vote-for`, `vote-against`, and `vote-abstain`. The values are mutually exclusive. Optionally, visible rollovers can be provided by the `title` attribute. Listing 2-17 shows an example.

Listing 2-17. A Vote Links Example

```
<link rel="profile" href="http://microformats.org/profile/vote-links" />
…
<a rev="vote-for" href="http://example.com/thumbsup/" ➥
 title="HTML should be the primary markup language">HTML5</a>
<a rev="vote-against" href="http://example.com/thumbsdown/" ➥
 title="XHTML should be the primary markup language">XHTML5</a>
```

XFN

The very first HTML microformat, XHTML Friends Network (XFN), was introduced in December 2003. XFN was designed by Global Multimedia Protocols Group to express human relationships with simple hyperlinks. XFN is especially useful for brochure-style home pages and blog entries. The name of the person should be provided as the text of the hyperlink (between `<a>` and `</a>`). The personal web site is the target of the hyperlink, in other words, the value of the `href` attribute. All relationship data can be provided by the `rel` attribute on `a` elements. Multiple values are allowed and should be separated by spaces. The friendship type can be `contact`, `acquaintance`, or `friend`. If the person is known personally, it can be expressed by the `met` attribute value of the `rel` attribute. For example, a friend of Leslie Sikos, whom he knows personally, can publish that relationship on his web site by XFN, as shown in Listing 2-18.

Listing 2-18. Link to the Web Site of a Friend

```
<link rel="profile" href="http://gmpg.org/xfn/11" />
…
I am an old friend of <a href="http://lesliesikos.com" rel="friend met">Leslie Sikos</a>.
```

The distance between the residence of the person and that of his friend can be expressed by the `co-resident` and `neighbor` values. Relatives can set to `child`, `parent`, `sibling`, `spouse`, or `kin`. The professional relationships `co-worker` and `colleague` are also supported. Feelings can also be expressed (`muse`, `crush`, `date`, `sweetheart`).

CSS styles can also be added to XFN metadata. For example, friends can be provided in bold and colleagues in italic, with the CSS rules shown in Listing 2-19.

Listing 2-19. Styling XFN

```
a[rel~="friend"] {
  font-weight: bold;
}

a[rel~="colleague "] {
  font-style: italic;
}
```

Although it is easy to create XFN from scratch, XFN creators such as XFN Creator [12] or Exefen [13] might speed up development.

XMDP

XHTML MetaData Profiles (XMDP) metadata is an XHTML-based format for defining metadata profiles that are both machine- and human-readable. XMDP consists of a property definition list, an optional description, and then, if applicable, one or more definition list items. The profile definition list is identified by the `class` (see Listing 2-20).

Listing 2-20. XMDP Profile Definition

```
<dl class="profile">
```

The definition term is identified by the `id` (see Listing 2-21).

Listing 2-21. Definition Term and Data for XMDP

```
<dt id="property1">property1</dt>
<dd>propertydesc</dd>
```

The informatively used `meta` properties `author` and `keywords`, for example, can be defined by XMDP, as shown in Listing 2-22.

Listing 2-22. A Complete XMDP Example

```
<dl class="profile">
  <dt id="author">author</dt>
  <dd>A person who wrote (at least part of) the document.</dd>
  <dt id="keywords">keywords</dt>
  <dd>A comma and/or space separated list of the keywords or keyphrases of the document.</dd>
</dl>
```

Drafts and Future Microformats

You can apply microformats to provide specific metadata on a wide variety of resources. Address information can be described by `adr`. Geographic coordinates (latitude-longitude pairs) can be provided according to the *World Geodetic System* (*WGS*) with the `geo` microformat. hAtom can be used for web syndication. Information about audio recordings can be embedded by using the hAudio microformat. The hListing microformat can be applied for open, distributed listings. Image, video, and audio media components can be described by hMedia. hNews is a microformat to provide news content on web sites. Product descriptions can be expressed in hProduct. Cooking and baking recipes can be described on the Web with hRecipe. Résumés and CVs can be published with hResume. Document reviews can be written in hReview. The `rel="directory"` microdata can indicate that a link destination is a directory listing that refers to the current page. File attachments provided for downloading can be indicated by the `rel="enclosure"` microformat. `rel="home"` provides a hyperlink to the home page of the web site. The `rel="payment"` microformat is an online payment mechanism. The reworking of the `robots meta` tag is the Robot Exclusion Profile. The `xFolk` microformat (stands for *xFolksomony*) was designed for publishing collections of bookmarks. The list goes on, and the Microformats Community welcomes metadata enthusiasts to create new microformats; however, other formats, such as RDFa and HTML5 Microdata, seem to replace microformats.

RDFa

RDFa (RDF in attributes) makes it possible to write RDF triples in the (X)HTML markup, XML, or SVG as attribute values. The full RDFa syntax (RDFa Core) [14] provides basic and advanced features for experts to express complex structured data in the markup, such as human relationships, places, and events. Those who want to express fairly simple structured data in their web documents can use the less expressive *RDFa Lite* [15], a minimal subset of RDFa that is easier to learn and suitable for most general scenarios. RDFa Lite supports the following attributes: `vocab`, `typeof`, `property`, `resource`, and `prefix`. In host languages that authorize the use of the `href` and `src` attributes, they are supported by RDFa Lite too.

A bunch of numbers has a different meaning in a math lesson than in the telephone book, while a word often has a different meaning in a poem than in real life. The meaning of words depends on the context, so in order to make computers understand the field or area (knowledge domain), we have to identify the machine-readable vocabulary that defines the terminology of the domain. In RDFa, the vocabulary can be identified by the `vocab` attribute, the type of the entity to describe is annotated by the `typeof` attribute, and the properties with the `property` attribute (see Listing 2-23).

Listing 2-23. Basic Machine-Readable Annotation of a Person in RDFa

```
<p vocab="http://schema.org/" typeof="Person">
  My name is <span property="name">Leslie Sikos</span> and you can find out more about me ↩
  by visiting <a property="url" href="http://www.lesliesikos.com">my web site</a>.
</p>
```

Once the preceding code is published and indexed, search engines will find the "web site of Leslie Sikos" more efficiently. To uniquely identify this entity on the Web, the `resource` attribute is used (see Listing 2-24). The `resource` attribute is one of the options to set the object of statements, which is particularly useful when referring to resources that are not navigable links, such as the ISBN number of a book.

Listing 2-24. A Unique Identifier of the Entity in RDFa

```
<p vocab="http://schema.org/" typeof="Person" resource="#sikos">
  My name is <span property="name">Leslie Sikos</span> and you can find out more about me ↩
  by visiting <a property="url" href="http://www.lesliesikos.com">my web site</a>.
</p>
```

The vocabulary declaration makes it possible to omit the full URI from each property (`name` refers to `http://schema.org/name`, `url` abbreviates `http://schema.org/url`). However, if you add RDFa annotation for more than one real-world object or person, you can declare the namespace of the vocabulary on the `html` element of your (X)HTML document (e.g., `<html xmlns:foaf="http://xmlns.com/foaf/0.1/" …>`) and associate it with a prefix that can be reused throughout the document. Every time you use a term from the vocabulary declared on the top of your document, you add the prefix followed by a colon, such as `foaf:name`, `schema:url`, etc. Using prefixes is not only handy but sometimes the only way to annotate your markup. For example, if you need terms from more than one vocabulary, additional vocabularies can be specified by the `prefix` attribute (see Listing 2-25). You can refer to any term from your most frequently used vocabulary (defined in the `vocab` attribute value) without the prefix, and terms from your second vocabulary with the prefix you define as the attribute value of the `prefix` attribute, or define them on the `html` element with the `xmlns` attribute followed by the prefix name and the namespace URI.

Listing 2-25. Using the Term "Textbook" from the FaBiO Ontology

```
<p vocab="http://schema.org/" typeof="Person" prefix="fabio: http://purl.org/spar/fabio/" ➥
 resource="#sikos">
  My name is <span property="name">Leslie Sikos</span> and you can find out more about me ➥
  by visiting <a property="url" href="http://www.lesliesikos.com">my web site</a>.
  I am the author of <a property="fabio:Textbook" ➥
  href="http://lesliesikos.com/mastering-structured-data-on-the-semantic-web/">Mastering ➥
  Structured Data on the Semantic Web</a>.
```

To make search engines "understand" that the provided link refers to a textbook of Leslie Sikos, we used the machine-readable definition of "textbook" from the FaBiO ontology. If you need more than one additional vocabulary for your RDFa annotations, you can add them to the attribute value of the `prefix` attribute as a space-separated list.

The most frequently used vocabulary namespaces are predefined in RDFa parsers, so you can omit them in your markup and still be able to use their terms in RDFa annotations (Table 2-7).

Table 2-7. *Widely Used Vocabulary Prefixes Predefined in RDFa [16]*

Prefix	URI	Vocabulary
cc	http://creativecommons.org/ns#	Creative Commons Rights Expression Language
ctag	http://commontag.org/ns#	Common Tag
dcterms	http://purl.org/dc/terms/	Dublin Core Metadata Terms
dc	http://purl.org/dc/elements/1.1/	Dublin Core Metadata Element Set, Version 1.1
foaf	http://xmlns.com/foaf/0.1/	Friend of a Friend (FOAF)
gr	http://purl.org/goodrelations/v1#	GoodRelations
ical	http://www.w3.org/2002/12/cal/icaltzd#	iCalendar terms in RDF
og	http://ogp.me/ns#	Facebook OpenGraph
rev	http://purl.org/stuff/rev#	RDF Review
sioc	http://rdfs.org/sioc/ns#	SIOC Core
v	http://rdf.data-vocabulary.org/#	Google Rich Snippets
vcard	http://www.w3.org/2006/vcard/ns#	vCard in RDF
schema	http://schema.org/	schema.org

More sophisticated annotations require additional attributes that are supported by RDFa Core only. Beyond the RDFa Lite attributes, RDFa Core supports the `about`, `content`, `datatype`, `inlist`, `rel`, and `rev` attributes.

The *current subject* is the web address[5] of the document or a value set by the host language, such as the `base` element in (X)HTML. As a result, any metadata written in a document will concern the document itself by default. The `about` attribute can be used to change the current subject and state what the data is about, making the properties inside the document body become part of a new object rather than referring to the entire document (as they do in the head of the document).

If some displayed text is different from the represented value, a more precise value can be added using the `content` attribute, which is a character data (CDATA) string to supply machine-readable content for a literal. A value can also optionally be typed using the `datatype` attribute (see Listing 2-26). Declaring the type ensures that machines can interpret strings, dates, numbers, etc., rather than considering them as a character sequence.

Listing 2-26. Using the `content` and `datatype` Attributes

```
<html xmlns="http://www.w3.org/1999/xhtml" ➥
  prefix="xsd: http://www.w3.org/2001/XMLSchema# dc: http://purl.org/dc/terms/">
  <head>
    <title>Leslie's Blog</title>
  </head>
  <body>
    <h1 property="dc:title">Leslie's Blog</h1>
    <p>
       Last modified: <span property="dc:modified"
       content="2014-11-28T12:43:00-09:30"
       datatype="xsd:dateTime">28 November 2014</span>.
    </p>
  </body>
</html>
```

In RDFa, the relationship between two resources (predicates) can be expressed using the `rel` attribute (see Listing 2-27).

Listing 2-27. Describing the Relationship Between Two Resources in RDFa

```
This document is licensed under the
<a prefix="cc: http://creativecommons.org/ns#" ➥
   rel="cc:license" ➥
   href="http://creativecommons.org/licenses/by-nc-nd/3.0/">Creative Commons By-NC-ND ➥
   License</a>.
```

When a predicate is expressed using `rel`, the `href` or `src` attribute is used on the element of the RDFa statement, to identify the object (see Listing 2-28).

Listing 2-28. Using `href` to Identify the Object

```
<link about="mailto:leslie@example.com" ➥
      rel="foaf:knows" href="mailto:christina@example.com" />
```

[5]Web address (Uniform Resource Identifier, URI), internationalized web address (Internationalized Resource Identifier, IRI), or compact web address (Compact URI, CURIE)

Reverse relationships between two resources (predicates) can be expressed with the `rev` attribute. The `rel` and `rev` attributes can be used on any element individually or together. Combining `rel` and `rev` is particularly useful when there are two different relationships to express, such as when a photo is taken by the person it depicts (see Listing 2-29).

Listing 2-29. Combining the `rel` and `rev` Attributes

```
<img about="http://www.lesliesikos.com" src="koalahug.jpg" rev="dc:creator" rel="foaf:img" />
```

■ **Caution** If a triple predicate is annotated using `rel` or `rev` only, but no `href`, `src`, or `resource` is defined on the same element, the represented triple will be incomplete [17].

The `inlist` attribute indicates that the object generated on the element is part of a list sharing the same predicate and subject (see Listing 2-30). Only the presence of the `inlist` attribute is relevant; its attribute value is always ignored.

Listing 2-30. Using the `inlist` Attribute

```
<p prefix="bibo: http://purl.org/ontology/bibo/ dc: http://purl.org/dc/terms/" ➥
 typeof="bibo:Website">
  The web site <span property="dc:title">Andrew Peno Graphic and Fine Artist</span> by ➥
   <a inlist="" property="dc:creator" href="http://www.andrewpeno.com">Andrew Peno</a> and ➥
   <a inlist="" property="dc:creator" href="http://www.lesliesikos.com">Leslie Sikos</a>.
</p>
```

RDFa DOM API

RDFa provides a Document Object Model (DOM) Application Programming Interface (API) to extract and utilize structured data from a web page, for advanced user interfaces and interactive applications [18].

HTML5 Microdata

HTML5 Microdata is an HTML5 module defined in a separate specification, extending the HTML5 core vocabulary with attributes for representing structured data [19].

Global Microdata Attributes

HTML5 Microdata represents structured data as a group of *name-value pairs*. The groups are called *items*, and each name-value pair is a *property*. Items and properties are represented by regular elements. To create an item, the `itemscope` attribute is used.[6] To add a property to an item, the `itemprop` attribute is used on a descendant of the item (a child element of the container element), as shown in Listing 2-31.

---

[6]In HTML5, most web designers use attribute minimization and omit the attribute value (even if it is irrelevant), which is not allowed in XHTML5. In other words, in HTML5, you can write `itemscope` on the container element without a value, while in XHTML5 you write `itemscope="itemscope"`, which is more verbose and more precise and validates as HTML5 and XHTML5. The XHTML5 syntax is used throughout the book.

Listing 2-31. A Person's Description in HTML5 Microdata

```
<div itemscope="itemscope" itemtype="http://schema.org/Person">
  <span itemprop="name">Leslie Sikos</span>
  <img src="lesliesikos.jpg" alt="Leslie Sikos" itemprop="image" />
  Leslie's web site:
  <a href="http://www.lesliesikos.com" itemprop="url">lesliesikos.com</a>
</div>
```

Property values are usually strings (sequences of characters) but can also be web addresses, as the value of the `href` attribute on the `a` element, the value of the `src` attribute on the `img` element, or other elements that link to or embed external resources. In Listing 2-31, for example, the value of the image item property is the attribute value of the `src` attribute on the `img` element, which is `lesliesikos.jpg`. Similarly, the value of the `url` item property is not the content of the `a` element, `lesliesikos.com`, but the attribute value of the `href` attribute on the `a` element, which is `http://www.lesliesikos.com`. By default, however, the value of the item is the content of the element, such as the value of the `name` item property in this example: `Leslie Sikos` (delimited by the `<span>` and `</span>` tag pair).

The type of the items and item properties are expressed using the `itemtype` attribute, by declaring the web address of the external vocabulary that defines the corresponding item and properties. In our example, we used the `Person` vocabulary from `http://schema.org` that defines properties of a person, such as `familyName`, `givenName`, `birthDate`, `birthPlace`, `gender`, `nationality`, and so on. The full list of properties is defined at `http://schema.org/Person`, which is the value of the `itemtype`. In the example, we declared the name with the `name` property, the depiction of the person with the `image` property, and his web site address using the `url` property. The allowed values and expected format of these properties are available at `http://schema.org/name`, `http://schema.org/image`, and `http://schema.org/url`, respectively.

The item type is different for each knowledge domain, and if you want to annotate the description of a book rather than a person, the value of the `itemtype` attribute will be `http://schema.org/Book`, where the properties of books are collected and defined, such as `bookFormat`, `bookEdition`, `numberOfPages`, `author`, `publisher`, etc. If the item has a global identifier (such as the unique ISBN number of a book), it can be annotated using the `idemid` attribute, as shown in Listing 2-32.

Listing 2-32. The Description of a Book in HTML5 Microdata

```
<div itemscope="itemscope" itemtype="http://schema.org/Book" ⬅
  itemid="urn:isbn:978-1-484208-84-7">
  <img itemprop="image" src="http://www.masteringhtml5css3.com/img/webstandardsbook.jpg" ⬅
   alt="Web Standards" />
  <span itemprop="name">Web Standards: Mastering HTML5, CSS3, and XML</span> ⬅
  by <a itemprop="author" href="http://www.lesliesikos.com">Leslie Sikos</a>
</div>
```

Although HTML5 Microdata is primarily used for semantical descriptions of people, organizations, events, products, reviews, and links, you can annotate any other knowledge domains with the endless variety of external vocabularies.

Groups of name-value pairs can be nested in a Microdata property by declaring the `itemscope` attribute on the element that declared the property (see Listing 2-33).

Listing 2-33. Nesting a Group of Name-Value Pairs

```
<div itemscope="itemscope">
  <p>Name: <span itemprop="name">Herbie Hancock</span></p>
  <p>Band: <span itemprop="band" itemscope="itemscope">
             <span itemprop="name">The Headhunters</span>
            (<span itemprop="size">7</span> members)
          </span>
  </p>
</div>
```

In the preceding example, the outer item (top-level Microdata item) annotates a person, and the inner one represents a jazz band.

An optional attribute of elements with an `itemscope` attribute is `itemref`,[7] which gives a list of additional elements to crawl to find the name-value pairs of the item. In other words, properties that are not descendants of the element with the `itemscope` attribute can be associated with the item using the `itemref` attribute, providing a list of element identifiers with additional properties elsewhere in the document (see Listing 2-34). The `itemref` attribute is not part of the HTML5 Microdata data model.

Listing 2-34. Using the `itemref` Attribute

```
<div itemscope="itemscope" id="herbie" itemref="a b"></div>
<p id="a">Name: <span itemprop="name">Herbie Hancock</span></p>
<div id="b" itemprop="band" itemscope="itemscope" itemref="c"></div>
<div id="c">
  <p>Band: <span itemprop="name">The Headhunters</span></p>
  <p>Size: <span itemprop="size">7</span> members</p>
</div>
```

The first item has two properties, declaring the name of jazz keyboardist `Herbie Hancock`, and annotates his jazz band separately on another item, which has two further properties, representing the name of the band as `The Headhunters`, and sets the number of members to `7` using the `size` property.

HTML5 Microdata DOM API

HTML5 Microdata has a DOM API for web developers to directly access structured data [20].

JSON-LD

In contrast to RDFa and HTML5 Microdata, the two other mainstream formats to add structured data to the web site markup, *JavaScript Object Notation for Linked Data* (*JSON-LD*) is described as JavaScript code rather than markup elements and attributes. As a result, JSON-LD is completely separate from the (X)HTML code. One of the advantages of this lightweight Linked Data format is that it is easy for humans to read and write. JSON-LD transports Linked Data using the JavaScript Object Notation (JSON), an open standard format using human-readable text to transmit attribute-value pairs [21]. If the JSON-LD code is written in a separate file rather than the markup, the de facto file extension is `.jsonld`. The Internet media

[7]The `itemref` attribute is not part of the Microdata data model and is purely a syntactic construct to annotate web page components for which creating a tree structure is not straightforward, as, for example, a table in which the columns represent items, and the cells the properties.

type of JSON-LD is `application/ld+json` and, if written in the markup, the JSON-LD code is delimited by curly braces between the `<script>` and `</script>` tags, as shown in Listing 2-35.

Listing 2-35. Compact JSON-LD Code in the Markup

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Person",
  "image": "lesliesikos.jpg",
  "name": "Leslie Sikos",
  "url": "http://www.lesliesikos.com"
}
</script>
```

This example uses the compact syntax of JSON-LD, which can be expanded to the full syntax notation demonstrated in Listing 2-36.

Listing 2-36. Expanded JSON-LD Code

```
[
  {
    "@type": [
      "http://schema.org/Person"
    ],
    "http://schema.org/image": [
      {
        "@id": "http://www.lesliesikos.com/images/lesliesikos.jpg"
      }
    ],
    "http://schema.org/name": [
      {
        "@value": "Leslie Sikos"
      }
    ],
    "http://schema.org/url": [
      {
        "@id": "http://www.lesliesikos.com"
      }
    ]
  }
]
```

JSON-LD DOM API

The API of JSON-LD provides a way to transform JSON-LD documents to be more easily consumed by specific applications [22].

GRDDL: XML Documents to RDF

Since valid XML documents comply to a very strict grammar, RDF triples can often be extracted from XML. *Gleaning Resource Descriptions from Dialects of Languages* (*GRDDL*, pronounced as "griddle") is a markup format for transforming XML documents, including XHTML documents (with or without microformats such as hCard or hCalendar) to RDF. These transformations are usually expressed in XSLT, and happen in the following three steps:

1. Source document declaration
2. Link to one or more extractors
3. GRDDL agent extracts RDF from the document

XHTML 1.x documents use the profile attribute on the head element to declare that the document supports GRDDL transformations, while the available transformations are provided as an `.xsl` file (Listing 2-37).

Listing 2-37. An XHTML 1.x Document That Supports GRDDL Transformations

```
<head profile="http://www.w3.org/2003/g/data-view">
<link rel="transformation" href="grddlxfn.xsl" />
```

■ **Caution** The `profile` attribute is not supported in XHTML5.

In XML documents such as the Atom syndication format (used for news feeds) or KML (used to display geographic data in Google Earth and Google Maps), a transformation can be associated with the XML namespace by simply pointing to the namespace (Listing 2-38).

Listing 2-38. An XML Namespace Declaration Pointing to NamespaceTransformation

```
<foo xmlns="http://example.com/1.0/">
```

When the `http://example.com/1.0/` namespace is accessed, it reveals the `namespaceTransformation`, allowing easy deployment of RDF/XML from XML documents.

For XHTML documents that contain microformats, the profile specific to the applied annotations is used. For example, an XHTML document that supports GRDDL and has hCard information has a profile like that shown in Listing 2-39.

Listing 2-39. An XHTML 1.x Document That Supports GRDDL and Contains hCard Information

```
<head profile="http://www.w3.org/2003/g/data-view http://www.w3.org/2006/03/hcard">
```

GRDDL agents can extract all the hCard data from pages that reference the link of profile transformation (Listing 2-40).

Listing 2-40. Profile Transformation Link

```
The RDF data is extracted by <a rel="profileTransformation" ↩
href="hcard2rdf.xsl">this XSL</a> from <a href="http://example.com/hcard/">this hCard</a>.
```

R2RML: Relational Databases to RDF

The majority of dynamic web site contents are powered by relational databases (RDB) such as Microsoft SQL, MySQL, Oracle, IBM DB2, or PostgreSQL. *RDB2RML* (*R2RML*) is a standard for direct mapping of relational databases to RDF [23], making data more accessible on the Semantic Web (see Figure 2-5).

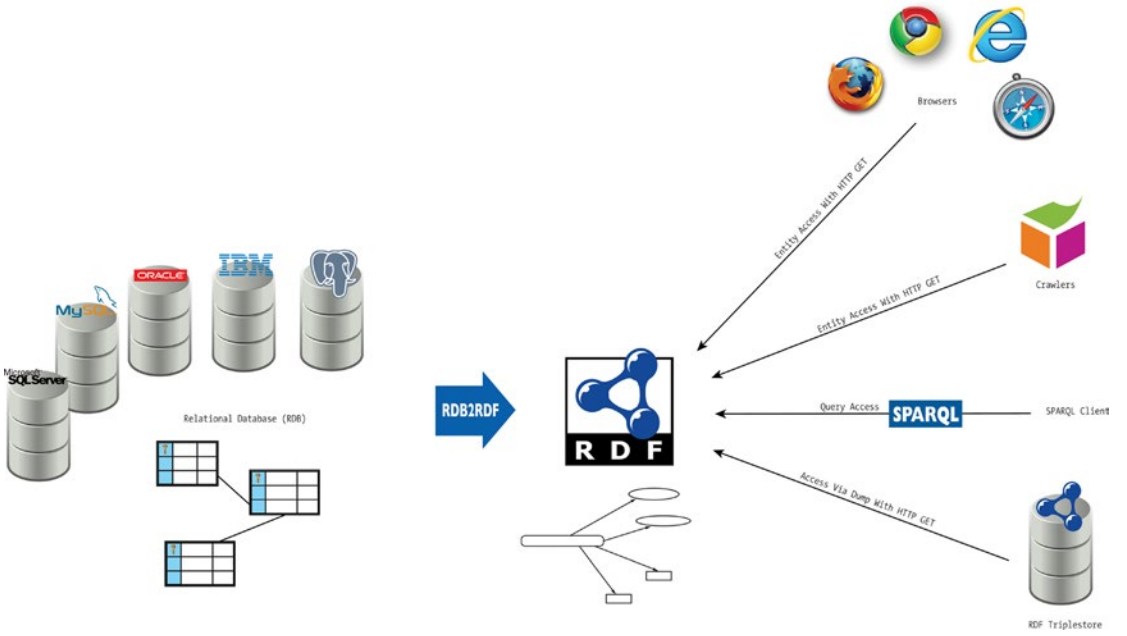

Figure 2-5. *RDB2RML enables RDF benefits for data from relational databases*

The direct mapping represents the RDB data and schema as an RDF graph called a *direct graph* and is described in the Turtle syntax. Assume we have two tables in a relational database, one of which collects people, the other addresses (Listing 2-41).

Listing 2-41. RDB Input

```
CREATE TABLE "Addresses" (
               "ID" INT, PRIMARY KEY("ID"),
                         "city" CHAR(10),
                         "state" CHAR(3)
)

CREATE TABLE "People" (
                "ID" INT, PRIMARY KEY("ID"),
                                "fname" CHAR(10),
                                "addr" INT,
                                FOREIGN KEY("addr") REFERENCES "Addresses"("ID")
)

INSERT INTO "Addresses" ("ID", "city", "state") VALUES (52, 'Adelaide', 'SA')
INSERT INTO "People" ("ID", "fname", "addr") VALUES (5, 'Leslie', 52)
```

Both tables have a unique identifier as the primary key. The address identifier provides the relation between the two tables (Figure 2-6).

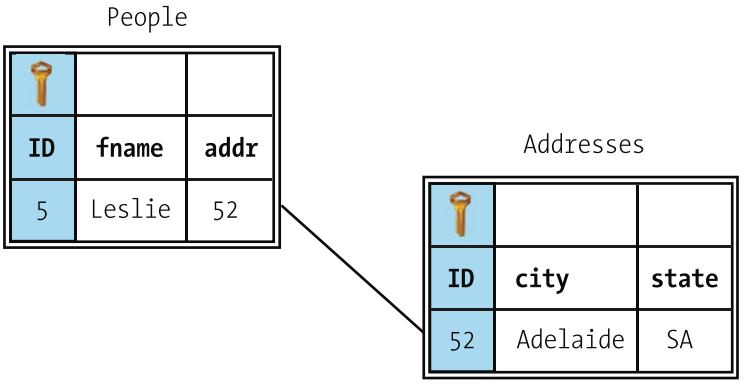

Figure 2-6. *RDB input tables*

The R2RML direct mapping of this example would create a `People` class with a `Leslie` entity with ID `5`, an `Addresses` class with the city and state details of the `Leslie` entity, and a link between the `Leslie` entity and the associated address (Listing 2-42).

Listing 2-42. RDF/Turtle Output

```
@base <http://example.com/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=5> rdf:type <People> .
<People/ID=5> <People#ID> 5 .
<People/ID=5> <People#fname> "Leslie" .
<People/ID=5> <People#addr> 52 .
<People/ID=5> <People#addr> <Addresses/ID=52>
<Addresses/ID=52> rdf:type <Addresses> .
<Addresses/ID=52> <Addresses#ID> 52 .
<Addresses/ID=52> <Addresses#city> "Adelaide" .
<Addresses/ID=52> <Addresses#state> "SA" .
```

RDFS

While RDF is the cornerstone of the Semantic Web, by itself it is not suitable for describing ontologies. RDFS (RDF Vocabulary Description Language, originally the RDF Schema Language) is a simple, RDF-based language for creating RDF ontologies by defining terms of a knowledge domain and the relationships between them [24]. RDFS is an extension of the RDF vocabulary with basic ontology elements and also reuses RDF properties. RDFS ontologies can be represented as RDF graphs. RDFS is suitable for describing various resource types, using specific properties. The RDFS classes and properties form the RDFS vocabulary, including a specialized set of predefined RDF resources with their meaning and using URI references with the prefix `http://www.w3.org/2000/01/rdfschema#` and the associated QName prefix `rdfs:`. The classes of the RDFS vocabulary are used to define a class resource (`rdfs:Resource`), the class of literal values such as strings and integers (`rdfs:Literal`), the class of classes (`rdfs:Class`), the class of RDF datatypes

(rdfs:Datatype), the class of RDF containers (rdfs:Container), and the class of container membership properties (rdfs:ContainerMembershipProperty). The properties of RDFS can express that the subject is a subclass of a class (rdfs:subClassOf), the subject is a subproperty of a property (rdfs:subPropertyOf), define a domain (rdfs:domain) or range of the subject property (rdfs:range), add a human-readable name for the subject (rdfs:label), declare a description of the subject resource (rdfs:comment), identify a member of the subject resource (rdfs:member), add information related to the subject resource (rdfs:seeAlso), and provide the definition of the subject resource (rdfs:isDefinedBy).

Defining RDFS Classes

An RDFS class corresponds to a type or category used for classification and hierarchy. In RDFS, a class C is defined by a triple of the form shown in Listing 2-43, where rdfs:Class is a predefined class and rdf:type is a predefined property.

Listing 2-43. Class Definition in RDFS

```
C rdf:type rdfs:Class .
```

For example, the example.com video rental company wants to use RDFS to provide information about movies, including westerns and comedies. The classes to represent these categories can be written as the statements (triples) shown in Listing 2-44.

Listing 2-44. Statements in RDFS

```
ex:Movie rdf:type rdfs:Class .
ex:Western rdf:type rdfs:Class .
ex:Comedy rdf:type rdfs:Class .
```

Defining RDFS Subclasses

Suppose example.com wants to define that westerns and comedies are movies. This can be done with RDFS subclasses shown in Listing 2-45.

Listing 2-45. Subclass Definition in RDFS

```
ex:Western rdfs:subClassOf ex:Movie .
ex:Comedy rdfs:subClassOf ex:Movie .
```

The rdfs:subClassOf property is reflexive, in other words, once an RDFS class is created, it is a subclass of itself, such as the definition of ex:Movie infers that ex:Movie rdfs:subClassOf ex:Movie . The rdfs:subClassOf property is also transitive. The predefined rdfs:subclassOf property is used as a predicate in a statement to declare that a class is a specialization of another more general class. The meaning of the rdfs:subClassOf predefined property in a statement of the form C1 rdfs:subClassOf C2 is that any instance of class C1 is also an instance of class C2. For example, if we have the statements ex:Comedy rdfs:subClassOf ex:Movie . (comedies are movies) and ex:ActionComedy rdf:type ex:Comedy . (action comedies are comedies), the statement ex:ActionComedy rdf:type ex:Movie . (action comedies are movies) can be inferred (knowledge explicitly not stated can be deducted).

Defining RDFS Instances

To define an instance for example.org, such as an individual movie, we can make an RDF statement that the film *Bad Boys* is an action comedy, as shown in Listing 2-46.

Listing 2-46. Instance Definition in RDFS

```
@prefix films:   <http://example.com/films> .
@prefix moviedb: <http://examplefilmdb.com> .

moviedb:BadBoys rdf:type films:ActionComedy .
```

The `rdf:type` predefined property is used as a predicate in a statement `I rdf:type C .` to declare that individual I is an instance of class C. In statements of the form `C rdf:type rdfs:Class .`, `rdf:type` is used to declare that class C (viewed as an individual object) is an instance of the `rdfs:Class` predefined class. Defining a class explicitly is optional. If we write a triple such as `I rdf:type C .`, C is inferred to be a class (namely, an instance of `rdfs:Class`). A class is not limited to one hierarchical level and can be a subclass or superclass of other classes that is usually represented as a directed graph (see Figure 2-7).

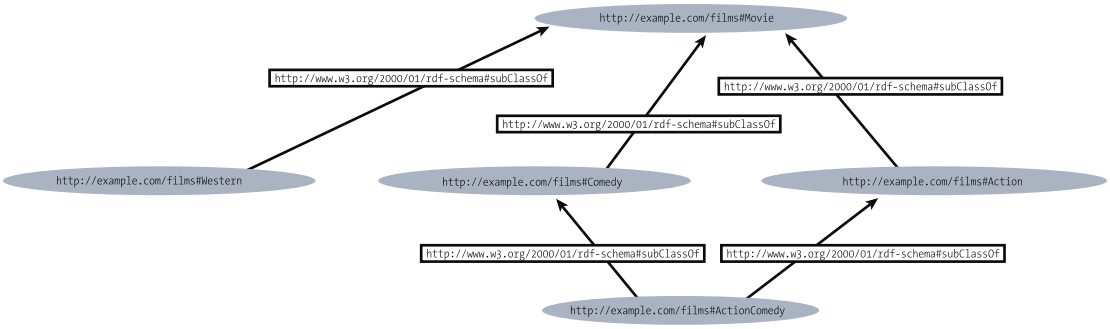

Figure 2-7. *Hierarchy of RDFS classes on a graph*

In our example, the graph represents the machine-readable statements that can be expressed by the ontology (Listing 2-47).

Listing 2-47. RDFS Classes Correspond to Relationships Represented on the Graph

```
@prefix films:   <http://example.com/films> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .

films:Movie rdf:type rdfs:Class .
films:Action rdf:type rdfs:Class .
films:Comedy rdf:type rdfs:Class .
films:Western rdf:type rdfs:Class .
films:ActionComedy rdf:type rdfs:Class .
films:Action rdfs:subClassOf films:Movie .
films:Comedy rdfs:subClassOf films:Movie .
films:Western rdfs:subClassOf films:Movie .
films:ActionComedy rdfs:subClassOf films:Comedy .
films:ActionComedy rdfs:subClassOf films:Action .
```

Defining RDFS Properties

Specific properties can be defined without references to classes or to characterize classes. To create a property for a class, one makes the statement that the property to be defined is an instance of the predefined `rdf:Property` class. For example, we write `ex:author rdf:type rdf:Property .`, so that the `ex:author` property can be used as a predicate in an RDF triple, such as `ex:LeslieSikos ex:author ex:WebStandards .` Because RDFS properties are resources too, properties can be either subjects or objects of triples. For example, `ex:author prov:definedBy ke:LeslieSikos` and `ke:LeslieSikos prov:defined ex:author`.

The `rdfs:label` property is an instance of `rdf:Property` that can be used to provide a human-readable version of the name of a resource. The `rdfs:comment` property is an instance of `rdf:Property` suitable for providing a human-readable description of a resource. A very frequently used RDFS property on the Semantic Web is `rdfs:seeAlso`, which is an instance of `rdf:Property` and used to indicate a resource that provides additional information about the subject resource. Assume we have an RDF description for the textbook *Web Standards: Mastering HTML5, CSS3, and XML*. We declare the title of the book with the title property from the Dublin Core vocabulary, so its namespace at `http://purl.org/dc/terms/` has to be included in the namespace declaration. To link the web site of the book to a web page that describes additional books by the author, the `rdfs:seeAlso property` can be used (see Listing 2-48). Because we use RDF and RDFS properties as well, their namespaces have to be added to the namespace declaration.

Listing 2-48. Tagging, Describing, and Linking Resources with RDFS Properties

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.masteringhtml5css3.com>
    rdfs:label "RDF Description of the web design book Web Standards" ;
    dcterms:title "Web Standards: Mastering HTML5, CSS3, and XML" ;
    rdfs:comment "Web Standards: Mastering HTML5, CSS3, and XML presents step-by-step ↩
    guides based on solid design principles and best practices, and shows the most common ↩
    web development tools and web design frameworks. You will master HTML5 and its XML ↩
    serialization, XHTML5, the new structuring and multimedia elements, the most important ↩
    HTML5 APIs, and understand the standardization process of HTML 5.1, HTML 5.2, and ↩
    future HTML5 versions." ;
    rdfs:seeAlso <http://www.lesliesikos.com/web-design-books/> .
```

The `rdfs:isDefinedBy` property is an instance of `rdf:Property` that is used to indicate a resource that defines the subject resource, such as a controlled vocabulary in which the resource is described.

Defining RDFS Domains and Ranges

Properties can be declared to apply only to certain instances of classes, by defining their domain and range, which indicate the relationships between RDFS classes and properties and RDF data. The `rdfs:domain` predicate indicates that a particular property applies to instances of a designated class (the domain of the property), in other words, declares the class of those resources that may appear as subjects in a triple with the predicate. The `rdfs:range` predicate indicates that the values of a particular property are instances of a designated class (the class of those resources that may appear as the object in a triple with the predicate, also known as the range of the property), as shown in Listing 2-49.

Listing 2-49. Using RDFS Domain and Range

```
ex:Book rdf:type rdfs:Class .
ex:Person rdf:type rdfs:Class .
ex:author rdf:type rdf:Property .
ex:author rdfs:domain ex:Book .
ex:author rdfs:range ex:Person .
Book1 ex:hasAuthor Author .
```

■ **Note** Not all properties have a domain or range.

The `rdfs:range` property can also indicate that a property value is declared with a typed literal[8] (Listing 2-50).

Listing 2-50. Using a Typed Literal

```
ex:age rdf:type rdf:Property .
ex:age rdfs:range xsd:integer .
```

Web Ontology Language (OWL)

While simple machine-readable ontologies can be created using RDFS, complex knowledge domains require more capabilities, such as

- Relations between classes (union, intersection, disjointness, equivalence)
- Property cardinality constraints (minimum, maximum, exact number, e.g., a Person has exactly one father)
- Rich typing of properties (object vs. datatype, specific datatypes)
- Characteristics of properties and special properties (transitive, symmetric, functional, inverse functional, e.g., `A ex:hasAncestor B` and `B ex:hasAncestor C` implies that `A ex:hasAncestor C`)
- Specifying that a given property is a unique key for instances of a particular class
- Domain and range restrictions for properties when they are used with a certain class
- Equality of classes, specifying that two classes with different URI references actually represent the same class
- Equality of individuals, specifying that two instances with different URI references actually represent the same individual
- Enumerated classes

[8]The datatype can also be expressed by `rdfs:Datatype` such as `xsd:integer rdf:type rdfs:Datatype .` or using `rdf:datatype` as for example `rdf:datatype="http://www.w3.org/2001/XMLSchema#string"`.

Web Ontology Language (*OWL*) is a knowledge representation language especially designed for creating web ontologies with a rich set of modeling constructors, addressing the limitations of RDFS. The development of the first version of OWL was started in 2002, and the second version, OWL2, in 2008. OWL became a W3C Recommendation in 2004 [25], and OWL2 was standardized in 2009 [26, 27]. OWL is based on RDF, semantically extending RDF and RDFS, as well as its predecessor language, *DAML+OIL*.

■ **Note** The abbreviation of *Web Ontology Language* is intentionally not WOL but OWL [28].

Description Logic

Ontologies on the Semantic Web often implement *mathematical logic*, a subfield of mathematics dealing with formal expressions, deductive reasoning, and formal proof. *Description Logic* (*DL*) is a family of formal knowledge representation languages in Artificial Intelligence used for logical formalism for ontologies, including formal reasoning of the concepts of a knowledge domain. Description Logic languages are more expressive than *propositional logic* (which deals with declarative propositions and does not use quantifiers) and more efficient in decision problems than *first-order predicate logic* (which uses predicates and quantified variables over non-logical objects). A Description Logic can model concepts, roles and individuals, and their relationships. A core modeling concept of a Description Logic is the *axiom*, which is a logical statement about the relation between roles and/or concepts. Most web ontologies written in OWL are implementations of a Description Logic.

Each Description Logic *Knowledge Base* (*KB*) consists of a *terminological part* (*TBox*) and an *assertional part* (*ABox*), both of which contain a set of axioms. A basic Description Logic is $\mathcal{AL}$, the *Attributive Language*, which supports atomic negation,[9] concept intersection, universal restrictions, and limited existential quantification.

■ **Note** The naming convention of Description Logics is to indicate additional constructors by appending a corresponding letter (see Table 2-8).

Table 2-8. *Common Letters Used in Description Logic Names*

Symbol	Includes	Example
$\mathcal{C}$	Complex concept constructor negation	The negation of arbitrary concepts
$\mathcal{S}$	An abbreviation of $\mathcal{ALC}$ with transitive roles	Apple's mobile operating system is iOS, and iOS is developed for iPhone smartphones, so iPhone smartphones are made by Apple.
$\mathcal{R}$	Limited complex role inclusion axioms, reflexivity and irreflexivity, role disjointness	"part of" and "has part"
$\mathcal{O}$	Enumerated classes of object value restrictions (nominals)	Africa, Antarctica, Asia, Australia, Europe, North America, South America

(*continued*)

[9]Negation of concept names that do not appear on the left-hand side of axioms.

Table 2-8. (*continued*)

Symbol	Includes	Example
$\mathcal{I}$	Inverse properties	Employ and employed by
$\mathcal{N}$	Cardinality restrictions	Each person has two parents.
$\mathcal{F}$	Functional properties, a special case of uniqueness quantification	"there is one and only one"
$\mathcal{Q}$	Qualified cardinality restrictions	Cardinality restrictions that have fillers other than $\top$
$(\mathcal{D})$	Data type properties, data values, or data types	The number annotated as integer in the statement "Christina is 30 years old"

An extension of $\mathcal{AL}$ is the *Attributive Concept Language with Complements*, the Description Logic abbreviated as $\mathcal{ALC}$. $\mathcal{ALC}$ supports ABox expressions such as individual assignments (e.g., Ford is a car), property assignments (e.g., Leslie has a wife, Christina), TBox expressions such as subclass relationships ($\sqsubseteq$) and equivalence ($\equiv$), as well as conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$), property restrictions ($\forall$,$\exists$), tautology ($\top$, a logical formula which is always true), and contradiction ($\bot$). By combining such mathematical operators, you can construct complex class expressions, which are denoted by the *C* in the name of this Description Logic. $\mathcal{ALC}$ can describe sets of individuals, sets of atomic classes, and sets of roles.

$\mathcal{SR}$ extends the capabilities of $\mathcal{ALC}$ with property chains, property characteristics, and role hierarchies. The property characteristics include transitivity (e.g., Ben has the ancestor Violet), symmetry (e.g., Christina is the spouse of Leslie, and Leslie is the spouse of Christina), asymmetry (e.g., Leslie has the son Ben), reflexivity (e.g., Christina has the relative Linda), irreflexive (e.g., Christina is the parent of Ben), functional (e.g., Christina has a husband) and inverse functional properties (e.g., Leslie is the husband of Christina). *SRO* extends $\mathcal{SR}$ with nominals, i.e., enumerated classes of object value restrictions. $\mathcal{SROI}$ adds inverse properties to $\mathcal{SRO}$. $\mathcal{SROIQ}$ extends $\mathcal{SRO}$ with qualified cardinality constraints. $\mathcal{SROIQ}^{(\mathcal{D})}$ extends $\mathcal{SRO}$ $\mathcal{IQ}$ with datatypes, including facets. In addition, $\mathcal{SROIQ}^{(\mathcal{D})}$ supports disjoint properties and adds tautology ($\top$) and contradiction ($\bot$) support for objects and datatypes (see Figure 2-8).

Beyond Abox and TBox, $\mathcal{SROIQ}^{(\mathcal{D})}$ also supports so-called *Role Boxes* (*RBox*) to collect all statements related to roles and the interdependencies between roles. Each RBox consists of a role hierarchy (including generalized role inclusion axioms) and a set of role assertions.

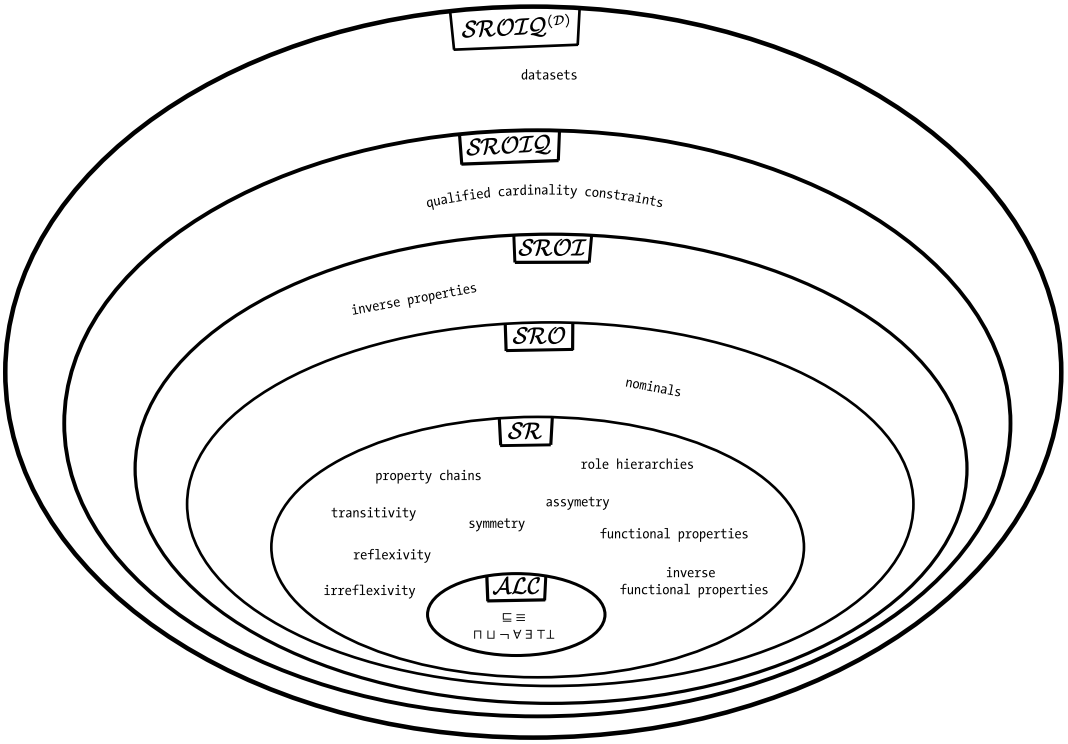

Figure 2-8. *Relationship between the description logic constructors of $\mathcal{ALC}$ and $\mathcal{SROIQ}^{(\mathcal{D})}$*

OWL Variants

There are three flavors of OWL, each constituting different compromises between expressive power and computational complexity (reasoning practicability):

- *OWL-Full*: No restrictions on the use of language constructs: no global restrictions or restrictions for RDF usage. Maximum expressiveness, syntactic freedom, and no computational guarantees. The semantics of OWL-Full is a mixture of RDFS and OWL-DL (*RDF-Based Semantics*).
- *OWL-DL*: A restricted version of OWL-Full that corresponds to a Description Logic. OWL-DL provides maximum expressiveness, computational completeness (all conclusions are guaranteed to be computable), and decidability (all computations can be finished in finite time). It inherits global restrictions from $\mathcal{SROIQ}^{(\mathcal{D})}$. In OWL-DL, RDF can be used only for expressing OWL axioms. OWL-DL implements the model-theoretic semantics of $\mathcal{SROIQ}^{(\mathcal{D})}$ called *OWL2 Direct Semantics*.
- *OWL-Lite*: A subset of OWL-DL designed for easy implementation. OWL-Lite has limited applicability, because it is suitable only for classification hierarchy and simple constraints.

OWL2 provides the expressiveness of the $\mathcal{SROIQ}^{(\mathcal{D})}$ Description Logic; OWL-DL is based on the $\mathcal{SHOIN}^{(\mathcal{D})}$ Description Logic; while OWL-Lite is based on the $\mathcal{SHIF}^{(\mathcal{D})}$ Description Logic.

OWL ontologies are RDF graphs, in other words, sets of RDF triples. Similar to RDF graphs, OWL ontology graphs can be expressed in various syntactic notations. OWL is a higher-level language than RDF; in fact, it is a vocabulary extension of RDF. Consequently, RDF graphs are OWL-Full ontologies. The default OWL namespace is `http://www.w3.org/2002/07/owl#`, which defines the OWL vocabulary. There is no MIME type defined specifically for OWL, but the `application/rdf+xml` or the `application/xml` MIME type is recommended for OWL documents with the `.rdf` or `.owl` file extension.

OWL has three components: classes, properties, and individuals. Classes and individuals are differentiated in OWL using `Class` and `Thing`. While in RDFS only subclasses of existing classes can be created, in OWL, classes can be constructed based on existing classes in any of the following ways:

- Enumerating the content
- Through intersection, union, or complement
- Through property restrictions

Syntaxes

At the high level, the OWL abstract syntax [29] and the OWL2 functional syntax [30] can be used. OWL also supports several exchange syntaxes, including the RDF syntaxes, such as RDF/XML and RDF/Turtle, the OWL2 XML syntax [31], and the Manchester syntax [32], but RDF/XML is the normative syntax.

■ **Note** In the examples, I use declarations for a hypothetical smartphone ontology.

The OWL2 functional syntax is compatible with the *Unified Modeling Language* (*UML*), one of the most widely deployed general-purpose standardized modeling languages (see Listing 2-51). It is clean, adjustable, modifiable, and easy to parse. The functional syntax is primarily used for defining the formal OWL2 grammar in the W3C specifications.

Listing 2-51. OWL2 Functional Syntax Example

```
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)

Ontology(<http://example.com/smartphone.owl>
  Declaration( Class( :Smartphone ) )
)
```

The notational variant of the OWL2 functional syntax is the OWL/XML syntax, which uses an XML tree structure instead of RDF triples, as shown in Listing 2-52.

Listing 2-52. OWL2 XML Syntax Example

```
<Ontology ontologyIRI="http://example.com/smartphone.owl">
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
  <Declaration>
    <Class IRI="Smartphone"/>
  </Declaration>
</Ontology>
```

The only normative syntax of OWL 2 is the RDF/XML syntax (see Listing 2-53). Every OWL2-compliant tool supports this syntax.

Listing 2-53. RDF/XML Syntax Example

```
<rdf:RDF ➥
 xmlns:owl="http://www.w3.org/2002/07/owl#" ➥
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Ontology rdf:about="Phone Ontology"/>
  <owl:Class rdf:about="#Smartphone"/>
</rdf:RDF>
```

A straightforward syntax for representing RDF triples for OWL ontologies is the RDF/Turtle syntax shown in Listing 2-54.

Listing 2-54. RDF/Turtle Example

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

<http://example.com/smartphone.owl>
  rdf:type owl:Ontology .
  :Smartphone rdf:type owl:Class .
```

The less frequently used Manchester syntax is a compact, user-friendly syntax for OWL-DL that collects information about a particular class, property, or individual into a single construct called a *frame*. The Manchester syntax is easy to read and write, especially for those who are not experts in mathematical logic. Complex descriptions consist of short, meaningful English words, while eliminating the logical symbols and precedence rules represented in other syntaxes, as shown in Listing 2-55.

Listing 2-55. Manchester Syntax Example

```
Prefix: owl: <http://www.w3.org/2002/07/owl#>

Ontology: <http://example.com/smartphone.owl>
Class: Smartphone
```

Properties

In OWL, the following types of properties exist:

- *Object properties* that link individuals to other individuals
- *Datatype properties* that link individuals to data values (subclasses of object properties)
- *Annotation property* (`owl:AnnotationProperty`)
- *Ontology property* (`owl:OntologyProperty`)

Property features are defined by the property axioms. The basic form expresses the existence only. For example, in a smartphone ontology, the property `hasTouchscreen` can be declared to express a major feature of mobile phones (see Listing 2-56).

Listing 2-56. A Property Declaration in OWL

```
<owl:ObjectProperty rdf:ID="hasTouchscreen" />
```

OWL property axioms can also define additional characteristics. OWL reuses RDF Schema constructs such as `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range`. Relations to other properties can be expressed by `owl:equivalentProperty` and `owl:inverseOf` (Listing 2-57).

Listing 2-57. Two Equivalent Smartphone Properties (Accelerometer and G-sensor)

```
<owl:ObjectProperty rdf:ID="hasAccelerometer">
  <owl:equivalentProperty>
    <owl:ObjectProperty rdf:ID="hasGsensor" />
  </owl:equivalentProperty>
</owl:ObjectProperty>
```

Global cardinality constraints are defined by `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` (see Listing 2-58). Symmetry and transitivity features are defined by `owl:SymmetricProperty` and `owl:TransitiveProperty` [33].

Listing 2-58. A `FunctionalProperty` in OWL

```
<owl:ObjectProperty rdf:about="&myMobile;manufactured_by">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="&myMobile;Mobile" />
</owl:ObjectProperty>
```

OWL provides precise declarations for expressing relationships, even if they are evident. For example, the *property hierarchy* of two smartphone features can be expressed using `rdfs:subPropertyOf`, as presented in Listing 2-59.

Listing 2-59. Property Hierarchy in OWL

```
<owl:ObjectProperty rdf:ID="hasGeotagging" />
  <owl:ObjectProperty rdf:ID="hasCamera">
  <rdfs:subPropertyOf rdf:resource="hasGeotagging" />
</owl:ObjectProperty>
```

Classes

Similar to RDF, OWL provides classes to group resources. There are six different *class descriptions* in OWL:

1. Class identifier (URI reference). A named instance of `owl:Class`, a subclass of `rdfs:Class`.[10] Listing 2-60 shows an example.

   Listing 2-60. A Class Identifier in OWL

   ```
   <owl:Class rdf:ID="Handheld"/>
   ```

[10]In OWL Lite and OWL DL. In OWL-Full they are equivalent.

2. Set of individuals (instances of a class) defined by the `owl:oneOf` property. For example, the class of smartphones can be declared in the RDF/XML syntax, with the RDF construct `rdf:parseType="Collection"`, as shown in Listing 2-61.

   Listing 2-61. Declaring Class Instances in OWL

   ```
   <owl:Class>
     <owl:oneOf rdf:parseType="Collection">
       <owl:Thing rdf:about="#Touch" />
       <owl:Thing rdf:about="#Type" />
       <owl:Thing rdf:about="#TouchType" />
     </owl:oneOf>
   </owl:Class>
   ```

3. Property restriction: a value constraint or a cardinality constraint (for example, see Listing 2-62).

   Listing 2-62. Property Restrictions in OWL

   ```
   <owl:Restriction>
     <owl:onProperty rdf:resource="hasGPS" />
     <owl:allValuesFrom rdf:resource="#Smartphone" />
   </owl:Restriction>
   ```

4. Intersection of two or more class descriptions. For example, the intersection of the `Smartphone` and the `MadeByApple` classes can be described by `owl:intersectionOf`, stating that iPhones are smartphones made by Apple (see Listing 2-63).

   Listing 2-63. Intersection in OWL

   ```
   <owl:Class rdf:ID="IPhone">
     <owl:intersectionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Smartphone" />
       <owl:Class rdf:about="#MadeByApple" />
     </owl:intersectionOf>
   </owl:Class>
   ```

5. Union of two or more class descriptions.

6. Complement of a class description. The class extension contains exactly those individuals that do not belong to the class extension of the class description that forms the object of the statement. The complement can be described by the `owl:complementOf` property.

Class descriptions can be combined into *class axioms*. Class hierarchy can be expressed by a *subclass axiom* (Listing 2-64).

Listing 2-64. Class Hierarchy in OWL

```
<owl:Class rdf:ID="Slide">
  <rdfs:subClassOf rdf:resource="#smartphone" />
</owl:Class>
```

The equivalence of two classes express that the individuals contained by them are identical. Listing 2-65 shows an example.

Listing 2-65. Equivalent Classes in OWL

```
<owl:Class rdf:about="VirtualKeyboard">
  <owl:equivalentClass rdf:resource="#Softquerty">
</owl>
```

Although individuals can be members of several classes in general, in many cases, memberships are exclusive. For example, smartphones either have a physical keyboard or a virtual keyboard (on the touchscreen). This *class disjointness* can be expressed as shown in Listing 2-66.

Listing 2-66. Class Disjointness in OWL

```
<owl:Class rdf:about="VirtualKeyboard">
  <owl:equivalentClass rdf:resource="#Softquerty" />
  <owl:disjointWith rdf:resource="Keyboard" />
</owl>
```

Simple Knowledge Organization System (SKOS)

Simple Knowledge Organization System (*SKOS*) is a W3C recommendation for representing taxonomies, thesauri, classification schemes, subject-heading systems, and structured controlled vocabularies. Being one of the most frequently implemented Semantic Web standards in industrial applications, SKOS is built upon RDF and RDFS to enable easy publication of controlled vocabularies as linked data. RDF provides interoperability, consistency, and integrity and allows knowledge organization systems to be used in distributed, decentralized metadata applications where metadata are retrieved from multiple resources.

The SKOS standard defines the SKOS data model as an OWL-Full ontology [34]. The elements of the SKOS data model are OWL classes and properties with individual URIs that form the SKOS vocabulary. The classes and properties of SKOS are suitable for representing the common features of thesauri (lists words in groups of synonyms and related concepts). The abstract concepts of SKOS are represented by terms and can be organized in hierarchies using relationships such as `broader` and `narrower` or linked by nonhierarchical (associative) relationships, such as `related`. Further SKOS classes and predicates can be used for basic descriptions (`Concept`, `ConceptScheme`), labeling (`prefLabel`, `altLabel`, `prefSymbol`, `altSymbol`), documentation (`definition`, `scopeNote`, `changeNote`), subject indexing (`subject`, `isSubjectOf`), grouping (`Collection`, `OrderedCollection`), and subject indication (`subjectIndicator`). SKOS also provides some inference rules similar to the RDFS inference rules.

Rule Interchange Format (RIF)

The additional information used to automatically make new discoveries on the Semantic Web are based either on ontologies or on *rule sets*. While ontologies focus on the classification methods by defining classes, subclasses, and relations, rule sets focus on general mechanisms for discovering and generating new relations, based on existing relations. Rule sets are collections of `IF-THEN` constructs called *rules*. If the condition in the `IF` part of the code holds, the conclusion of the `THEN` part of the code is processed. Rules are simplifications of a first-order predicate logic, are relatively easy to implement, and beyond syntax and semantics, and they can express existential quantification, disjunction, logical conjunction, negation, functions, non-monotonicity, and other features.

There are many different rule languages, for example, the *Rule Markup Language* (*RuleML*), an XML approach to represent both forward (bottom-up) and backward (top-down) rules, or the *Semantic Web Rule Language* (*SWRL*), which was introduced as an extension to OWL. Due to the different paradigms, semantics, features, syntaxes, and commercial interests of rule languages, there is a need for *rule exchange*.

The *Rule Interchange Format* (*RIF*) was designed for rule sharing and exchange between existing rule systems, in other words, allowing rules written for one application to be shared and reused in other applications and rule engines while preserving semantics. RIF is a collection of rigorously defined rule languages called *dialects*. The Core Dialect of RIF is a common subset of most rule engines. The Basic Logic Dialect (BLD) adds logic functions, equality and named arguments, and supports the Horn logic (a disjunction of literals with at most one positive literal). The Production Rules Dialect (PRD) provides action with side effects in rule conclusion. RIF has a mapping to RDF.

Reasoning

Description logic-based ontologies are crucial in describing the meaning of web resources and can leverage powerful description logic *reasoning tools* to facilitate machine-processability of semantic web sites. *Reasoning* derives facts that are not expressed explicitly in machine-readable ontologies or knowledge bases. Description logic reasoners implement the *analytic tableau method* (truth tree) for semantic reasoning, which is the most popular proof procedure for formulas of first-order predicate logic. Among other benefits, this makes it possible to determine the satisfiability of formula sets. Reasoners can determine whether a description of the concept is not contradictory, or whether a description is more general than another description. They can check consistency and whether an individual is an instance of a concept or not. Reasoners can retrieve all instances of a particular concept and find the most specific concept individuals belong to. Due to decidability, computational complexity, and the level of formality, automatic processing is not always feasible.

Parsers

Semantic parsing is the process of mapping a natural language sentence into a formal representation of its meaning. A basic form of semantic parsing is the case-role analysis (semantic role labeling), which identifies roles such as source or destination. An advanced semantic parsing represents a sentence in predicate logic or other formal language for automated reasoning.

Summary

In this chapter, you became familiar with the most common controlled vocabularies and ontologies, so that you can identify the suitable vocabularies and ontologies for your projects, in addition to the right classes and properties. You know how to model statements in RDF, represent them as directed graphs, and write them in RDF/XML or Turtle, as well as annotate them in RDFa, Microdata, or JSON-LD.

The next chapter will show you how to create datasets from structured data and link them to other datasets, making your dataset part of the Linked Open Data Cloud.

References

1. DMOZ—the Open Directory Project. www.dmoz.org. Accessed 20 March 2015.
2. Cyganiak, R., Wood, D., Lanthaler, M. (eds.) (2014) RDF 1.1 Concepts and Abstract Syntax. World Wide Web Consortium. www.w3.org/TR/rdf11-concepts/. Accessed 18 January 2015.
3. Gandon, F., Schreiber, G. (eds.) (2014) RDF 1.1 XML Syntax. World Wide Web Consortium. www.w3.org/TR/rdf-syntax-grammar/. Accessed 18 January 2015.
4. Carothers, G., Seaborne, A. (2014) RDF 1.1 N-Triples. A line-based syntax for an RDF graph. World Wide Web Consortium. www.w3.org/TR/n-triples/. Accessed 18 January 2015.
5. Bizer, C., Cyganiak, R. (2014) RDF 1.1 TriG. RDF Dataset Language. World Wide Web Consortium. www.w3.org/TR/trig/. Accessed 18 January 2015.
6. Carroll, J. J., Stickler, P. (2004) RDF Triples in XML. HP Laboratories. www.hpl.hp.com/techreports/2003/HPL-2003-268.pdf. Accessed 18 January 2015.
7. Klyne, G., Carroll, J. J., McBride, B. (eds.) (2014) RDF 1.1 Concepts and Abstract Syntax. World Wide Web Consortium. www.w3.org/TR/rdf11-concepts/. Accessed 18 January 2015.
8. Sindice (2014) Sindice Web Data Inspector. Sindice Ltd. http://inspector.sindice.com. Accessed 18 January 2015.
9. King, R., Çelik, T. (2012) hCalendar Creator. http://microformats.org/code/hcalendar/creator.html. Accessed 20 March 2015.
10. Çelik, T. (2005) hCard Creator. The Microformats Community. http://microformats.org/code/hcard/creator. Accessed 18 January 2015.
11. Casserly, C. et al (eds.) (2015) Licenses. Creative Commons. http://creativecommons.org/about/licenses/. Accessed 14 April 2015
12. Mullenweg, M., Çelik, T. (2004) XFN 1.1 Creator. Global Multimedia Protocols Group. http://gmpg.org/xfn/creator. Accessed 18 January 2015.
13. Mullenweg, M. (2014) Exefen. http://ma.tt/tools/exefen.php/. Accessed 18 January 2015.
14. Adida, B., Birbeck, M., McCarron, S., Herman, I. (eds.) (2013) RDFa Core 1.1—Second Edition. Syntax and processing rules for embedding RDF through attributes. World Wide Web Consortium. www.w3.org/TR/rdfa-core/. Accessed 18 January 2015.
15. Sporny, M. (ed.) (2012) RDFa Lite 1.1. World Wide Web Consortium. www.w3.org/TR/rdfa-lite/. Accessed 18 January 2015.
16. Herman, I. (2014) RDFa Core Initial Context. World Wide Web Consortium. www.w3.org/2011/rdfa-context/rdfa-1.1. Accessed 18 January 2015.
17. Adida, B., Birbeck, M., McCarron, S., Herman, I. (eds.) (2012) Completing incomplete triples. In RDFa Core 1.1. www.w3.org/TR/2012/REC-rdfa-core-20120607/#s_Completing_Incomplete_Triples. Accessed 18 January 2015.

18. Rixham, N., Birbeck, M., Herman, I. (2012) RDFa API. World Wide Web Consortium. www.w3.org/TR/rdfa-api/. Accessed 18 January 2015.

19. Hickson, I. (2013) HTML Microdata. World Wide Web Consortium. www.w3.org/TR/microdata/. Accessed 18 January 2015.

20. Hickson, I. (ed.) (2013) HTML Microdata. World Wide Web Consortium. www.w3.org/TR/microdata/#using-the-microdata-dom-api. Accessed 18 January 2015.

21. Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Lindström, N. (2014) JSON-LD 1.0. World Wide Web Consortium. www.w3.org/TR/json-ld/. Accessed 18 January 2015.

22. Longley, D., Kellogg, G., Lanthaler, M., Sporny, M. (2014) JSON-LD 1.0 Processing Algorithms and API. World Wide Web Consortium. www.w3.org/TR/json-ld-api/. Accessed 18 January 2015.

23. Das, S., Sundara, S., Cyganiak, R. (eds.) (2012) R2RML: RDB to RDF Mapping Language. World Wide Web Consortium. www.w3.org/TR/r2rml/. Accessed 18 January 2015.

24. Brickley, D., Guha, R. V. RDF Schema 1.1. World Wide Web Consortium. www.w3.org/TR/rdf-schema/. Accessed 18 December 2014.

25. Dean, M., Schreiber, G. (eds.), Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) OWL Web Ontology Language Reference. World Wide Web Consortium. www.w3.org/TR/owl-ref/. Accessed 18 January 2015.

26. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., Rudolph, S. (eds.) (2012) OWL 2 Web Ontology Language—Primer 2nd ed. World Wide Web Consortium. www.w3.org/TR/owl-primer/. Accessed 18 January 2015.

27. Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.), Calvanese, D., Carroll, J., De Giacomo, G., Hendler, J., Herman I., Parsia, B., Patel-Schneider, P. F., Ruttenberg, A., Sattler, U., Schneider, M. (2012) OWL 2 Web Ontology Language—Profiles. World Wide Web Consortium. www.w3.org/TR/owl2-profiles/. Accessed 18 January 2015.

28. Herman, I. (2010) "Why OWL and not WOL?" Tutorial on Semantic Web Technologies. World Wide Web Consortium. www.w3.org/People/Ivan/CorePresentations/RDFTutorial/Slides.html#%28114%29. Accessed 18 January 2015.

29. Patel-Schneider, P. F., Horrocks, I. (eds.) (2004) Abstract Syntax. In: OWL Web Ontology Language. Semantics and Abstract Syntax. World Wide Web Consortium. www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html. Accessed 18 January 2015.

30. Motik, B., Patel-Schneider, P. F., Parsia, B. (eds.), Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M. (2012) OWL 2 Web Ontology Language. Structural Specification and Functional-Style Syntax 2nd Ed. World Wide Web Consortium. www.w3.org/TR/owl-syntax/. Accessed 18 January 2015.

31. Motik, B., Parsia, B., Patel-Schneider, P. F. (eds.), Bechhofer, S., Grau, B. C., Fokoue, A., Hoekstra, R. (2012) OWL 2 Web Ontology Language. XML Serialization 2nd Ed. World Wide Web Consortium. www.w3.org/TR/owl-xml-serialization/. Accessed 18 January 2015.

32. Horridge, M., Patel-Schneider, P. F. (2012) OWL 2 Web Ontology Language. Manchester Syntax. World Wide Web Consortium. www.w3.org/TR/owl2-manchester-syntax/. Accessed 18 January 2015.

33. Dean, M., Schreiber, G. (eds.), Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2004) Properties. In: OWL Web Ontology Language Reference. World Wide Web Consortium. www.w3.org/TR/owl-ref/#Property. Accessed 18 January 2015.

34. Miles, A., Bechhofer, S. (2009) SKOS Simple Knowledge Organization System Reference. World Wide Web Recommendation. www.w3.org/TR/skos-reference/. Accessed 18 January 2015.