

## Chapter 3

# Other RDF-Related Technologies: Microformats, RDFa and GRDDL

### 3.1 Introduction: Why Do We Need These?

So far at this point, we have learned the concept of the Semantic Web, and we have learned RDF. Let us think about these two for a moment.

Recall the vision of the Semantic Web is to add meaning into the current Web so machines can understand its contents. Based on what we have learned about RDF, we understand that RDF can be used to express the meaning of a Web document in a machine-processable way. More specifically, for a given Web document, we can create a set of RDF triples to describe its meaning, and somehow indicate to the machine that these RDF statements are created for the machine to understand this document.

Although we are not quite there yet, it is not hard for us to understand the feasibility of this idea. In fact, it is called *semantic markup*, as we will see in later chapters.

However, there is one obvious flaw with this approach: it is simply too complex for most of us. More specifically, to finish this markup process, we have to first create a collection of RDF statements to describe the meaning of a Web document, then put them into a separate file, and finally, we have to somehow link the original Web document to this RDF file. Is there a simpler way of doing all these?

The answer is yes, and that is to use microformats or RDFa. They are simpler since microformats or RDFa constructs can be *directly* embedded into XHTML to convey the meaning of the document itself, instead of collecting them into separate documents.

This in fact plays an important role in the grand plan for the Semantic Web, since a single given Web page is now readable not only by human eyes, but also by machines. A given application that understands microformats or RDFa can perform tasks that are much more complex than those performed by the applications that are built solely based on screen scraping. In fact, in Chap. 10, we will see Google's Rich Snippets, a direct result of microformats or RDFa.

To understand how GRDDL (pronounced “*griddle*”) fits into the picture, think about the semantic information an XHTML page contains when it is embedded with microformats or RDFa constructs. It will be quite useful if we can obtain RDF statements from this XHTML page automatically. GRDDL is a tool that can help us to accomplish this. Once we can do this, the RDF statements harvested from these XHTML pages can be aggregated together to create even more powerful applications.

And these are the reasons why we need microformats, RDFa and GRDDL. If you skip this chapter for now, you can still continue learning the core technology components of the Semantic Web. However, you need to understand this chapter in order to fully understand Chap. 10.

## 3.2 Microformats

### 3.2.1 *Microformats: The Big Picture*

To put it simply, microformats are a way to embed specific semantic data into the HTML content that we have today, so when a given application accesses this content, it will be able to tell what this content is about.

We are all familiar with HTML pages that represent people, so let us start from here. Let us say we would like to use microformats to add some semantic data about people. To do so, we need the so-called `hCard` microformat, which offers a group of constructs you can use to markup the content:

- a root class called `vcard`;
- a collection of properties, such as `fn` (formatted name) and `n` (name), and quite a few others.

We will see more details about `hCard` microformat in the next section. For now, understand that `hCard` microformat can be used to markup the page content where a person is described. In fact, `hCard` microformat is not only used for people, it can also be used to markup content about companies, organizations and places, as we will see in the next section.

Now, what if we would like to markup some other content? For example, some event described in a Web document? In this case, we will need to use the `hCalendar` microformat, which also provides a group of constructs we can use to markup the related content:

- a root class called `vcalendar`;
- a collection of properties, such as `dtstart`, `summary`, `location`, and quite a few others.

By the same token, if we would like to markup a page content that contains a person’s resume, we then need to use the `hResume` microformat. What about

`hRecipe` microformat? Obviously, it is use for adding markups to a page content where a cooking recipe is described.

By now, the big picture about microformats is clear, and we can define microformats as follows:

Microformats are a collection of individual microformats, with each one of them representing a specific domain (such as person, event, location, etc.) that can be described by a Web content page. Each one of these microformats provides a way of adding semantic markup to these Web pages, so that the added information can be extracted and processed by software applications.

With this definition in mind, it is understandable that the microformats collection is always growing: there are existing microformats that cover a number of domains, and for the domains that have not been covered yet, new microformats are created to cover them.

For example, `hCard` microformat and `hCalendar` microformat are stable microformats, while `hResume` microformat and `hRecipe` microformat are still in draft state. In fact, there is a microformats community that is actively working on new microformats. You can always find the latest news from their official Web site,<sup>1</sup> including a list of stable microformats and a list of draft ones that are under discussion.

Finally, notice that microformats are not a W3C standard or recommendation. They are offered by an open community, and are open standards originally licensed under Creative Commons Attribution. They have been placed into the public domain since December 29, 2007.

### 3.2.2 *Microformats: Syntax and Examples*

In this section, we take a closer look at how to use microformats to markup a given Web document. As we discussed earlier, microformats are a collection of individual microformats, and to present each one of them in this chapter is not only impossible but also unnecessary. In fact, understanding one such microformat will be enough; the rest of them are quite similar when it comes to actually using them to markup a page.

With this said, we focus on the `hCard` microformat in this section since at the time of this writing, the `hCard` microformat is one of the most popular and well-established microformats. We will begin with an overview of `hCard` microformat, followed by some necessary HTML knowledge, and as usual, we will learn `hCard` by examples.

---

<sup>1</sup> <http://microformats.org>

3.2.2.1 From vCard to hCard Microformat

hCard microformat has its root in vCard, and can be viewed as a vCard representation in HTML, hence the letter *h* in hCard (HTML vCard). It is therefore helpful to have a basic understanding about vCard.

vCard is a file format standard that specifies how basic information about a person or an organization should be presented, including name, address, phone numbers, e-mail addresses, URLs, etc. This standard was originally proposed in 1995 by the Versit Consortium, which included Apple, AT&T Technologies, IBM and Siemens as its members. In late 1996, this standard was passed on to the Internet Mail Consortium, and since then it has been used widely in address book applications to facilitate the exchange and backup of contact information.

To date, this standard has been given quite a few extensions, but its basic idea remains the same: vCard has defined a collection of properties to represent a person or an organization. Table 3.1 shows some of these properties.

Since this standard was formed before the advent of XML, the syntax is just simple text that contains property–value pairs. For example, my own vCard object can be expressed as shown in List 3.1.

List 3.1 My vCard object

```
BEGIN:VCARD
FN:Liyang Yu
N:Yu;Liyang;;;
URL:http://www.liyangyu.com
END:VCARD
```

First, notice this vCard object has a BEGIN:VCARD element and an END:VCARD element, which mark the scope of the object. Inside the object, the FN property has a value of Liyang Yu, which is used as the display name. The N property represents the structured name, in the order of first, last, middle names, prefixes and suffixes, separated by semicolons. This can be parsed by a given

Table 3.1 Example properties contained in vCard standard

Property name	Property description	Semantic
N	Name	The name of the person, place or thing associated with the vCard object.
FN	Formatted name	The formatted name string associated with the vCard object.
TEL	Telephone	Phone number string for the associated vCard object.
EMAIL	E-mail	E-mail address associated with the vCard object.
URL	URL	A URL that can be used to get online information about the vCard object.

**Table 3.2** Examples of vCard properties mapped to hCard properties

vCard property	hCard properties and subproperties
FN	fn
N	n with subproperties: family-name, given-name, additional-name, honorific-prefix, honorific-suffix
EMAIL	email with subproperties: type, value
URL	url

application in order to understand each component in the person’s name. Finally, URL is the URL of the Web site that provides more information about the vCard object.

With understanding of the vCard standard, it is much easier to understand the hCard microformat, since it is built directly on the vCard standard. More specifically, the properties supported by the vCard standard are mapped directly to the properties and subproperties contained in the hCard microformat, as shown in Table 3.2.

Notice Table 3.2 does not include all the property mappings, which you can find on the microformats’ official Web site (see Sect. 3.2.1). As a high-level summary, hCard properties can be grouped into six categories:

- personal information properties: this includes properties such as fn, n, nickname, etc.;
- address properties: this includes properties such as adr, with subproperties such as street-address, region and postal-code, etc.;
- telecommunication properties: this includes properties such as email, tel and url, etc.;
- geographical properties: this includes properties such as geo, with subproperties such as latitude and longitude;
- organization properties: this includes properties such as logo, org, with subproperties such as organization-name and organization-unit;
- annotation properties: this include properties such as title, note and role, etc.

With the above mapping in place, the next issue is to represent a vCard object (contained within BEGIN:VCARD and END:VCARD) in hCard microformat. To do so, hCard microformat uses a root class called vcard, and in HTML content, an element with a class name of vcard is itself called an hCard.

Now, we are ready to take a look at some examples to understand how exactly we can use the hCard microformat to markup some page content.

3.2.2.2 Using hCard Microformat to Markup Page Content

Let us start with a very simple example. Suppose that in one Web page, we have some HTML code as shown in List 3.2:

**List 3.2** Example HTML code without hCard microformat markup

```
... <!-- other HTML code -->
<div>
  <a href="http://www.liyangyu.com/">Liyang Yu</a>
</div>
... <!-- other HTML code -->
```

Obviously, for our human eyes, we understand that the above link is pointing to a Web site that describes a person named Liyang Yu. However, any application that sees this code does not really understand that, except for showing a link on the screen as follows:

[Liyang Yu](http://www.liyangyu.com/)

Now let us use the hCard microformat to add some semantic information to this link. The basic rules when doing markup can be summarized as follows:

- Use `vcard` as the class name for the element that needs to be marked up, and this element now becomes an hCard object.
- The properties of an hCard object are represented by elements inside the hCard object. An element with class name taken from a property name represents the value of that property. If a given property has subproperties, the values of these subproperties are represented by elements inside the element for that given property.

Based on these rules, List 3.3 shows one possible markup implemented by using hCard microformat.

**List 3.3** hCard microformat markup added to List 3.2

```
... <!-- other HTML code -->
<div class="vcard">
  <div class="fn">Liyang Yu</div>
  <div class="n">
    <div class="given-name">Liyang</div>
    <div class="family-name">Yu</div>
  </div>
  <div class="url">http://www.liyangyu.com</div>
</div>
... <!-- other HTML code -->
```

This markup is not hard to follow. For example, the root class has a name given by `vcard`, and the property names are used as class names inside it. And certainly,

this simple markup is able to make a lot of difference to an application: any application that understands the hCard microformat will be able to understand the fact that this is a description of a person, with the last name, first name and URL given.

If you open up List 3.3 using a browser, you will see it is a little bit different from the original look and feel. Instead of a clickable name, it actually shows the full name, first name, last name and the URL separately. So let us make some changes to our initial markup, without losing the semantics, of course.

First, a frequently used trick when implementing markup for HTML code comes from the fact that `class` (also including `rel` and `rev` attributes) attribute in HTML can actually take a space-separated list of values. Therefore, we can combine `fn` and `n` to reach something as shown in List 3.4:

### List 3.4 An improved version of List 3.3

```
... <!-- other HTML code -->
<div class="vcard">
  <div class="n fn">
    <div class="given-name">Liyang</div>
    <div class="family-name">Yu</div>
  </div>
  <div class="url">http://www.liyangyu.com</div>
</div>
... <!-- other HTML code -->
```

This is certainly some improvement; at least we don't have to encode the name twice. However, if you open up List 3.4 in a browser, it still does not show the original look. To go back to its original look, we at least need to make use of element `<a>` together with its `href` attribute.

In fact, microformats do not force the content publishers to use specific elements; we can choose any element and use it together with the `class` attribute. Therefore, List 3.5 is our best choice:

### List 3.5 Final hCard microformat markup for List 3.2

```
... <!-- other HTML code -->
<div class="vcard">
  <a class="n fn url" href="http://www.liyangyu.com">
    <span class="given-name">Liyang</span>
    <span class="family-name">Yu</span>
  </a>
</div>
... <!-- other HTML code -->
```

And this is it: if you open up List 3.5 from a Web browser, you get exactly the original look and feel. And certainly, any application that understands hCard microformat will be able to understand what a human eye can see: this is a link to a Web page that describes a person, whose last name is Yu, and first name is Liyang.

List 3.6 is another example of using hCard microformat. It is more complex and certainly more interesting. We present it here so you can get more understanding about using the hCard microformat to markup content files.

**List 3.6** A more complex hCard microformat markup example

```
<div id="hcard-liyang-yu" class="vcard">
  <a class="n fn url" href="http://www.liyangyu.com">
    <span class="given-name">Liyang</span>
    <span class="family-name">Yu</span>
  </a>
  <div class="org">Delta Air Lines</div>
  <div class="tel">
    <span class="type">work</span>
    <span class="value">404.773.8994</span>
  </div>
  <div class="adr">
    <div class="street-address">1030 Delta Blvd.</div>
    <span class="locality">Atlanta</span>,
    <span class="region">GA</span>
    <span class="postal-code">30354</span>
    <div class="country-name">USA</div>
  </div>
  <a class="email" href="mailto:liyang.yu@delta.com">
    liyang.yu@delta.com
  </a>
</div>
```

And List 3.7 shows the result rendered by a Web browser.

**List 3.7** Rendering result of List 3.6

[Liyang Yu](#)  
 Delta Air Lines  
 work 404.773.8994  
 1030 Delta Blvd.  
 Atlanta, GA 30354  
 USA  
[liyang.yu@delta.com](mailto:liyang.yu@delta.com)

### 3.2.3 *Microformats and RDF*

At this point, we have learned the `hCard` microformat. With what you have learned here, it is not hard for you to explore other microformats on your own.

In this section, we will first summarize the benefits offered by microformats, and more importantly, we will also take a look of the relationship between microformats and RDF.

#### 3.2.3.1 What's So Good About Microformats?

First off, microformats do not require any new standards; instead, they leverage existing standards. For example, microformats reuse HTML tags as much as possible, since almost all the HTML tags allow `class` attributes to be used.

Second, the learning curve is minimal for content publishers, who continue to mark up their Web documents as they normally would. The only difference is that they are now invited to make their documents more semantically rich by using `class` attributes with standardized properties values, such as those from the `hCard` microformat as we have discussed.

Third, the added semantic markup has no impact on the document's presentation, if it is done right.

Last, and perhaps most important, is the fact that this small change in the markup process brings a significant change to the whole Web world. The added semantic richness can be utilized by different applications, since applications can start to understand at least part of the document on the Web now. We will see some exciting applications in Chaps. 10 and 11, and it is also possible that at the time you are reading this book, more applications built upon microformats have become available to us.

With this said, how are microformats related to RDF? Do we still need RDF at all? Let us answer these questions in the next section.

#### 3.2.3.2 Microformats and RDF

Obviously, the primary advantage microformats offer over RDF is that we can embed metadata directly in the XHTML documents. This not only reduces the amount of markup we need to write, but also provides one single content page for both human readers and machines. The other advantage of microformats is that microformats have a simple and intuitive syntax, therefore they do not need much of a learning curve compared to RDF.

However, microformats were not designed to cover the same scope as RDF was, and they simply do not work on the same exact level. To be more specific, the following are something offered by RDF, but not by microformats (notice at this

point, you may not be able to fully appreciate all the items in the list, but after you read more of this book, you will be able to):

- RDF does not depend on predefined “formats”, and it has the ability to utilize, share and even create any number of vocabularies.
- With the help from these vocabularies, RDF statements can participate in reasoning processes and new facts can be discovered by machines.
- Resources in RDF statements are represented as URIs, allowing a linked data Web to be created.
- RDF itself is infinitely extensible and open-ended.

You can continue to grow this list once you learn more about microformats and RDF from this book and your real development work. However, understanding microformats is also a must, and this will at least enable you to pick up the right tool for the right situation.

## 3.3 RDFa

### 3.3.1 *RDFa: The Big Picture*

With what we have learned so far, the big picture of RDFa is quite simple to understand: it is just another way to directly add semantic data into XHTML pages. Unlike microformats, which reuse the existing `class` attribute on most HTML tags, RDFa provides a set of new attributes that can be used to carry the added markup data. Therefore, in order to use RDFa to embed the markup data within the Web documents, some attribute-level extensions to XHTML have to be made. In fact, this is also the reason for the name: RDFa means RDF in HTML attributes.

Notice that unlike microformats, RDFa is a W3C standard. More specifically, it became a W3C standard on October 14, 2008, and you can find the main standard document on W3C official Web site.<sup>2</sup> Based on this document, RDFa is officially defined as follows:

RDFa is a specification for attributes to express structured data in any markup language.

Another W3C RDFa document, *RDFa for HTML Authors*,<sup>3</sup> has provided the following definition of RDFa:

RDFa is a thin layer of markup you can add to your web pages that make them understandable for machines as well as people. By adding it, browsers, search engines, and other software can understand more about the pages, and in so doing offer more services or better results for the user.

---

<sup>2</sup> <http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/>

<sup>3</sup> <http://www.w3.org/MarkUp/2009/rdfa-for-html-authors>

Once you have finished Sect. 3.3, you should be able to understand both these definitions better.

### 3.3.2 *RDFa Attributes and RDFa Elements*

First off, attributes introduced by RDFa have names. For example, `property` is one such attribute. Obviously, when we make reference to this attribute, we say `attribute property`. In order to avoid repeating the word `attribute` too often, `attribute property` is often written as `@property`. You will see this a lot if you read about RDFa. And in what follows, we will write `@attributeName` to represent one attribute whose name is given by `attributeName`.

The following attributes are used by RDFa at the time of this writing:

```
about
content
datatype
href
property
rel
resource
rev
role
src
typeof
```

Some of them are used more often than others, as we will discuss in later sections. Before we get into the detail, let us first understand with what XHTML elements these attributes can be used.

The rule is very simple: you can use these attributes with just about any element. For example, you can use them on `div` element, on the `p` element, or even on the `h2` (or `h3`, etc.) element. In real practice, there are some elements that are more frequently used with these attributes.

The first such element is the `span` element. It is a popular choice for RDFa simply because you can insert it anywhere in the body of an XHTML document. `link` and `meta` elements are also popular choices, since you can use them to add RDFa markups to the `head` element of an HTML document. This is in fact one of the reasons why RDFa is gaining popularity: these elements have been used to add metadata to the `head` element for years; therefore, any RDFa-aware software can extract useful metadata from them with only minor modifications needed.

The last frequently used element when it comes to adding RDFa markup into the content is the `a` linking element. With what you have learned about RDF from Chap. 2, it is not hard for you to see the reason here: a linking element actually expresses a relationship between one resource (the one where it is stored) and

another (the resource it links to). In fact, as you will see in the examples, we can always use `@rel` on a link element to add more information about the relationship, and this information serves as the predicate of a triple stored in that a element.

### 3.3.3 *RDFa: Rules and Examples*

In this section we explain how to use RDFa to markup a given content page, and we also summarize the related rules when using the RDFa attributes. We will not cover all the RDFa attributes as listed in Sect. 3.3.2, but what you will learn here should be able to get you far in the world of RDFa if you so desire.

#### 3.3.3.1 **RDFa Rules**

Before we set off to study the usage of each RDFa attribute, let us understand its basic rules first. Notice that at this point, these rules may seem unclear to you, but you will start to understand them better when we start to see more examples.

As we learned in Chap. 2, any given RDF statement has three components: subject, predicate and object. It turns out that RDFa attributes are closely related to these components:

- attributes `rel`, `rev` and `property` are used to represent predicates;
- for attribute `rel`, its subject is the value of `about` attribute, and its object is the value of `href` attribute;
- for attribute `rev`, its subject and object are reversed compared to `rel`: its subject is the value of `href` attribute, and its object is the value of `about` attribute;
- for attribute `property`, its subject is the value of `about` attribute, and its object is the value of `content` attribute.

Now recall the fact that we always have to be careful about the object of a given RDF statement: its object can either take a literal string as its value, or it can use another resource (identified by a URI) as its value. How is this taking effect when it comes to RDFa? Table 3.3 summarizes the rules:

Based on Table 3.3, if the object of an RDF statement takes a literal string as its value, this literal string will be the value of `content` attribute. Furthermore, the subject of that statement is identified by the value of `about` attribute, and the predicate of that statement is given by the value of `property` attribute. If the object of a RDF statement takes a resource (identified by a URI) as its value, the URI will be the value of `href` attribute. Furthermore, the subject of that statement is identified by the value of `about` attribute, and the predicate of that statement is given by the value of `rel` attribute.

Let us see some examples along this line. Assume I have posted an article about the Semantic Web on my Web site. In that post, I have some simple HTML code as shown in List 3.8.

**Table 3.3** RDFa attributes as different components of a RDF statement

Object values	Subject attribute	Predicate attribute	Object
Literal strings	about	property	Value of content attribute
Resource (identified by URI)	about	rel	Value of href attribute

**List 3.8** Some simple HTML code in my article about the Semantic Web

```
<div>
  <h2>This article is about the Semantic Web and written by
  Liyang.</h2>
</div>
```

This can be easily understood by a human reader of the article. First, it says this article is about the Semantic Web; second, it says the author of this article is Liyang. Now I would like to use RDFa to add some semantic markup so machines can see these two facts. One way to do this is shown in List 3.9.

**List 3.9** Use RDFa to markup the content HTML code in List 3.8

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/">
  <p>This article is about <span about="http://www.liyangyu.com
/article/theSemanticWeb.html" rel="dc:subject" href="http://dbpe-
dia.org/resource/Semantic_Web"/>the Semantic Web and written by
<span about="http://www.liyangyu.com/article/theSemanticWeb.html"
property="dc:creator" content="Liyang"/>Liyang.</p>
</div>
```

Recall dc represents the Dublin Core vocabulary namespace (review Chap. 2 for more understanding about Dublin Core). We can pick up the RDFa markup segments from List 3.9, and show them in List 3.10:

**List 3.10** RDFa markup text taken from List 3.9

```
<span about="http://www.liyangyu.com/article/theSemanticWeb.html"
rel="dc:subject"
href="http://dbpedia.org/resource/Semantic_Web"/>

<span about="http://www.liyangyu.com/article/theSemanticWeb.html"
property="dc:creator" content="Liyang"/>
```

Clearly, in the first `span` segment, the object is a resource identified by a URI. Therefore, `@rel` and `@href` have to be used as shown in List 3.10. Notice [http://dbpedia.org/resource/Semantic\\_Web](http://dbpedia.org/resource/Semantic_Web) is used as the URI identifying the object resource. This is an URI created by DBpedia project (discussed in Chap. 8) to represent the concept of the Semantic Web. Here we are reusing this URI instead of inventing our own. To see more details about reusing URIs, review Chap. 2.

On the other hand, in the second `span` segment, the object is represented by a literal string. Therefore, `@property` and `@content` have to be used.

The last rule we need to discuss here is about attribute `about`. At this point, we understand attribute `about` is used to represent the subject of the RDF statement. But for a given XHTML content marked up by RDFa, how does an RDFa-aware application exactly identify the subject of the markup? This can be summarized as follows:

- If attribute `about` is used explicitly, then the value represented by `about` is the subject.
- If a RDFa-aware application does not find `about` attribute, it will assume the `about` attribute on the nearest ancestor element represents the subject.
- If an RDFa-aware application searches through all the ancestors of the element with RDFa markup information, and does not find an `about` attribute, then the subject is an empty string, and effectively indicates the current document.

These rules about subject are in fact quite intuitive, especially the last one, given that lots of a document's markup information will be typically about the document itself.

With all this understanding about RDFa rules, we can now move on to the example of RDFa markup.

### 3.3.3.2 RDFa Examples

In this section, we use examples to show how semantic markup information can be added by using RDFa attributes. Notice we will be able to cover only a subset of ways to add RDFa metadata in an XHTML document, it is however enough to get you far if you decide to explore more on yourself.

A common usage of RDFa attributes is to add *inline* semantic information. This is in fact the original motivation that led to the creation of RDFa: how to take human-readable Web page content and make it machine-readable. List 3.9 is a good example of this inline markup. You can compare List 3.8 with List 3.9; List 3.8 is the original page content that is written for human-eyes, and List 3.9 is what we have after inline RDFa markup. Notice the presentation rendered by any Web browser is not altered at all.

Another example is to markup the HTML code shown in List 3.2. It is a good exercise for us since we have already marked up List 3.2 using `hCard`

microformats, and using RDFa to markup the same HTML content shows the difference between the two.

List 3.11 shows the RDFa markup of List 3.2. It accomplishes the same goal as shown in List 3.5. It tells an RDFa-aware application the following: this is a link to the homepage of a person whose first name is Liyang, and whose last name is Yu.

**List 3.11** RDFa markup for the HTML code shown in List 3.2

```
... <!-- other HTML code -->
<div xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <a about="http://www.liyangyu.com#liyang"
    rel="foaf:homepage"
    href="http://www.liyangyu.com/">Liyang Yu</a>
  <span property="foaf:firstName" content="Liyang"/>
  <span property="foaf:lastName" content="Yu"/>
</div>
... <!-- other HTML code -->
```

Again, if you open up the above with a Web browser, you see the same output as given by List 3.2. With what we have learned so far, understanding List 3.11 should not be difficult at all.

Notice that FOAF vocabulary is used for RDFa to markup the content; we covered FOAF briefly in Chap. 2, and you will see a detailed discussion of FOAF in Chap. 7. For now, just remember that FOAF is a vocabulary, with a collection of words that one can use to describe people and their basic information.

This is in fact an important difference between microformats and RDFa. More specifically, when using microformats to markup a given document, the possible values for the properties are predefined. For example, if `hCard` microformat is used, only `hCard` properties and subproperties can be used in the markup (see List 3.5 for example). However, this is not true for RDFa markup: you can in fact use anything as the values for the attributes. For example, List 3.11 could have been written as shown in List 3.12.

**List 3.12** Another version of List 3.11

```
... <!-- other HTML code -->
<div xmlns:yu="http://www.liyangyu.com/yu">
  <a about="http://www.liyangyu.com#liyang"
    rel="yu:myHomepage"
    href="http://www.liyangyu.com/">Liyang Yu</a>
  <span property="yu:myFirstName" content="Liyang"/>
  <span property="yu:myLastName" content="Yu"/>
</div>
... <!-- other HTML code -->
```

However, this is not a desirable solution at all. In order for any RDFa-aware application to understand the markup in List 3.12, that application has to understand your vocabulary first. And clearly, if all the Web publishers went ahead and invented their own keywords, the world of available keywords would have become quite messy. Therefore, it is always the best choice to use words from a well-recognized vocabulary when it comes to markup your page. Again, FOAF vocabulary is one such well-accepted vocabulary, and if you use it in your markup (as shown in List 3.11), chances are any application that understands RDFa will be able to understand FOAF as well.

In fact, this flexibility of the possible values of RDFa attributes is quite useful for many markup requirements. For example, assume in my Web site, I have the following HTML snippet as shown in List 3.13.

**List 3.13** HTML code about my friend, Dr. Ding

```
... <!-- other HTML code -->
<div>
<p>My friend, Dr. Ding, also likes to play tennis.</p>
</div>
... <!-- other HTML code -->
```

And I would like to markup the code in List 3.13 so that machines will understand these facts: first, I have a friend whose name is Dr. Ding, second, Dr. Ding likes to play tennis.

You can certainly try to use microformats to reach the goal; however, RDFa seems to be quite easy to use, as shown in List 3.14.

**List 3.14** RDFa markup of List 3.13

```
... <!-- other HTML code -->
<div xmlns:foaf="http://xmlns.com/foaf/0.1/">
<p>My friend, <span about="http://www.liyangyu.com#liyang" rel=
"foaf:knows" href="http://www.example.org#ding">Dr.Ding</span>,
also likes to play <span about="http://www.example.org#ding" rel=
"foaf:interest" href="http://dbpedia.org/resource/Tennis">tennis.
</span></p>
<span about="http://www.example.org#ding" property="foaf:title"
content="Dr."/> <span about="http://www.example.org#ding"
property="foaf:lastName" content="Ding"/>
</div>
... <!-- other HTML code -->
```

Again, notice <http://dbpedia.org/resource/Tennis> is used as the URI identifying tennis as a sport. This is also a URI created by the DBpedia project, as you will see in later chapters. We are reusing this URI since it is always good to reuse existing ones. On the other hand, <http://www.example.org#ding> is a URI that we invented to represent Dr. Ding, since there is no URI for this person yet.

An application which understands RDFa will generate the RDF statements as shown in List 3.15 from List 3.14 (expressed in Turtle format).

**List 3.15** RDF statements generated from the RDFa markup in 114

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<http://www.liyangyu.com#liyang>
foaf:knows <http://www.example.org#ding>.
<http://www.example.org#ding>
foaf:interest <http://dbpedia.org/resource/Tennis>.
<http://www.example.org#ding> foaf:title "Dr.".
<http://www.example.org#ding> foaf:lastName "Ding".
```

So far, all the examples we have seen are about inline markup. Sometimes, RDFa semantic markup can also be added about the containing document without explicitly using attribute `about`. Since this is a quite common use case of RDFa, let us take a look at one such example.

List 3.16 shows the markup that can be added to the document header:

**List 3.16** RDFa markup about the containing document

```
<html xmlns:dc="http://purl.org/dc/elements/1.1/">
  <head>
    <meta property="dc:title" content="Liyang Yu's Homepage"/>
    <meta property="dc:creator" content="Liyang Yu"/>
  </head>
  <body>
    <!-- body of the page -->
```

Clearly, there is no `about` attribute used. Based on the RDFa rules we discussed earlier, when no subject is specified, an RDFa-aware application assumes an empty string as the subject, which represents the document itself.

As this point, we have covered the following RDFa attributes: `about`, `content`, `href`, `property` and `rel`. These are all frequently used attributes, and understanding these can get you quite far.

The last attribute we would like to discuss here is attribute `typeof`. It is quite important and useful since it presents a case where a blank node is created. Let us take a look at one example.

Assume on my homepage, I have the following HTML code to identify myself as shown in List 3.17.

**List 3.17** HTML code that identifies me

```
<div>
  <p>Liyang Yu</p>
  <p>E-mail: <a
    href="mailto:liyang@liyangyu.com">liyang@liyangyu.com</a>
  </div>
```

We would now like to use RDFa to markup this part so the machine will understand that this whole `div` element is about a person whose name is Liyang Yu and whose e-mail address is `liyang@liyangyu.com`.

List 3.18 shows this markup:

**List 3.18** RDFa markup of the HTML code shown in List 3.17

```
<div typeof="foaf:Person"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <p property="foaf:name">Liyang Yu</p>
  <p>E-mail: <a rel="foaf:mbox"
    href="mailto:liyang@liyangyu.com">liyang@liyangyu.com</a>
  </div>
```

Notice the usage of attribute `typeof`. More specifically, this RDFa attribute is designed to be used when we need to declare a new data item with a certain type. In this example, this type is the `foaf:Person` type. For now, just understand `foaf:Person` is another keyword from the FOAF vocabulary, and it represents human being as a class called `Person`. Again, you will see more about FOAF vocabulary in a later chapter.

Now, when `typeof` is used as one attribute on the `div` element, the whole `div` element represents a data item whose type is `foaf:Person`. Therefore, once reading this line, any RDFa-aware application will be able to understand this `div` element is about a person. In addition, `foaf:name` and `foaf:mbox` are used with `@property` and `@rel` respectively to accomplish our goal to make machines understand this information, as you should be familiar with by now.

Notice we did not specify attribute about like we have done in the earlier examples. So what would be the subject for these properties then? In fact, attribute `typeof` on the

enclosing `div` does the trick: it implicitly sets the subject of the properties marked up within that `div`. In other words, the name and e-mail address are associated with a new node of type `foaf:Person`. Obviously, this new node does not have a given URI to represent itself; it is therefore a blank node. Again, this is a trick you will see quite often if you are working with RDFa markup, so make sure you are comfortable with it.

The last question before we move on: if this new node is a blank node, how do we use it when it comes to data aggregation? For example, the markup information here could be quite important: it could be some supplemental information about a resource we are interested in. However, without a URI identifying it, how do we relate this information to the correct resource at all?

In this case, the answer is yes, we can indeed relate this markup information to another resource that exists outside the scope of this document. The secret lies in the `foaf:mbox` property. As you will see in Chap. 5, this property is an inverse functional property, and that is how we know which resource should be the subject of this markup information, even the subject itself is represented by a blank node.

### 3.3.4 *RDFa and RDF*

#### 3.3.4.1 What's So Good About RDFa?

In Sect. 3.2.3.1, we discussed the benefits offered by microformats. In fact all these are still true for RDFa, and we can add one more here: RDFa is useful because microformats only exist as a collection of centralized vocabularies. More specifically, what if we want to markup a web page about a resource, for which there is no microformat available to use? In that case, RDFa is always a better choice, since you can in fact use any vocabulary for your RDFa markup.

In this chapter, we only see the Dublin Core vocabulary and the FOAF vocabulary. However, as you will see after you finish more chapters, there are quite a lot of vocabularies out there, covering different domains, and all are available to you when it comes to using RDFa to markup a given page. In fact, you can even invent your own if it is necessary (again, more on this later).

#### 3.3.4.2 RDFa and RDF

At this point in the book, RDFa and RDF can be understood as the same thing. To put it simply, RDFa is just a way of expressing RDF triples inside given XHTML pages.

However, RDFa does make it much easier for people to express semantic information in conjunction with normal web pages. For instance, while there are many ways to express RDF (such as in serialized XML files that live next to standard web pages), RDFa helps machines and humans read exactly the same content. This is one of the major motivations for the creation of RDFa.

It might be a good idea to come back to this topic after you have finish the whole book. By then, you will have a better understanding of the whole picture. For example, having an HTML representation and a separate RDF/XML representation (or N3 and Turtle, etc.) is still a good solution for many cases, where HTTP content negotiation is often used to decide which format should be returned to the client (details in Chap. 9).

## 3.4 GRDDL

### 3.4.1 *GRDDL: The Big Picture*

As we discussed in Sect. 3.1, Gleaning Resource Descriptions from Dialects of Languages (GRDDL) is a way (a markup format, to be more precise) that enables users to obtain RDF triples out of XML documents (called *XML dialects*), in particular XHTML documents. The following GRDDL terminology is important for us to understand GRDDL:

- *GRDDL-aware agent*: a software agent that is able to recognize the GRDDL transformations and run these transformations to extract RDF.
- *GRDDL Transformation*: an algorithm for getting RDF from a source document.

GRDDL became a W3C recommendation on September 11, 2007.<sup>4</sup> In this standard document, GRDDL is defined as the following:

GRDDL is a mechanism for Gleaning Resource Descriptions from Dialects of Languages. The GRDDL specification introduces markup based on existing standards for declaring that an XML document includes data compatible with RDF and for linking to algorithms (typically represented in XSLT), for extracting this data from the document.

You can also find more information about GRDDL from the official Web site of the W3C GRDDL Working Group.<sup>5</sup>

In this section, we take a quick look at GRDDL, and introduce the markup formats needed for extracting markup information created by using microformats and RDFa. What you learn from here will give you enough if you decide to go further into GRDDL.

The last words before we move on: do not bury your semantic markup data in (X)HTML pages. Instead, when you publish a document that contains markup data,

---

<sup>4</sup> <http://www.w3.org/TR/2007/REC-grddl-20070911/>

<sup>5</sup> <http://www.w3.org/2001/sw/grddl-wg/>

do reference GRDDL profiles and/or transformations for their extraction. You will see how to do this in the next two sections.

### 3.4.2 *Using GRDDL with Microformats*

There are a number of ways to reference GRDDL in a document where microformats markup data is added. Referencing GRDDL transformations directly in the head of the HTML document is probably the easiest implementation: only two markup lines are needed.

More specifically, the first thing is to add a `profile` attribute to the `head` element to indicate the fact that this document contains GRDDL metadata. List 3.19 shows how to do this.

#### **List 3.19** Adding a `profile` attribute for GRDDL transformation

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head profile="http://www.w3.org/2003/g/data-view">
  <title>Liyang Yu's Homepage</title>
</head>
<body>
<!-- body of the page -->
```

In HTML, `profile` attribute in `head` element is used to link a given document to a description of the metadata schema that document uses. The URI for GRDDL is given by the following,

<http://www.w3.org/2003/g/data-view>

And by including this URI as shown in List 3.19, we declare that the metadata in the markup can be interpreted using GRDDL.

The second step is to add a `link` element containing the reference to the appropriate transformation. More specifically, recall that microformats are a collection of individual microformats such as the `hCard` microformat and the `hCalendar` microformat. Therefore, when working with markup data added by using microformats, it is always necessary to name the specific GRDDL transformation.

Let us assume the document in List 3.19 contains `hCard` microformat markups. Therefore, the `link` element has to contain the reference to the specific transformation for converting HTML containing `hCard` patterns into RDF. This is shown in List 3.20.

**List 3.20** Adding `link` element for GRDDL transformation (hCard microformat)

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head profile="http://www.w3.org/2003/g/data-view">
  <title>Liyang Yu's Homepage</title>
  <link rel="transformation"
        href="http://www.w3.org/2006/vcard/hcard2rdf.xsl"/>
</head>
<body>
<!-- body of the page -->
```

These two steps are all there is to it: the profile URI tells a GRDDL-aware application to look for a `link` element whose `rel` attribute contains the token `transformation`. Once the agent finds this element, the agent should use the value of `href` attribute on that element to decide how to extract the hCard microformat markup data as RDF triples from the enclosing document.

What if hCalendar microformat markup has been used in the document? If that is the case, we should use the following transformation as the value of `href` attribute:

<http://www.w3.org/2002/12/cal/glean-hcal.xsl>

### 3.4.3 Using GRDDL with RDFa

With what we learned from Sect. 3.4.2, it is now quite easy to use GRDDL with RDFa. The first step is still the same, i.e., we need to add a `profile` attribute to the `head` element, as shown in List 3.19. For the second step, as you have guessed, we will have to switch the transformation itself (List 3.21).

**List 3.21** Adding `link` element for GRDDL transformation (RDFa)

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head profile="http://www.w3.org/2003/g/data-view">
  <title>Liyang Yu's Homepage</title>
  <link rel="transformation"
        href="http://www.w3.org/2001/sw/grddl-wg/td/RDFa2RDFXML.xsl"/>
</head>
<body>
<!-- body of the page -->
```

## 3.5 Summary

This chapter covers the technical details of both microformats and RDFa. GRDDL, a popular markup format which automatically converts microformats and RDFa markup information into RDF triples, is also included.

From this chapter, you should have learned the following main points:

- the concepts of microformats and RDFa, and how they fit into the whole idea of the Semantic Web;
- the language constructs of both microformats and RDFa, and how to markup a given (X)HTML page by using these constructs;
- the advantages and limitations of both microformats and RDFa, and their relationships to RDF;
- the concept of GRDDL, how it fits into the idea of the Semantic Web, and how to use GRDDL to automatically extract markup data from (X)HTML pages.

With all this said, the final goal is for you to understand these technical components, and also to be able to pick the right one for a given development assignment.

A Developer's Guide to the Semantic Web

Yu, L.

2014, XXV, 829 p. 624 illus. in color., Hardcover

ISBN: 978-3-662-43795-7