Chapter 11
Linked Open Data

In Chap. 9 we have studied semantic wiki, where semantic information is manually added to the Web content. In Chap. 10, we have studied DBpedia project, where semantic documents are automatically generated. As we have discussed in Chap. 7, besides annotating the pages manually or generating the markup documents automatically, there is indeed another solution: to create a machine-readable Web all from the scratch.

The idea is simple: if we start to publish machine-readable data, such as RDF documents on the Web, and somehow make all these documents connected to each other, then we will be creating a Linked Data Web that can be processed by machines.

This is the idea behind the Linked Open Data (LOD) project, the topic of this chapter.

11.1 The Concept of Linked Data and Its Basic Rules

In recent years, the concept of *Linked Data*, and the so-called Web of Linked Data, has attracted tremendous attention from both the academic world and real application world. In this section, we will examine the concept of Linked Data and its basic rules. What we will learn here from this section will provide a solid foundation for the rest of this chapter.

11.1.1 The Concept of Linked Data

The concept of Linked Data was originally proposed by Tim Berners-Lee in his 2006 Web architecture note.[1] Technically speaking, Linked Data refers to data published on the Web in such a way that it is machine readable, its meaning is explicitly defined, it is linked to other external datasets, and it can in turn be linked to from external datasets as well. Conceptually, Linked Data refers to a set of best practices for publishing and connecting structured data on the Web.

[1] http://www.w3.org/DesignIssues/LinkedData.html

L. Yu, *A Developer's Guide to the Semantic Web*,
DOI 10.1007/978-3-642-15970-1_11, © Springer-Verlag Berlin Heidelberg 2011

The connection between Linked Data and the Semantic Web is quite obvious: publishing and consuming machine-readable data is the center for both of these concepts. In fact, in recent years, Linked Data and the Semantic Web have become two concepts that are interchangeable. After finishing this chapter, you will reach your own conclusion regarding the relationship between Linked Data and the Semantic Web.

In practice, the basic idea of Linked Data is quite straightforward and can be summarized as follows:

- use the RDF data model to publish structured data on the Web and
- use RDF links to interlink data from different data sources.

Applying these two simple tenets repeatedly leads to the creation of a Web of Data that machine can read and understand. This Web of Data, at this point, can be understood as one realization of the Semantic Web. The Semantic Web, therefore, can be viewed as created by the linked structured data on the Web.

Given the fact that Linked Data is also referred to as the *Web of Linked Data*, it is then intuitive to believe that it must share lots of common traits exhibited by the traditional Web. This is a true intuition, yet for every single one of these traits, the Web of Linked Data is profoundly different from the Web of document.

Let us take a look at this comparison, which will certainly give us more understanding about Linked Data and the Semantic Web. Note that at this point, some of the comparisons may not make perfect sense to you, but rest assured that they will become clear after you have finished the whole chapter.

- On the traditional Web, anyone can publish anything at his/her will, at any time.

  The same is true for the Linked Data Web: anyone, at any time, can publish anything on the Web of Linked Data, except that the published documents have to be RDF documents. In other words, these documents are for machines to use, not for human eyes.

- To access the traditional Web, we use Web browsers.

  The same is true for the Web of Linked Data. However, since the Web of Linked Data is created by publishing RDF documents, we use Linked Data browsers that can understand RDF documents and can follow the RDF links to navigate between different data sources. Traditional Web browsers, on the other hand, are designed to handle HTML documents, and they will not be the best choices when it comes to accessing the Web of Linked Data.

- Traditional Web is interesting since everything on the Web is linked together.

  The same is true for the Web of Linked Data. An important fact, however, is that under the hood, the HTML documents contained by the traditional Web are connected by un-typed hyperlinks. For the Web of Linked Data, rather than simply connecting documents, it uses RDF model to make *typed links* that connect

arbitrary things in the world. The result is that we can then build much smarter applications as we will see in the later part of this chapter.

- Traditional Web can provide structured data which can be consumed by Web-based applications.

  This is especially true with more and more APIs being published by major players on the Web. For example, eBay, Amazon, Google all have published their APIs. Web applications that consume these APIs are collectively named as *mashups*, and they do offer quite impressive Web experiences to their users. On the other hand, under the Web of Linked Data, mashups are called *semantic mashups*, and they can be developed in a much more scalable and efficient way. More importantly, they have the ability to grow dynamically upon unbounded datasets, and that is what makes them much more useful than traditional mashups. Again, details will be covered in later sections.

Before we move on, understand that the technical foundation for the Web of Linked Data is not something we have to create from the ground up. To its very bottom, the Web of Linked Data is a big collection of RDF triples, where the subject of any triple is a URI reference in the namespace of one dataset, and the object of the triple is a URI reference in the namespace of another. In addition, by employing HTTP URIs to identify resources, HTTP protocol as retrieval mechanism and RDF data model to represent resource descriptions, Linked Data is directly built upon the general architecture of the Web – a solid foundation that has been tested for more than 20 years.

Furthermore, what we have learned so far, such as RDF model, RDF Schema, OWL, and SPARQL, all these technical components will find their usages in the world of Linked Data.

11.1.2 How Big Is the Web of Linked Data and the LOD Project

The most accurate way to calculate the size of the Web of Linked Data is to use a crawler to count the number of RDF triples that it has collected when traveling on the Web of Linked Data. This is quite a challenging task, and given the fact that some of the RDF triples are generated dynamically, we therefore have to run the crawler repeatedly in order to get the most recent count.

However, the size of the Web of Data can be estimated based on the dataset statistics collected by the LOD community in the ESW Wiki.[2] According to these statistics, the Web of Data, on 4 May 2010, consists of 13.1 billion RDF triples, which are interlinked by around 142 million RDF links (as of 29 September 2009). Note the majority of these triples are generated by the so-called wrappers, which are utility applications responsible for generating RDF statements from existing

[2] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistic, http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/LinkStatistics

relational database tables, and only a small portion of these triples are generated manually.

The Linking Open Data Community Project has been focusing on the idea and implementation of the Web of Data for the last several years. It was originally sponsored by W3C Semantic Web Education and Outreach Group, and you can find more information about this group from this URL:

```
http://www.w3.org/2001/sw/sweo/
```

For the rest of this chapter, we will mainly examine the Linked Data project from technical perspective; you can always find more information on the project from the following Web sites:

- Linking Open Data project wiki home page:

```
http://esw.w3.org/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
```

- Linked Data at the ESW Wiki page:

```
http://esw.w3.org/topic/LinkedData
```

- Linked Data Community Web site:

```
http://linkeddata.org/
```

11.1.3 The Basic Rules of Linked Data

The basic idea of Linked Data is to use RDF model to publish structured data on the Web and also use RDF links to interlink data from different data sources.

In practice, to make sure the above idea is carefully and correctly followed when constructing the Web of Linked Data, four basic rules are further proposed by Tim Berners-Lee in his 2006 Web architecture note:

Rule 1. Use URIs as names for things.
Rule 2. Use HTTP URIs so that a client (machine or human reader) can look up these names.
Rule 3. When someone looks up a URI, useful information should be provided.
Rule 4. Include links to other URIs, so that a client can discover more things.

The first rule is obvious, and it is also what we have been doing all the time: for a given resource or concept, we should use a unique and universal name to identify it. This simple rule eliminates the following two ambiguities on the traditional Web: (1) same name (word) in different documents can refer to completely different resources or concepts and (2) a given resource or concept can be represented by different names (words) in different documents.

The second rule simply puts one more constraint on the first rule by specifying that not only should we use URIs to represent objects and concepts, but we should also only use HTTP URIs.

The reason behind this rule is quite obvious. To make sure that data publishers can come up with identifiers that are indeed globally unique without involving any centralized management, the easiest way is to use HTTP URIs, since the domain part of these URIs can automatically guarantee their uniqueness. In addition, HTTP URIs naturally suggest to the clients that these URIs can be directly used as a means of accessing information about the resources over the Web.

The third rule further strengthens the second rule: if the client is dereferencing a given URI in a Web browser, there should always be some useful information returned back to the client. In fact, at the early days of the Semantic Web, this was not always true: when a given URI was used in a browser, there might or might not be any information coming back at all. We will see more details on this rule later.

The last rule is to make sure the Linked Data world will grow into a real Web: without the links, it will not be a Web of data. In fact, the real interesting thing happens only when the data are linked together and the unexpected fact is discovered by exploring the links.

Finally, note that the above are just the rules of the Web of Linked Data; breaking these rules does not destroy anything. However, without these rules, the data will not be able to provide anything that is interesting.

Now that we have all the background information and we have also learned all the rules, let us take a detailed look into the world of Linked Data. In the next two sections, we will first study how exactly to publish RDF data on the Web; we will then explore different ways to link these data together on the Web.

11.2 Publishing RDF Data on the Web

RDF data are the building blocks of Linked Data. To publishing RDF data on the Web means to follow these steps:

- identifying things by using URIs;
- choosing vocabularies for RDF data;
- producing RDF statements to describe the things;
- creating RDF links to other RDF datasets; and finally
- serving your RDF triples on the Web.

Let us study each one of them in detail.

11.2.1 Identifying Things with URIs

11.2.1.1 Web Document, Information Resource, and URI

To begin with, URI is not something new, and for most of us, a URI represents a Web document. For example, the following URI:

```
http://www.liyangyu.com/
```

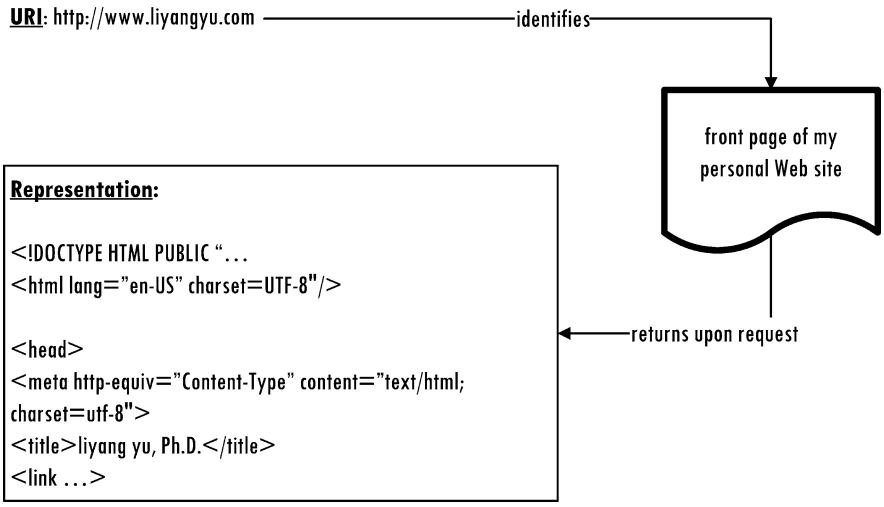

Fig. 11.1 URI/URL for information resource

represents the front page of my personal Web site. This page, like everything else on the traditional Web, is a Web document. We often call the above URI a URL, and as far as Web document is concerned, URL and URI are interchangeable: URL is a special type of URI; it tells us the location of the given Web document. In other words, if a user types in the above URL (URI) into a Web browser, the front page of my Web site will be returned.

Recall that a Web document is defined as something that has a URI and can return representations of the identified resource in response to HTTP requests. The returned representations can take quite a few formats including HTML, JPEG, or RDF, just to name a few.

In recent years, Web documents have a new name: *information resources*. More precisely, everything we find on the traditional document Web, such as documents, images (and other media files) are information resources. In other words, information resources are the resources that satisfy the following two conditions:

- can be identified by URIs;
- can return representations when the identified resources are requested by the users.

Figure 11.1 shows the above concept.

Currently on the Web, to request the representations of a given Web document, clients and servers use HTTP to communicate. For example, the following could be the request sent to the server:

```
GET / HTTP/1.1
Host: www.liyangyu.com
Connection: close
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no
Accept-Language: de,en;q=0.7,en-us;q=0.3
```

And the server will answer with a response header, which tells the client whether the request has been successful, and if successful, the content (representation) will follow the response header.

Let us go back to our basic question in this section: what URIs should we use to identify things in the world? At this point, we can come up with part of the answer: for all the information resources, we can simply use the good old URLs as their URIs to uniquely identify them.

Now, what URIs should we use for the rest of the things (resources) in the world?

11.2.1.2 Non-information Resources and Their URIs

Except for the information resources, the rest of the resources in the world are called *non-information resources*. In general, non-information resources include all the real-world objects that exist outside the Web, such as people, places, concepts, ideas, anything you can imagine, and anything you want to talk about.

To come up with URIs that can be used to identify these non-information resources, there are two important rules proposed by W3C Interest Group.[3] Let us use some examples to understand them.

Let us say I want to come up with a URI to represent myself (a non-information resource). Since I already have a personal Web site, `www.liyangyu.com`, could I then use the following URI to identify myself?

```
http://www.liyangyu.com/
```

This idea is quite intuitive, given the fact that the Web document at the above location does describe me and the URI itself is also unique. However, this clearly confuses a person with a Web document. For any user, the first question that comes to mind will be, does this URI represent this person's home page, or does it represent him as a person?

If we do use the above URI to identify myself, it is then likely that part of my FOAF file would look like the following:

```
<rdf:Description rdf:about="http://www.liyangyu.com/">
   <foaf:name>liyang yu</foaf:name>
   <foaf:title>Dr</foaf:title>
   <foaf:givenname>liyang</foaf:givenname>
   <foaf:family_name>yu</foaf:family_name>
   <foaf:mbox rdf:resource="liyang910@yahoo.com"/>
```

[3] Cool URIs for the Semantic Web, W3C Interest Group Note 03 December 2008 (http://www.w3.org/TR/cooluris/).

Now, if this URI represents my home page, then how could a home page have `foaf:name`, and how could it also have a `foaf:mbox`? On the other hand, if this URI does represent a person named Liyang Yu, then the above FOAF document in general seems to be describing a home page which has a Web address given by `www.liyangyu.com`.

All these said, it seems to be clear that I need another unambiguous URI to represent myself. And this gives the first rule summarized by W3C Interest Group:

> *Be unambiguous*: There should be no confusion between identifiers for Web documents and identifiers for other resources. URIs are meant to identify only one of them, so one URI cannot stand for both a Web and a real-world object.

Now let us say I have already come up with a URI to represent myself, for example,

```
http://www.liyangyu.com/foaf.rdf#liyang
```

then what happens if the above URI is dereferenced in a browser – do we get anything back at all? If yes, what do we get back?

For information resources, we get one possible form of representation back, could be a HTML page, for example. For non-information resources, based on the following rule proposed by W3C Interest Group, when their URIs are used in a browser, related information should be retrieved as follows:

> *Be on the Web*: Given only a URI, machines and people should be able to retrieve a description about the resource identified by the URI from the Web. Such a look-up mechanism is important to establish shared understanding of what a URI identifies. Machines should get RDF data and humans should get a readable representation, such as HTML. The standard Web transfer protocol, HTTP, should be used.

This rule makes it clear that for URIs identifying non-information resources, some descriptions should be returned to the clients. However, it does not specify any details for implementation purpose.

It turns out in the world of Linked Data, the implementation of this rule also dictates how the URIs for non-information resources are constructed. Let us cover the details next.

11.2.1.3 URIs for Non-information Resources: 303 URIs and Content Negotiation

The first solution is to use the so-called *303 URIs* to represent non-information resources. The basic idea is to create a URI for a given non-information resource, and when a client posts a request using this URI, the server will return the special HTTP status code `303 See Other`. This not only indicates the fact that the requested resource is not a regular Web document, but also further redirects the client to some other document which provides information about the thing identified by this URI. By doing so, we will be able to satisfy the above two rules and also avoid the ambiguity between the real-world object and the non-information resource that represents it.

As a side note, if the server answers the request using a status code in the 200 range, such as `200 OK`, it is then clear that the given URI represents a normal Web document or information resource.

Now, in case where a `303 See Other` status code is returned, which document should the server redirect its client to? This depends on the request from the client. If the client is an RDF-enabled browser (or some applications that understands RDF model), it will more likely prefer a URI which points to an RDF document. If the browser is a traditional HTML browser (or the client is a human reader), it will then more likely prefer a URI that points to a HTML document. In other words, when sending the request, the client will include information in the HTTP header to indicate what type of representation it prefers. The server will inspect this header to return a new URI that links to the appropriate response. This process is called *content negotiation*.

It is now a common practice that for a given real-world resource, we can often have three URIs for it. For example, for myself as a non-information resource, the following three URIs will be in use:

- a URI that identifies myself as a non-information resource:

  `http://www.liyangyu.com/resource/liyang`

- a URI that identifies a Web document which has an RDF/XML representation describing myself. This URI will be returned when a client prefers an RDF description:

  `http://www.liyangyu.com/data/liyang`

- a URI identifies a Web document that has a HTML representation describing myself. This URI will be returned when a client prefers a HTML document:

  `http://www.liyangyu.com/page/liyang`

  And the first URI,

  `http://www.liyangyu.com/resource/liyang`

is often the one that is seen by the outside world as my URI.

The above schema for constructing URIs for non-information resources is also viewed as the best practice by the Linked Data community. Another example is the following three URIs about Berlin, as seen in DBpedia project:

- a URI that is used as the identifier for Berlin:

  `http://dbpedia.org/resource/Berlin`

- a URI that identifies a representation in HTML format (for human readers):

  `http://dbpedia.org/page/Berlin`

- a URI that identifies a representation in RDF/XML format (for machines):

  `http://dbpedia.org/data/Berlin`

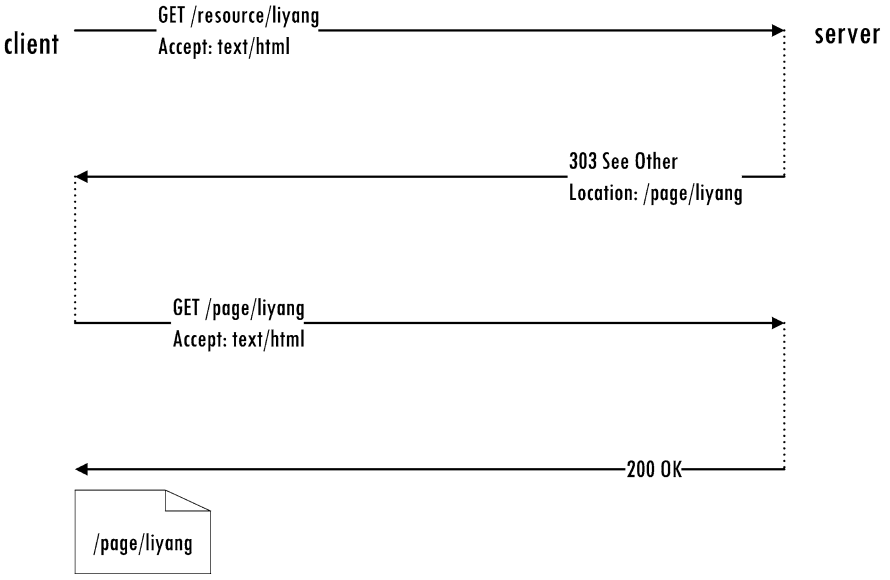

Fig. 11.2 Example of content negotiation

It is now clear that for a given resource, there could be multiple content types for its representation, such as HTML format and RDF/XML format as seen above. Figure 11.2 shows the process of content negotiation, using my own URI as an example.

The following steps show the interaction between the server and a client:

- Client is requesting a HTML Web document:

```
GET /resource/liyang HTTP/1.1
Host: www.liyangyu.com
Accept: text/html
```

- Server's response header should include the following fields:

```
HTTP/1.1 303 See Other
Location: http://www.liyangyu.com/page/liyang
```

- Client is requesting a machine-readable document for the resource:

```
GET /resource/liyang HTTP/1.1
Host: www.liyangyu.com
Accept: application/rdf+xml
```

- Server's response header should include the following fields:

```
HTTP/1.1 303 See Other
Location: http://www.liyangyu.com/data/liyang
```

As a summary, 303 URIs require content negotiation when they are used in a browser to retrieve their descriptions. Furthermore, content negotiation requires at least two HTTP round-trips to the server to retrieve the desired document. However, 303 URIs eliminate the ambiguity between information and non-information resources, therefore they provide a uniform and consistent way of representing resource in the real world.

11.2.1.4 URIs for Non-information Resources: Hash URIs

A *hash URI* is a URI that contains a fragment, i.e., the part that is separated from the rest of the URI by a hash symbol ("#"). For example, the following is a hash URI to identify myself as a resource:

```
http://www.liyangyu.com/foaf.rdf#liyang
```

and `liyang` (to the right of #) is the fragment part of this URI.

Hash URI provides an alternative choice when it comes to identifying non-information resources. The reason behind this solution is related to the HTTP protocol itself.

More specifically, when a hash URI is used in a browser, the HTTP protocol requires the fragment part to be stripped off before sending the URI to the server. For example, if you dereference the above URI into a Web browser and also monitor the request sent out to the server, you will see the following lines in the request:

```
GET /foaf.rdf HTTP/1.1
Host: www.liyangyu.com
```

Clearly, the fragment part is gone. Instead of retrieving this URI,

```
http://www.liyangyu.com/foaf.rdf#liyang
```

the client is in fact requesting this one:

```
http://www.liyangyu.com/foaf.rdf
```

In other words, a URI that includes a hash fragment cannot be retrieved directly, therefore it does not identify a Web document at all. As a result, any URI including a fragment part is a URI that identifies a non-information resource, thus the ambiguity is avoided.

Now that there is no ambiguity associated with a hash URI, what should be served if the URI is dereferenced in a browser? Since we know the fragment part will be taken off by the browser, we can simply serve a document (either human readable or machine readable) at the resulting URI which does not have the fragment part. Again using the following as the example,

```
http://www.liyangyu.com/foaf.rdf#liyang
```

we can then serve an RDF document identified by the URI:

```
http://www.liyangyu.com/foaf.rdf
```

Note that there is no need for any content negotiation, which is probably the main reason why hash URIs look attractive to us.

Hash URI does have its own downside. Consider the following three URIs:

```
http://www.liyangyu.com/foaf.rdf#liyang
http://www.liyangyu.com/foaf.rdf#connie
http://www.liyangyu.com/foaf.rdf#ding
```

which represent three different resources. However, using any one of them in a browser will send a single request to this common URI:

```
http://www.liyangyu.com/foaf.rdf
```

and if someone is only interested in #connie, still the whole document will have to be returned. Obviously, using hash URIs lacks the flexibility of configuring a response for each individual resource.

It is also worth mentioning that even when hash URIs are in used, we can still use content negotiation if we want to serve both HTML and RDF representations for the resources identified by the URIs. For example,

- Client is requesting a HTML Web document for the following resource,

  ```
  http://www.liyangyu.com/foaf.rdf#liyang
  ```

  and you will see these lines in the request:

  ```
  GET /roaf.rdf HTTP/1.1
  Host: www.liyangyu.com
  Accept: text/html
  ```

- Response header from the server should include the following fields:

  ```
  HTTP/1.1 303 See Other
  Location: http://www.liyangyu.com/foaf.html
  ```

  Note that we assume there is a HTML file called `foaf.html` which includes some HTML representations of the given resource.

- Now client is requesting machine-readable document for the resource:

  ```
  GET /roaf.rdf HTTP/1.1
  Host: www.liyangyu.com
  Accept: application/rdf+xml
  ```

- Response header from the server should include the following fields:

  ```
  HTTP/1.1 303 See Other
  Location: http://www.liyangyu.com/foaf.rdf
  ```

  And similarly, the following two hash URIs,

  ```
  http://www.liyangyu.com/foaf.rdf#connie
  http://www.liyangyu.com/foaf.rdf#ding
  ```

will have exactly the same content negotiation process, since their fragment parts will be taken off by the browser before sending them out to the server.

11.2.1.5 URIs for Non-information Resources: 303 URIs vs. Hash URIs

Now that we have introduced both 303 URIs and hash URIs, the next question is about when a 303 URI should be used and when a hash URI should be used. Table 11.1 briefly summarizes the advantages and disadvantages of both URIs.

The following is a simple guideline. Once you have more experience working with the Linked Data and the Semantic Web, you will be able to add more to it:

- For ontologies that are created by using RDF Schema and OWL, it is preferred to use hash URIs to represent all the terms defined in the ontology, and frequent access of this ontology will not generate lots of network redirects.
- If you need a quicker and easier way of publishing Linked Data or small and stable datasets of RDF resource files, hash URI should be the choice.
- Other than the above, 303 URIs should be used to identify non-information resources if possible.

11.2.1.6 URI Aliases

When it comes to identifying things with URIs, one obvious fact we have noticed so far is the lack of centralized control of any kind. In fact, anyone can talk about any resource and come up with a URI to represent that resource. It is therefore quite possible that different users happen to talk about the same non-information resource. Furthermore, since they are not aware of each other's work, they create different URIs to identify the same resource or concept. Since all these URIs are created to identify the same resource or concept, they are called *URI aliases*.

It is commonly suggested that when you plan to publish RDF statements about a given resource, you should try to find at least some of the URI aliases for this resource first. If you can find one or multiple URIs for the resource, by all means reuse one of them, create your own if only you have very strong reason to do so. And in which case, you should use `owl:sameAs` to link it to at least one existing

Table 11.1 303 URI vs. Hash URI: advantages and disadvantages

	303 URI	Hash URI
Advantages	Provides the flexibility of configuring redirect targets for each resource Provides the flexibility of changing/updating these targets easily and freely, at any given time	Does not require content negotiation, therefore reduces the number of HTTP round-trips Since content negotiation is not required, publishing Linked Data is easier and quicker
Disadvantages	Requires two round-trips for each use of a given URI	All the resource descriptions have to be collected in one file

URI. Certainly, you can create your own URI if you cannot find any existing ones at all.

Now, how do you find the URI aliases for the given resource? At the time of this writing, there are some tools available on the Web. Let us use one example to see how these tools can help us.

Assume that we want to publish some RDF statements about Roger Federer, the tennis player who holds the most grand slam titles at current time. Since he is such a well-known figure, it is safe to assume that we are not the first one who would like to say something about him. Therefore, there should be at least one URI identifying him, if not more.

A good starting place where we can search for these URI aliases is the *Sindice* Web site. You can access this Web site here:

```
http://sindice.com/
```

More specifically, Sindice can be viewed as a Semantic Web search engine, and it was originally created at DERI (Digital Enterprise Research Institute) as a research project. Its main idea is to index the Semantic Web documents over the Web, so for a given URI, it can search its datasets and further tell us which dataset has mentioned this given URI.

To us, a more useful feature of Sindice is when searching its datasets, Sindice not only accepts URIs, but also takes keywords. When it accepts keywords, it will find all the URIs that either describe or closely match the given keywords first, then it will locate all the datasets that contain these URIs. This is what we need when we want to know if there are any existing URIs identifying Roger Federer. Figure 11.3 shows the query session.

And Fig. 11.4 shows the Sindice search result.

The first result in Fig. 11.4 shows a URI identifying Roger Federer (we know this by noticing the file type of this result, i.e., an RDF document), and this URI is given as follows:

```
http://dbpedia.org/resource/Roger_Federer
```

To collect other URI aliases identifying Roger Federer, we can continue to use Sindice. However, another tool, called *sameAs*, can also be very helpful. You can find sameAs by accessing its Web site:

```
http://www.sameas.org/
```

It will help us to find the URI aliases for a given URI. In our case, our search is shown in Fig. 11.5, and Fig. 11.6 shows the result:

Clearly, at the time of this writing, there are about 23 URIs identifying Roger Federer, as shown in Fig. 11.6. It is now up to us to pick one of these URIs so we can publish something about Roger Federer.

As you can tell, these two Web sites are very helpful on finding existing URIs. In fact, `www.sameas.org` even provides a link to `www.sindice.com`, as shown in Fig. 11.5. You can either directly enter a URI in the `<sameAs>` box to search

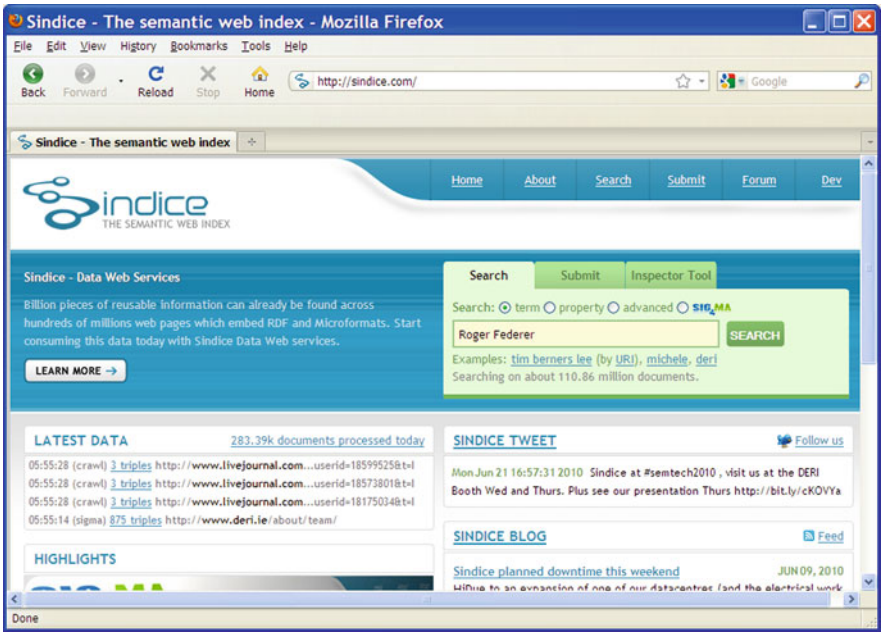

Fig. 11.3 A Sindice search session (search for Roger Federer)

for its URI aliases or you can use Sindice first by entering the keywords in the `Sindice` box.

Recall the lookup service we have discussed in Chap. 10 about DBpedia – it is another service we can use to locate URIs that are created by DBpedia for a given resource. See Fig. 10.4 and Sect. 10.3.3 for details.

At this point, we have briefly discussed about URI aliases. With the development of the Semantic Web, let us hope that there will better and better solutions out there, which will greatly facilitate the reuse of URIs.

11.2.2 Choosing Vocabularies for RDF Data

By now, you should understand that when publishing RDF statements, you should always try to use terms defined in one or more ontologies. For example, the predicate of an RDF statement should always be a URI that comes from the ontologies you are using. In addition, it is recommended that instead of inventing your own ontology, you should always use the terms from well-known existing ontologies. Reusing ontologies will make it possible for clients to understand your data and further process your data, therefore the data you have published can easily become part of the Web of Linked Data.

At this point, there is already a good collection of some well-known ontologies covering multiple application domains. You can find this collection at the Linking

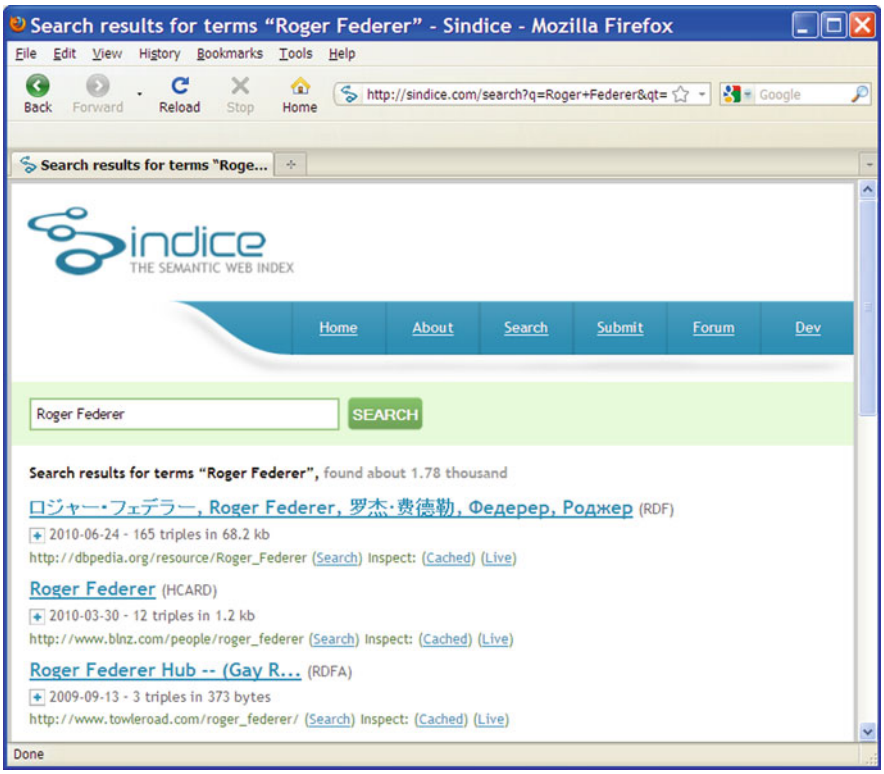

Fig. 11.4 Search results from Fig. 11.3

Open Data project wiki home page (see Sect. 11.1.2) and make sure to check back often for updates. The following is a short list, just to name a few:

- Friend-of-a-Friend (FOAF): terms for describing people;
- Dublin Core (DC): terms for general metadata attributes;
- Semantically Interlinked Online Communities (SIOC): terms for describing online communities;
- Description of a Project (DOAP): terms for describing projects;
- Music Ontology: terms for describing artists, albums, and tracks;
- Review Vocabulary: terms for representing reviews.

In case you do need to create your own ontology, it is still important to make use of the terms that are defined in these well-known ontologies. In fact, some of the ontologies given above, such as the Music Ontology, make use of the terms defined in other ontologies. For example, List 11.1 is taken from the Music Ontology, and it shows the definition of class `SoloMusicArtist`.

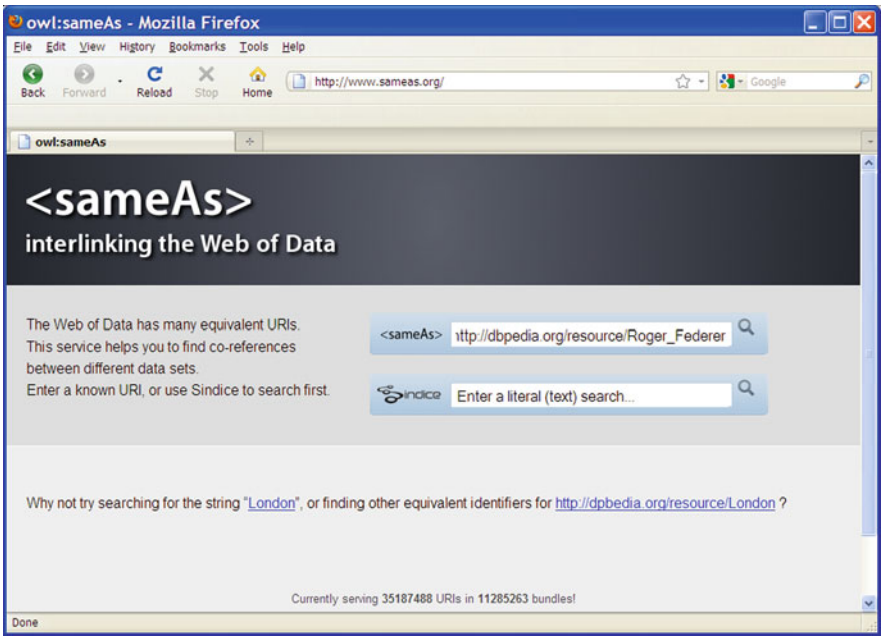

Fig. 11.5 Use sameAs to find URI aliases

List 11.1 Part of the Music Ontology

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY dc 'http://purl.org/dc/elements/1.1/'>
 <!ENTITY mo 'http://purl.org/ontology/mo/'>
 <!ENTITY ns1 'http://www.w3.org/2003/06/sw-vocab-status/ns#'>
 <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
]>

<rdf:RDF
    xmlns:dc="&dc;"
    xmlns:mo="&mo;"
    xmlns:ns1="&ns1;"
    xmlns:owl="&owl;"
    xmlns:rdf="&rdf;"
    xmlns:rdfs="&rdfs;"
    xmlns:xsd="&xsd;"
>

...
```

```
<rdfs:Class rdf:about="&mo;SoloMusicArtist"
    mo:level="1"
    rdfs:label="SoloMusicArtist"
    ns1:term_status="stable">
  <rdfs:subClassOf rdf:resource="&mo;MusicArtist"/>
  <rdfs:subClassOf
       rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdf:type rdf:resource="&owl;Class"/>
  <rdfs:comment>Single person whose musical creative work shows
sensitivity and imagination.
  </rdfs:comment>
  <rdfs:isDefinedBy rdf:resource="&mo;"/>
</rdfs:Class>
...
```

Note that `SoloMusicArtist` is defined as a sub-class of `foaf:Person`. Therefore, if a given client sees the following:

```
<rdf:Description
   rdf:about="http://zitgist.com/music/artist/79239441-bfd5-4981-
   a70c-55c3f15c1287">
   <rdf:type
    rdf:resource="http://purl.org/ontology/mo/SoloMusicArtist"/>
</rdf:Description>
```

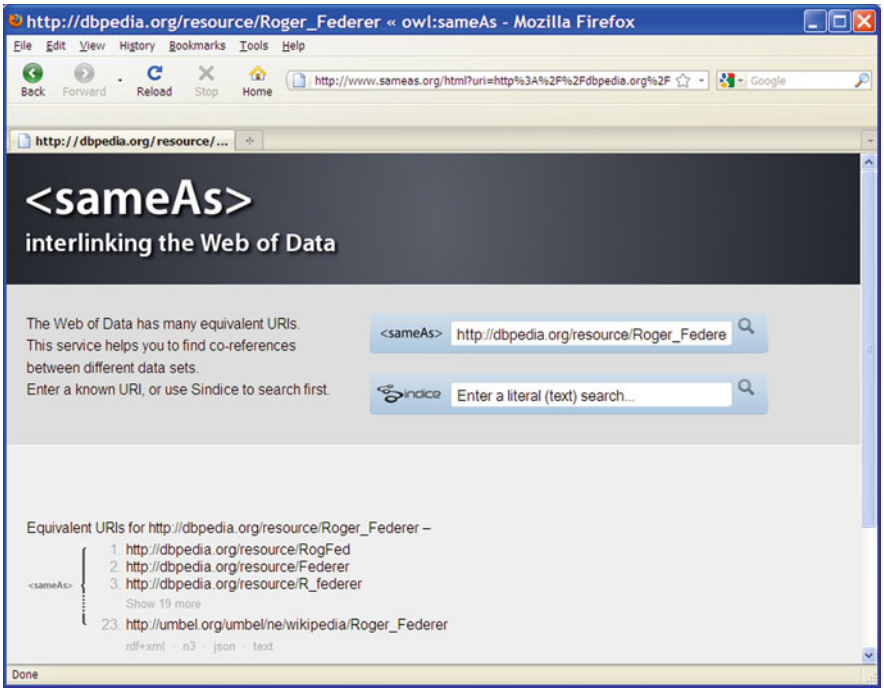

Fig. 11.6 sameAs search result

it will know the real-world resource identified by this URI,

```
http://zitgist.com/music/artist/79239441-bfd5-4981-a70c-
55c3f15c1287
```

must be an instance of `foaf:Person`. If this client is not interested in any instance of `foaf:Person`, it can safely disregard any RDF statements that are related to this resource. Clearly, this reasoning is possible only when the authors of the Music Ontology have decided to make use of the terms defined in the FOAF ontology.

Creating ontology, like any other design work, requires not only knowledge, but also experience. It is always helpful to learn how other ontologies are created, so check out the ontologies listed above. After reading and understanding how these ontologies are designed and coded, you will be surprised to see how much you have learned. Also, with the knowledge you have gained, it is more likely that you will be doing a solid job when creating your own.

11.2.3 Creating Links to Other RDF Data

Now that you have come up with the URIs, and you have the terms from the ontologies to use, you can go ahead to make your statements about the world. There is only one thing you need to remember: you need to make links to other RDF datasets so your statements can participate in the Linked Data cloud.

In this section, we discuss the basic language constructs and ways you can use to add these links.

11.2.3.1 Basic Language Constructs to Create Links

Let us start with a simpler case: you are creating a FOAF document. The easiest way to make links in this case is to use `foaf:knows`, as we have shown in Chap. 7. Using `foaf:knows` will not only make sure you can join the "circle of trust", but also put your data into the Linked Data cloud.

In fact, besides `foaf:knows`, there are couple other FOAF terms you can use to create links. Let us take a look at some examples:

- Use **`foaf:interest`** to show your interest:

  For example,

```
<rdf:RDF
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:foaf=http://xmlns.com/foaf/0.1/"
  <!-- other namespace definitions -->
>
<foaf:interest>
   <rdf:Description
        rdf:about="http://dbpedia.org/resource/Photography">
```

```
        <dc:title>photography</dc:title>
    </rdf:Description>
</foaf:interest>
```

This will link you to the world of photography as defined in DBpedia. And as you know, DBpedia is a major component of the Linked Data cloud.

- Use `foaf:base_near` to show where you are located:

  For example,

```
<rdf:RDF
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      <!-- other namespace definitions -->
>
  <rdf:Description
       rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
     <foaf:name>liyang yu</foaf:name>
     <foaf:based_near
           rdf:resource="http://dbpedia.org/resource/Beijing"/>
     <!-- other descriptions I may want -->
  </rdf:Description>
</rdf:RDF>
```

This will link you to Beijing, the capital city of China, which is represented by DBpedia as `http://dbpedia.org/resource/Beijing`. Again, this is good enough for putting you into the Linked Data cloud.

With the above two examples, you understand that there are different FOAF terms you can use to link to the Web of Linked Data. We will leave it to you to discover other FOAF terms that can be used besides the above two examples.

For a more general case, at least two properties should be considered when making links: `rdfs:seeAlso` and `owl:sameAs`.

`rdfs:seeAlso` is defined in W3C's RDFS vocabulary, and it is used to indicate the fact that another resource might provide additional information about the subject resource. It therefore can be used to link the current RDF document into the Linked Data world.

In addition, note that `rdfs:domain` of `rdfs:seeAlso` is `rdfs:Resource`, and `rdfs:range` of `rdfs:seeAlso` is also `rdfs:Resource`. As a result, this property is entirely domain-neutral, and works for people, companies, documents, etc.

List 11.2 shows one simple example of using `rdfs:seeAlso`.

List 11.2 Use `rdfs:seeAlso` to create link

```
<rdf:Description
     rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
  <foaf:name>liyang yu</foaf:name>
  <foaf:title>Dr</foaf:title>
```

```
  <foaf:givenname>liyang</foaf:givenname>
  <!-- other descriptions here -->
  <rdfs:seeAlso>
     <rdf:Description
       rdf:about="http://www.liyangyu.com/people/connie.rdf">
     </rdf:Description>
  </rdfs:seeAlso>
</rdf:Description>
</rdf:RDF>
```

`rdfs:seeAlso` property in List 11.2 says that you can find more information about resource `http://www.liyangyu.com/foaf.rdf#liyang` from anther RDF document (`connie.rdf`). A given client can follow this link to download `connie.rdf` and expect to be able to parse this file and collect more information about the current resource.

This simple example in fact raises a very interesting question: when we build our application, is it safe to assume that the value of `rdfs:seeAlso` property will always be a document that can be parsed as RDF/XML?

Unfortunately, the answer is no. As we have discussed, the formal definition of `rdfs:seeAlso` is couched in very neutral terms, allowing a wide variety of document types. You could certainly reference a JPEG or PDF or HTML document with `rdfs:seeAlso`, which are not RDF documents at all. Therefore, an application should always account for all these possibilities when following the `rdfs:seeAlso` link.

Sometimes, it is a good idea to explicitly indicate that `rdfs:seeAlso` property is indeed used to reference a document that is in RDF/XML format. List 11.3 shows how this can be implemented.

List 11.3 Use `rdfs:seeAlso` together with `dc:format` to provide more information

```
<rdf:Description
     rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
  <foaf:name>liyang yu</foaf:name>
  <foaf:title>Dr</foaf:title>
  <foaf:givenname>liyang</foaf:givenname>
  <!-- other descriptions here -->
  <rdfs:seeAlso>
    <rdf:Description
       rdf:about="http://www.liyangyu.com/people/connie.rdf">
       <dc:format>application/rdf+xml</dc:format>
    </rdf:Description>
  </rdfs:seeAlso>
</rdf:Description>
</rdf:RDF>
```

Another useful feature about rdfs:seeAlso is that it is often used as a typed link, which can be very helpful to clients. List 11.4 shows one example of a typed link specified using rdfs:seeAlso.

List 11.4 rdfs:seeAlso used with typed link

```
<rdf:Description
     rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
   <foaf:name>liyang yu</foaf:name>
   <foaf:title>Dr</foaf:title>
   <foaf:givenname>liyang</foaf:givenname>
   <!-- other descriptions here -->
   <rdfs:seeAlso>
      <rdf:Description
          rdf:about="http://www.liyangyu.com/publication/liyang">
        <rdf:type
    rdf:resource="http://example.org/someAuthorClassDefinition"/>
        <dc:format>application/rdf+xml</dc:format>
      </rdf:Description>
   </rdfs:seeAlso>
   <rdfs:seeAlso>
      <rdf:Description
       rdf:about="http://www.liyangyu.com/publication/yu_cv.rdf">
        <rdf:type
    rdf:resource="http://example.org/someResumeClassDefinition"/>
        <dc:format>application/rdf+xml</dc:format>
     </rdf:Description>
  </rdfs:seeAlso>
</rdf:Description>
</rdf:RDF>
```

Imagine an application that is only interested in publications (not CVs). This typed link will help the application to eliminate the second rdfs:seeAlso, but only concentrate on the first one.

owl:sameAs is not something new either. It is defined by OWL to state that two URI references refer to the same individual. It is now frequently used by Linked Data publishers to create links between datasets. For example, Tim Berners-Lee, in his own FOAF file, has been using the following URI to identify himself:

```
http://www.w3.org/People/Berners-Lee/card#i
```

and he also uses the following four owl:sameAs properties to state that the individual identified by the above URI is the same individual as identified by these URIs:

```
<owl:sameAs rdf:resource="http://identi.ca/user/45563"/>
<owl:sameAs
rdf:resource="http://www.advogato.org/person/timbl/foaf.rdf#me"/>
<owl:sameAs rdf:resource=
"http://www4.wiwiss.fu-berlin.de/bookmashup/persons/Tim+Berners-
Lee"/>
```

```
<owl:sameAs rdf:resource=
  "http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007"/>
```

and clearly, some of these URIs can be used to link his FOAF document into the Linked Data cloud. In fact, at this point, the last two URIs are indeed used for this purpose.

`owl:sameAs` can certainly be used in other RDF documents. Generally speaking, when instances of different classes refer to the same individual, these instances can be identified and linked together by using `owl:sameAs` property. This directly supports the idea that the same individual can be seen in different context as entirely different entities, and by linking these entities together, we can discover the unexpected facts that are both interesting and helpful to us.

The above discussion has listed some basic language constructs we can use to create links. In practice, when it comes to creating links in RDF documents, there are two methods: creating the links manually or generating the links automatically. Let us briefly discuss these two methods before we close this section.

11.2.3.2 Creating Links Manually

Manually creating links is quite intuitive, yet it does require you to be familiar with the published and well-known linked datasets out there, therefore you can pick your linking targets. In particular, the following steps are normally followed when creating links manually in your RDF document:

- Understand the available linked datasets.

  This can be done by studying the currently available datasets published and organized by experts in the field. For example, as of July 2009, Richard Cyganiak has published the *LOD Cloud* as shown in Fig. 11.7. The updated version can be accessed from this location:

  `http://linkeddata.org/images-and-posters`

  And if you access this Linked Data collection at the above location, you can actually click each dataset and start to explore that particular dataset. This will help you to get an overview of all the datasets that are available today, and you can also select the dataset(s) that you wish to link into.

- Find the URIs as your linking targets.

  Once you have selected the datasets to link into, you can then search in these datasets to find the URIs that you want to link to. Most datasets provide a search interface, such as a SPARQL endpoint, so you can locate the appropriate URI references for your purpose. If there is no search interface provided, you can always use Linked Data browsers to explore the dataset, as we will discuss in a later section.

With the above two steps, you can successfully create your links. Let us take a look at a simple example.

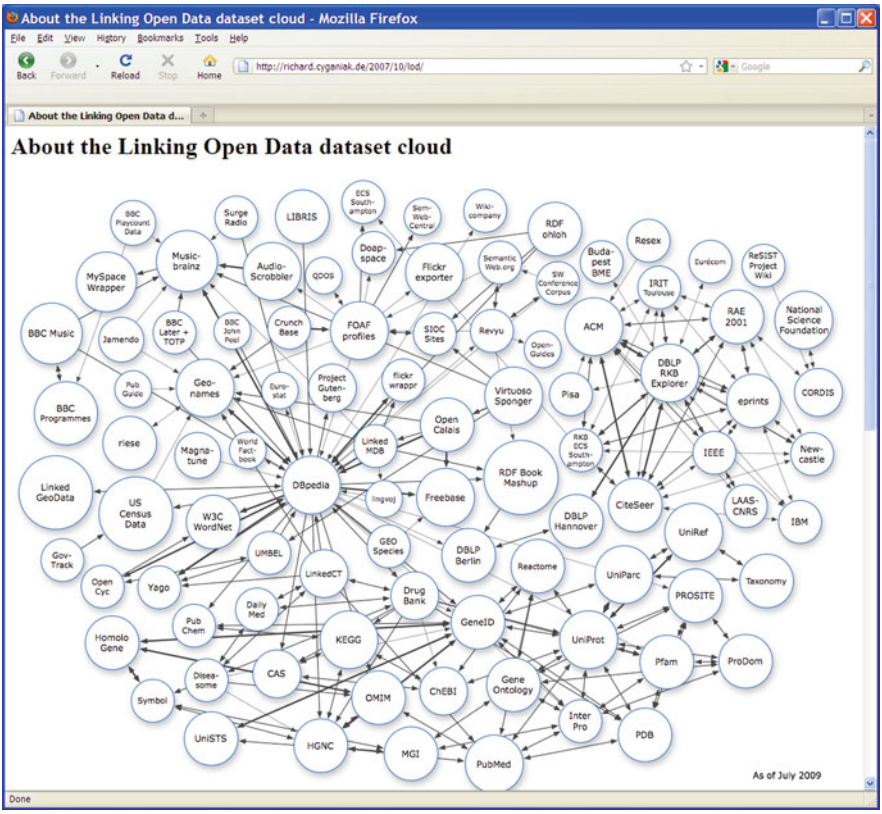

Fig. 11.7 Richard Cyganiak's clickable version of LOD cloud as of July 2009

Assume I have created my own FOAF document, and I am ready to create some links which point to some available linked dataset. The first step is to choose such a dataset. Let us say I would like to express the fact that I am interested in the Semantic Web, and I need to find a dataset which describes the concept of the Semantic Web. Since DBpedia is machine-readable version of Wikipedia, it is safe to assume that DBpedia might already have included the concept of the Semantic Web. Therefore, DBpedia is currently chosen as the target dataset.

The second step is to search in DBpedia datasets for the URI that represents the Semantic Web. In fact, I have found the following URI that describes this concept:

```
http://dbpedia.org/resource/Semantic_Web
```

and with this, I can add the following link into my own FOAF:

```
<http://www.liyangyu.com/foaf.rdf#liyang>
foaf:topic_interest <http://dbpedia.org/resource/Semantic_Web>.
```

This will successfully put my own small FOAF document into the Web of Linked Data.

In some cases, you can use a relatively direct way to find the URI reference that you can use to create your links. For example, without selecting any datasets, we can directly search the phrase "the Semantic Web" in `Sindice.com`. We can easily find a list of URIs that have been created to identify this concept, including the URI coined by DBpedia.

11.2.3.3 Creating Links Automatically

Compared to manually creating links, generating links automatically is certainly more efficient and more scalable, and it is always the preferred method if possible. However, at the time of this writing, there is still a lack of good and easy-to-use tools to automatically generate RDF links. In most cases, dataset-specific algorithms have to be designed to accomplish the task. In this section, we will briefly discuss this topic so as to give you some basic idea along this direction.

A collection of often used algorithms is the so-called *pattern-based algorithms*. This group of algorithms take advantage of the fact that for a specific domain, there may exist some generally accepted naming pattern, which could be useful for generating links.

For example, in the publication domain, if ISBN is included as part of the URI that is used to identify a book, such as the case in the RDF Book Mashup dataset, then a link can be created with ease. More specifically, DBpedia can locate all the wiki pages for books, and if a given wiki page has an ISBN number included, this number is used to search among the URIs used by RDF Book Mashup dataset. When a match is found, an `owl:sameAs` link will be created to link the URI of the book in DBpedia to the corresponding RDF Book Mashup URI. This algorithm has helped to generate at least 9000 links between DBpedia and RDF Book Mashup dataset.

In cases where no common identifiers can be found across datasets, more complex algorithms have to be designed based on the characteristics of the given datasets. For example, many geographic places appear in Geonames[4] dataset as well as in DBpedia dataset. To make the two sets of URIs representing these places link together, the Geonames team has designed a property-based algorithm to automatically generate links. More specifically, properties such as latitude, longitude, country, and population are taken into account, and a link will be created if all these properties show some similarity as defined by the team. This algorithm has generated about 70,500 links between the datasets.

As a summary, automatic generation of links is possible for some specific domain, or with a specifically designed algorithm. When it is used properly, it is much more scalable than the manual method.

[4] http://www.geonames.org/ontology/

11.2.4 Serving Information as Linked Data

11.2.4.1 Minimum Requirements for Being Linked Open Data

Before we can put our data onto the Web, we need to make sure it satisfies some minimal requirements in order to be qualified as "Linked Data on the Web":

1. If you have created any new URI representing non-information resource, this new URI has to be dereferenceable in the following sense:
   - your Web server must be able to recognize the MIME-type `application/rdf+xml`;
   - your Web server has to implement the 303 redirect as described in Sect. 11.2.1.3. In other words, your Web server should be able to return a HTTP response containing a HTTP redirect to a document that satisfies the client's need (either an `rdf+xml` document or a `html+text` document, for example);
   - if implementing 303 redirect on your Web server is not your plan, your new URIs have to be hash URIs as we have discussed in Sect. 11.2.1.5.
2. You should include links to other data sources, so a client can continue its navigation when it visits your data file. These links can be viewed as outbound links.
3. You should also make sure there are external RDF links pointing at URIs contained in your data file, so the open Linked Data cloud can find your data. These links can be viewed as the inbound links.

At this point, these requirements should look fairly straightforward. The following are some technical details that you should be aware of.

First off, you need to make sure your Web server is able to recognize the `rdf+xml` as a MIME type. Obviously, this is necessary since once you have published your data into the Linked Data cloud, different clients will start to ask for `rdf+xml` files from your server. In addition, this is a must if you are using hash URIs to identify real-world resources.

A popular tool we can use for this purpose is called cURL,[5] which provides a command-line HTTP client that communicates with a given server. It is therefore able to help us to check whether a URI supports some given requirements, such as understanding `rdf+xml` as a MIME type, supporting 303 redirects and content negotiation, just to name a few.

To get this free tool, go to this place:

```
http://curl.haxx.se/download.html
```

and on this page, you will find different packages for different platforms. For windows users, you can find the download here:

```
http://curl.haxx.se/download.html#Win32
```

[5] http://curl.haxx.se/

Once you have downloaded the package, you can extract it to a location of your choice, and you should be able to find `curl.exe` in that location. You can then start to test whether a given server is able to recognize the `rdf+xml` MIME type.

For testing purpose, we can request my own URI as follows:

```
curl -I http://www.liyangyu.com/foaf.rdf#liyang
```

Note that the `–I` parameter has to be used here (refer to cURL's documentation for details). Once we submit the above line, the server sends back the content type and other HTTP headers along with the response. For this example, the following is part of the result:

```
HTTP/1.1 200 OK
Last-Modified: Tue, 11 Aug 2009 02:49:10 GMT
Accept-Ranges: bytes
Content-Length: 1152
Content-Type: application/rdf+xml
Connection: close
```

The important line is the `Content-Type` header. We see the file is served as `application/rdf+xml`, just as it should be. If we were to see `text/plain` here or if the `Content-Type` header was missing, the server configuration would have to be changed.

When it comes to fixing the problem, it does depend on the server. Use Apache as an example, the fix is simple: just add the following line to `httpd.conf` file, or to a `.htaccess` file in the Web server's directory where the RDF files are located:

```
AddType application/rdf+xml.rdf
```

That is it. And since you are on it, you might as well go ahead and add the following two lines to make sure your Web server can recognize two more RDF syntaxes, i.e., N3 and Turtle:

```
AddType text/rdf+n3;charset=utf-8.n3
AddType application/x-turtle.ttl
```

Now, when it comes to configuring your Web server to implement 303 redirect and furthermore content negotiation, it is unfortunately not all that easy. This process depends heavily on your particular Web server and its local configuration; it is some times quite common that you may not even have the access rights that are needed to make the configuration changes. Therefore, we will not cover this in detail, but remember, it is one step that is needed to publish Linked Data on the Web and it is not hard at all if you have the full access to your server.

With all these said, let us take a look at one example showing how to publish Linked Data on the Web.

11.2.4.2 Example: Publishing Linked Data on the Web

A good starting point is to publish our own FOAF files as Linked Data on the Web. Let us start with my own FOAF file. To satisfy the minimal requirements that we have discussed above, we can follow these steps:

Step 1. Check whether our Web server is configured to return the correct MIME type when serving `rdf/xml` files.

Let us assume this step has been done correctly or you can always follow the previous discussion to make sure your Web server is configured properly.

Step 2. Since we are not going to configure our Web server to implement 303 redirect and content negotiation, we decide to use Hash URI to identify myself:

`http://www.liyangyu.com/foaf.rdf#liyang`

Again, when a client attempts to dereference this URI, the hash fragment (`#liyang`) will be taken off by the client before it sends the URI to the server. The resulting URI is therefore given by the following:

`http://www.liyangyu.com/foaf.rdf`

Now, all we need to do is to make sure that we put the RDF file, `foaf.rdf`, at the right location on the server, so a client submitting the above URI will be able to look into the response and find the RDF file successfully.

In this example, `foaf.rdf` file should be located in the root directory on our server, and it could look like something as shown in List 11.5.

List 11.5 My own FOAF document

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <rdf:RDF
3:       xml:lang="en"
4:       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6:       xmlns:foaf="http://xmlns.com/foaf/0.1/">
7:
8: <rdf:Description
8a:     rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
9:   <foaf:name>liyang yu</foaf:name>
10:  <foaf:title>Dr</foaf:title>
11:  <foaf:givenname>liyang</foaf:givenname>
12:  <foaf:family_name>yu</foaf:family_name>
13:  <foaf:mbox_sha1sum>1613a9c3ec8b18271a8fe1f79537a7b08803d896
13a: </foaf:mbox_sha1sum>
14:   <foaf:homepage rdf:resource="http://www.liyangyu.com"/>
15:   <foaf:workplaceHomepage
15a:        rdf:resource="http://www.delta.com"/>
16:   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
17: </rdf:Description>
18: </rdf:RDF>
```

Step 3. Make sure you have outbound links.

We can add some outbound links to the existing linked datasets. As shown in List 11.6, properties `<foaf:knows>` (lines 18–24) and `<foaf:topic_interest>` (line 26) are used to add two outbound links. This will ensure any client visiting my FOAF document can continue its journey into the Linked Data cloud.

List 11.6 My FOAF document with outbound links

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <rdf:RDF
3:        xml:lang="en"
4:        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6:        xmlns:foaf="http://xmlns.com/foaf/0.1/">
7:
8: <rdf:Description
8a:      rdf:about="http://www.liyangyu.com/foaf.rdf#liyang">
9:    <foaf:name>liyang yu</foaf:name>
10:   <foaf:title>Dr</foaf:title>
11:   <foaf:givenname>liyang</foaf:givenname>
12:   <foaf:family_name>yu</foaf:family_name>
13:   <foaf:mbox_sha1sum>1613a9c3ec8b18271a8fe1f79537a7b08803d896
13a: </foaf:mbox_sha1sum>
14:   <foaf:homepage rdf:resource="http://www.liyangyu.com"/>
15:   <foaf:workplaceHomepage
15a:        rdf:resource="http://www.delta.com"/>
16:   <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
17:
18:   <foaf:knows>
19:     <!-- the following is for testing purpose -->
20:     <foaf:Person>
21:        <foaf:mbox
21a:            rdf:resource="mailto:libby.miller@bristol.ac.uk"/>
22:        <foaf:homepage
22a:           rdf:resource="http://www.ilrt.bris.ac.uk/~ecemm/"/>
23:     </foaf:Person>
24:   </foaf:knows>
25:
26:   <foaf:topic_interest
26a:     rdf:resource="http://dbpedia.org/resource/Semantic_Web"/>
27:
28: </rdf:Description>
29: </rdf:RDF>
```

Step 4. Make sure you have inbound links.

This step is to make sure my FOAF document can be discovered by the outside world. The details about this can be found in Chap. 7, refer back to that chapter if you need to.

After these four steps, we are ready to upload my FOAF document onto the server at the right location and claim success. However, for those curious minds, how do

we know it is published as Linked Data correctly based on the given standards? Is there a way to check this?

The answer is yes, and let us now take a look at how to make sure we have done everything correctly.

11.2.4.3 Make Sure You Have Done It Right

Just as we have validators for checking RDF documents including RDF instance files and OWL ontologies, we also have Linked Data validator which can be used to confirm whether some structured data are published correctly as Linked Data, based on the current best practices as we have been discussing in this chapter.

Here is one such tool you can use. It is called Vapour and you can access this service from this location:

```
http://vapour.sourceforge.net/
```

and Fig. 11.8 shows this page.

Let us again use my own FOAF document as shown in List 11.6 as the example, and we will validate whether this FOAF document is published correctly as Linked

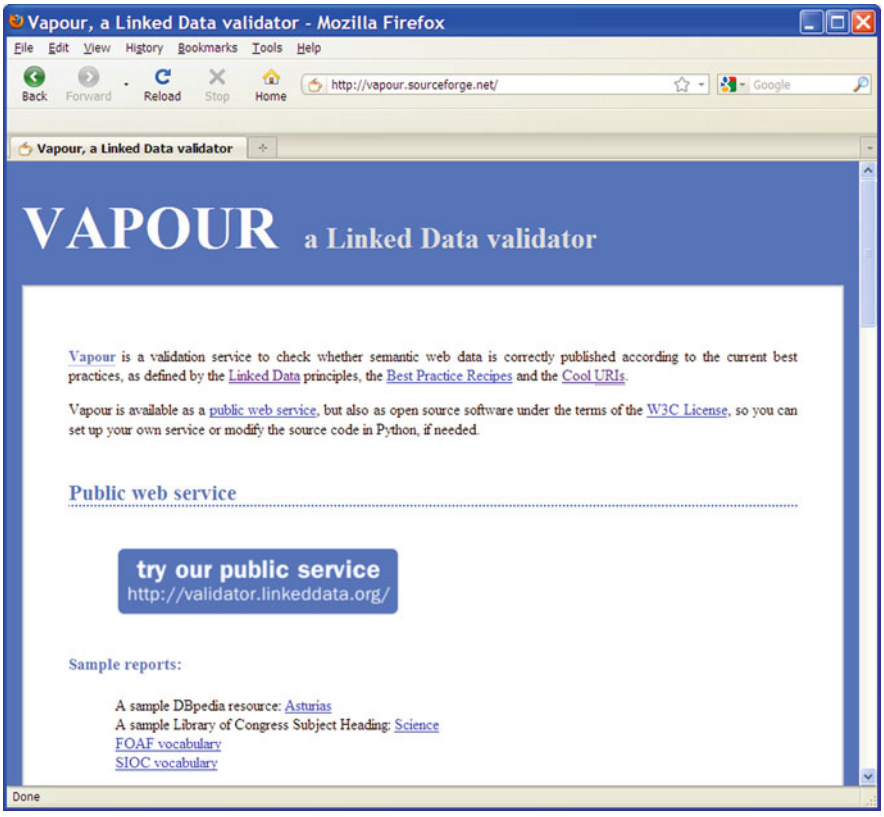

Fig. 11.8 Vapour: a Linked Data validator

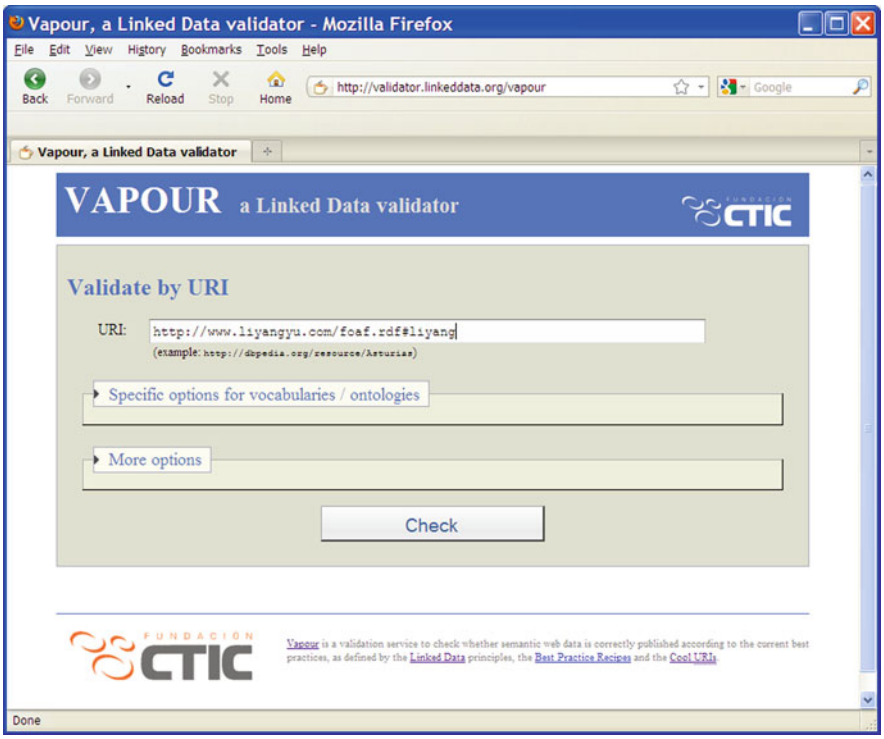

Fig. 11.9 Check my own URI to make sure it is published correctly as Linked Data

Data. To do so, click `try our public service` link in Fig. 11.8 and enter the following URI as shown in Fig. 11.9:

```
http://www.liyangyu.com/foaf.rdf#liyang
```

Once we click `Check` button, we get the result as shown in Fig. 11.10. Clearly, all tests are passed, meaning that my FOAF document is indeed published correctly as Linked Data.

You can also see more details on the same page if you try this test out, including dereferencing resource URI with and without content negotiation. As a summary, it is always a good idea to use a validation service when you publish Linked Data on the Web, to make sure your data will participate in the loop successfully.

11.3 The Consumption of Linked Data

Now that we understand how the Web of Linked Data is built, the next step is to study what to do with it. In general, this involves discovering Linked Data, accessing Linked Data, and building applications that run on top of the Web of Linked Data, as summarized below:

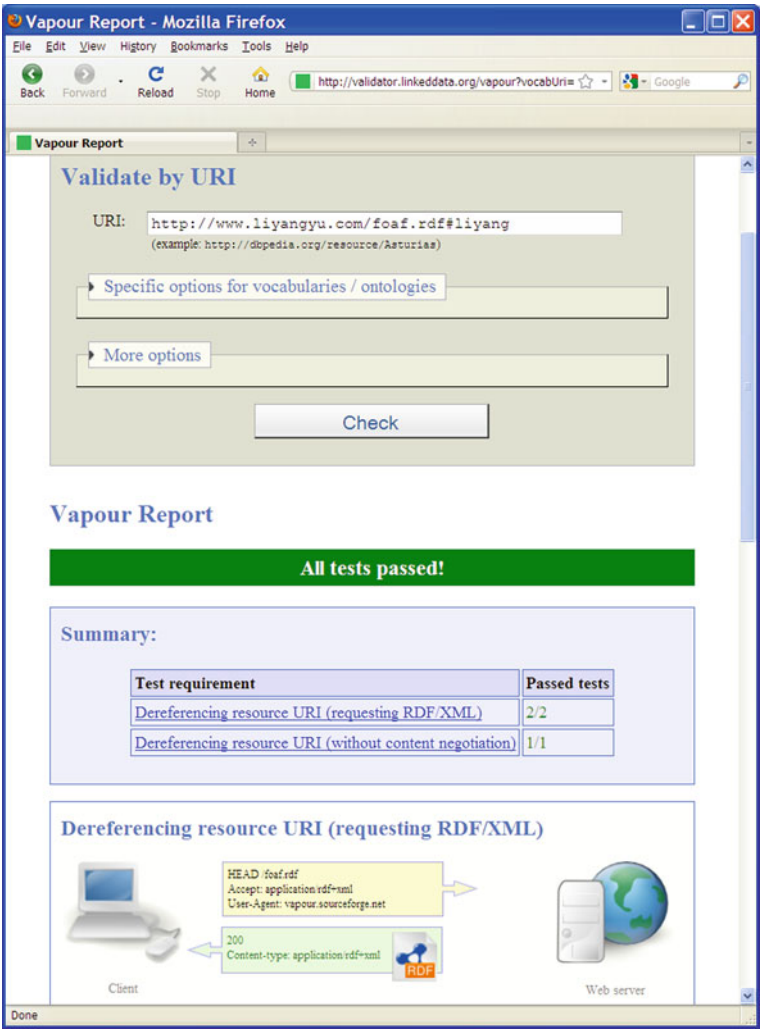

Fig. 11.10 Validating result from Fig. 11.9

- Discovery of Linked Data

  For a given resource in the world, for example, a city or a tennis player, how do we know this resource has already been a subject of Linked Data? Is there any Linked Data search engine that crawls the Web of Linked Data by following links between data sources, and therefore provides answers to our questions?

- Accessing Linked Data

  We use Web browsers to access our current Web, the Web of documents. For the Web of Linked Data, do we have similar Linked Data browsers that we can use to

access the Web of Linked Data? If we have indeed discovered some Linked Data that we are interested in, how can we start from there? And by using Linked Data browser, can we start browsing in one data source and then navigate along links into related data sources?

- Applications built upon Linked Data

  Given the fact that the Web of Linked Data is built for machine to read and understand, we can go beyond discovery and accessing the Web of Linked Data and create new applications built upon the Web of Linked Data. Compared to Web 2.0 mashups, Linked Data applications offer much more flexibility and completeness in their operations, as we will see later in this chapter.

11.3.1 Discover Specific Target on the Linked Data Web

In the world of traditional hypertext Web, discovery almost exclusively means using one of the major search engines to find the information you are interested in. Search engines are therefore the places where the navigation process begins.

For the Web of Linked Data, the same is true: we need search engines that can work on the Web of Linked Data and therefore can provide us with a tool to make our discovery.

It will not be too surprising if you are seeing a different look-and-feel from the search engines that work on the Web of Linked Data. After all, the Web of Linked Data is quite different from our traditional Web of documents. In fact, Semantic Web search engines are mostly geared toward the needs of applications, not that of human eyes. Nevertheless, some researchers and developers have designed search engines that have a similar look-and-feel as the traditional search engines, and this breed of search engines can be very useful to at least some user groups.

In this section, we will cover both these types, with the goal of discovering Linked Data on the Web. To make things easier, we will start from those Semantic Web search engines that look familiar to our human eyes.

11.3.1.1 Semantic Web Search Engine for Human Eyes

First off, remember that these kinds of search engines are Semantic Web search engines from their roots. Instead of crawling the Web of document and indexing each document, these search engines crawl the Web of Linked Data by following their RDF links, and prepare their indexations based on the Web of Linked Data.

Falcons is a good example of this type of search engine. Falcons represents "Finding, Aligning and Learning ontologies, ultimately for Capturing knowledge via ONtology-driven approacheS," and it is developed by the Institute of Web Science (IWS), Southeast University of China. You can access it from the following link:

`http://iws.seu.edu.cn/services/falcons/`

The first thing to note about Falcons is that it provides a keyword-based search service, i.e., the user is presented with a search box, where keywords related to the topics in mind can be entered. Falcons then reacts by returning a list of results that may be related to the topic. Clearly, this closely mimics the same look-and-feel offered by current market leaders such as Google and Yahoo!.

Let us say we want to discover if there is any Linked Data about tennis player Roger Federer. Obviously, if Roger Federer is indeed mentioned in the Web of Linked Data, he has to be some instance of a given class. For example, he could be an instance of some class such as `Person` defined in some ontology. With this in mind, we should use the `Object` search in Falcons, and enter Roger Federer in the search box. This will tell Falcons that the results we are searching for has to contain "Roger Federer" as keywords and should be coming from some instance data, not class or type definitions.

Once we submit this query, Falcons responses by returning a list of results. When presenting the results, for each object (instance data), Falcons shows its title, label, comment, image, page, type, and URI, if applicable. Clearly, for the Web of Linked Data, type and URI are all important since type identifies the class of this instance data, and URI uniquely identifies the instance. For our example, the first result is a good hit: the URI is given by

`http://dbpedia.org/resource/`**`Roger_Federer`**

and its type is `Person`. Clearly, the above URI comes from DBpedia, and we know already that DBpedia is a key component of Linked Data. Therefore, just based on the very first result, we know we have discovered some Linked Data for tennis player Roger Federer, which can be a good start point for whatever we plan to do next.

Note that Falcons also provides a `Type` pane together with the search result, and this is in fact a very useful feature. Recall that when we first started our search for any Linked Data related to Roger Federer, we can only say that if this data exists, it has to be some instance data of some class type. However, we don't really know what exactly this class type is, except that the correct type should be something like `Person` or `Athlete`.

Now, the `Type` pane on the result page shows all the types that are found in the results. Note that the initial search will focus on "Any type", therefore it does not put any further constraints on the type at all. Once we have the initial results back, we can further narrow down the type by clicking a specific type in the `Type` pane. For example, we can click `Person` in the `Type` pane, telling Falcons that we believe Roger Federer should be an instance of some `Person` class. Once we do this, all the sub-classes of type `Person` are now summarized in `Type` pane, and you can continue to narrow down your search. Therefore, `Object` search can be guided by recommended concepts, and we can further refine search results by selecting object types.

Besides `Object` search tab as we have discussed above, Falcons provides two more search tabs: `Concept` and `Document`. `Concept` search is not much related to discovering Linked Data on the Web, it is more suited for locating classes and

properties defined in ontologies that are published on the Web. It is quite useful if you want to find classes and properties that you can reuse, instead of inventing them again.

`Document` search gives you a more traditional search engine experience, especially the look-and-feel of the search results. If you search for Roger Federer, any RDF document that contains these words will be returned as part of the result list, be these search items in the instance data or the class or property definitions. Although not quite efficient, this search can also be used to discover Linked Data on the Web.

11.3.1.2 Semantic Web Search Engine for Applications

We have made use of Sindice search engine in previous sections, with the goal of discovering if there is any URI existing for some real-world object that we would like to talk about. For example, if we want to say something about Roger Federer, we can use Sindice to discover the URI for him, as shown in Figs. 11.3 and 11.4.

Sindice search engine therefore can also be used as a tool to discover Linked Data on the Web. When used by human users, it has a similar look-and-feel as Falcons does: a certain number of keywords can be provided to Sindice, and Sindice will return RDF documents on the Web which contains these keywords.

Furthermore, if you know the URI for some real-world object, you can search it in Sindice, and Sindice will return all the RDF documents on the Web which contains this given URI. For example, Fig. 11.11 shows the result from Sindice when we search for the URI of Roger Federer.

Clearly, these RDF documents are all linked to some extent since they all have the URI of Roger Federer in their triples.

By far, Sindice feels much like Falcons. As human users, we can use both to discover Linked Data on the Web. However, there is more to Sindice: it can be used by applications as well.

We have not yet discussed Linked Data applications at this point. However, it is quite intuitive to realize that the first thing each Linked Data application will have to do is to somehow harvest some Linked Data before it can do anything interesting with the data. As a result, each Linked Data application will have to implement its own crawling and indexing component, just to find the interested Linked Data. Clearly, moving this common infrastructure for crawling and indexing the Web of Linked Data to a search engine that each individual Linked Data application can then use will be a much better and cleaner design.

This is the rationale behind the design and implementation of Sindice's Data Web Services API and also the reason why we claim Sindice can be a search engine used by applications. More specifically, each application, by using the Sindice API, can query Sindice's collection and receive a set of links that point to those potentially relevant RDF documents, which can then be processed by the application to create other interesting results.

At the time of this writing, Sindice API is still in early beta version and is experiencing rapid changes and developments. Therefore, we are not going to show any

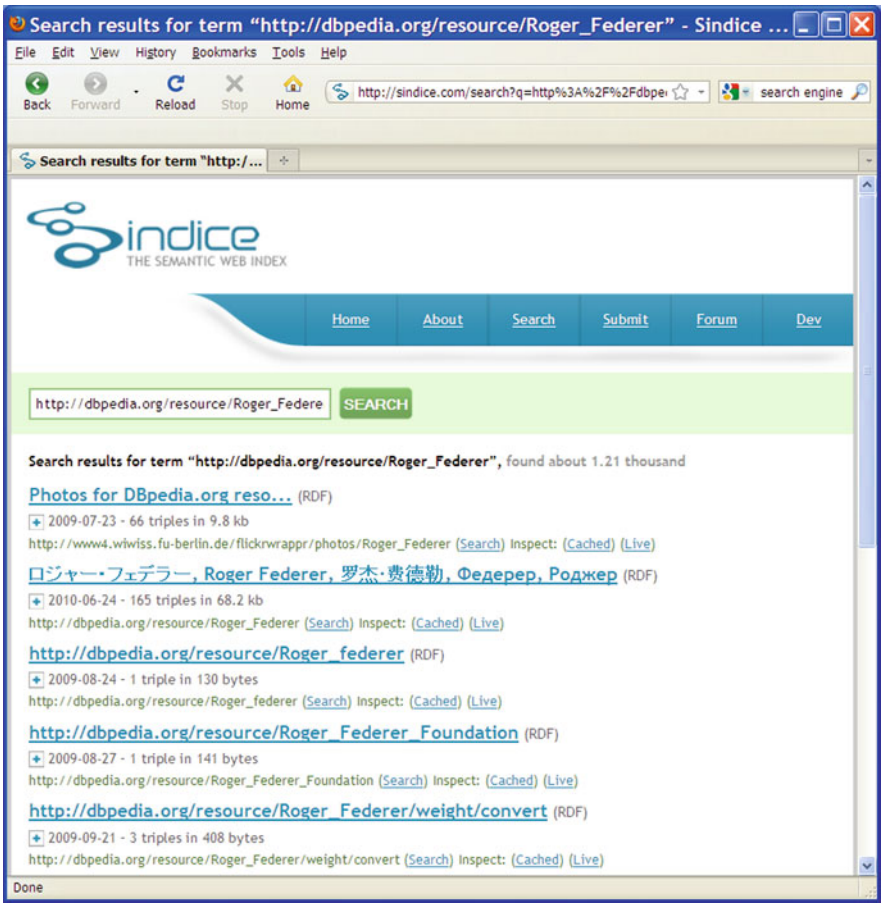

Fig. 11.11 Sindice results when searching for the URI of Roger Federer

concrete examples here, but the basic idea as we have discussed above will not change.

Before we conclude this section, let us very briefly discuss two more Semantic Web search engines, just to give you a flavor of other choices when it comes to discovering Linked Data on the Web:

- SWSE

  SWSE (Semantic Web Search Engine) is developed by DERI Ireland and can provide search capabilities more suitable toward human users. It accepts keyword-based search and further offers access to its underlying data store via SPARQL query language. At this point, you can access SWSE from this URL:

  `http://swse.deri.org/`

Also, note that similar to Sindice, SWSE is more related to search for instance data, not types and properties.

- Swoogle

  Swoogle is developed by UMBC Ebiquity Research group, which consists of faculty and students from the Department of Computer Science and Electrical Engineering (CSEE) of University of Maryland, Baltimore County (UMBC). Unlike Sindice or SWSE, Swoogle is designed to search ontologies that related to the concepts provided by its users. Swoogle also provides Web services to the public users, which can be used by applications that are built on top of the Linked Data Web. At this point, Swoogle can be accessed at this location:

  ```
  http://swoogle.umbc.edu/
  ```

11.3.2 Accessing the Web of Linked Data

Accessing the Web of Linked Data has two different meanings. First, human users can access it in a way that is similar to what has been done in traditional Web of documents, i.e., Linked Data browsers can be used to manually navigate from one data source to another. Second, applications that are built to understand Linked Data can access the Linked Data Web and further accomplish different requirements from us. For example, the so-called Follow-Your-Nose method can be used by applications to browse the Web of Linked Data. In this section, we will take a closer look at both these methods.

11.3.2.1 Using a Linked Data Browser

As human users, we can access the Web of Linked Data manually. This normally requires us to discover a specific piece of Linked Data on the Web first, which is then used as the starting point for further navigation on the Web of Linked Data.

As traditional Web browsers allow us to access the Web by following hypertext links, their counterparts in the Web of Linked Data, the so-called Linked Data browsers, allow us to navigate from the starting data source to the next data source. This process can go on by following links that are expressed and coded as RDF triples, and that is all there is to it when it comes to manually accessing the Web of Linked Data.

Therefore, the key component in this process is the Linked Data browser. To learn how to manually access the Linked Data Web is to learn how to use one of these browsers.

In recent years, there are quite a few browsers that have been developed and deployed for public use. To find a list of these browsers, you can visit the W3C's ESW Wiki page,[6] which is also updated quite frequently.

[6] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/SemWebClients

In this section, we will take the Sig.ma browser as an example to show you how Linked Data browsers can be used to access the Web of Linked Data.

Sig.ma is built on top of Sindice, which provides the data needs for Sig.ma. To some extend, Sig.ma acts much like Sindice's front-end GUI. You can access Sig.ma at this location:

```
http://sig.ma/
```

And its main page is shown in Fig. 11.12.

To start using it, simply enter the keywords you want to search in the input box and hit the SEARCH button, quite like using Sindice as a search engine. Obviously, this step is to find some entry point to access the Web of Linked Data.

Once you hit the SEARCH button, Sig.ma starts its work by doing the following steps:

1. select 20 data sources from the Web of Linked Data based on the keywords you have entered;
2. aggregate the information contained in these data sources; and
3. present the aggregated information and the data sources back to the user.

Note the first step is accomplished by using the underlying Sindice search engine to carry out a keyword-based search in the Web of Linked Data to select the relevant

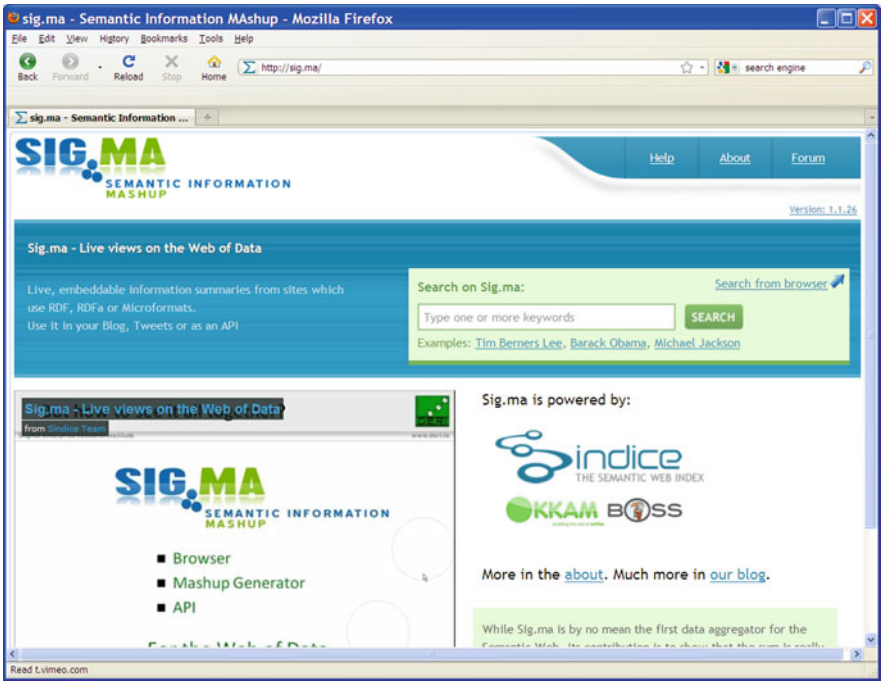

Fig. 11.12 Sig.ma: a Linked Data Web browser

data sources. If there are less than 20 data sources that are considered to be relevant to the given keywords, then whatever that are available will be included in the result set. If there are more than 20 data sources that are relevant, you can add the other data sources later on, as will be discussed soon.

Aggregating over the selected data sources essentially means to collect everything each data source says about the resource or concept represented by the keywords you have provided. And since the data sources are all taken from the Web of Linked Data, aggregating different data sources can be done easily.

Once the first two steps are done, Sig.ma presents the results back to the user by dividing the screen into the left pane and the right pane. The left pane shows the aggregated information, which is called the "sigma" for this search. The right pane shows the data sources based on which the sigma is obtained.

Let us use one example to see how it works. This time, instead of searching for Roger Federer, I will search for myself. The reason being that if we were to search Roger Federer, there would be too many datasets to be included. To show you how to use Sig.ma, a search that does not yield too many results is better.

Now, enter the keywords "liyang yu" in the search box (remember to include them in a pair of double quotes). Once you hit the SEARCH button, you will be presented with the result page as shown in Fig. 11.13 (notice if you are trying it, it is likely that what you see is not the same as what we printed here, and the reason is obvious).

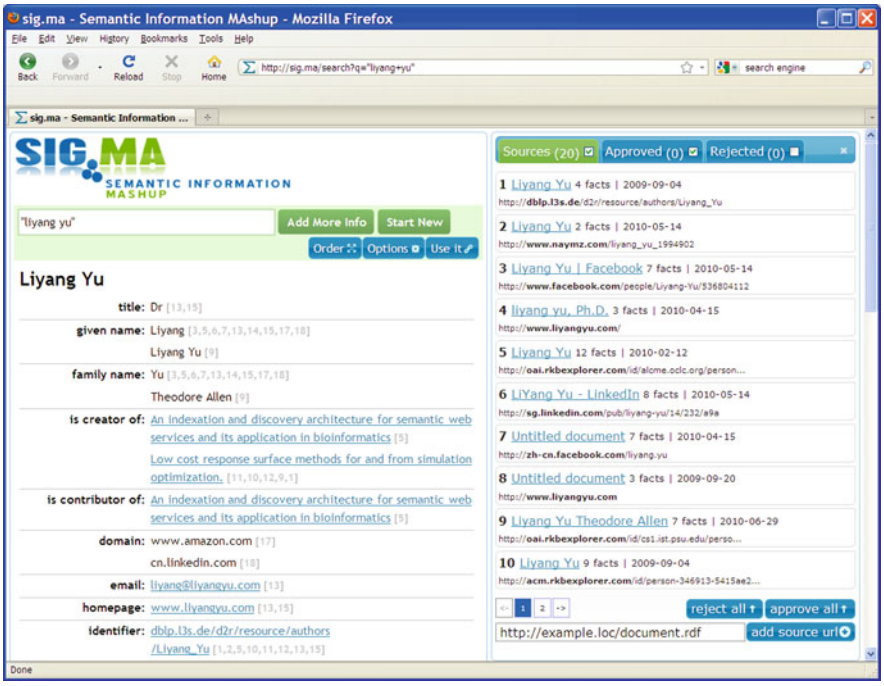

Fig. 11.13 Using Sig.ma to search for "liyang yu"

Let us first take a look at the left pane, i.e., the sigma of this search. It is quite different from what you would have seen if you had used a Google search. More specifically, Google search simply gives you back a list of links that point to a collection of Web pages, with each one of them containing the keyword "liyang yu." Google itself is not able to tell you the fact such as, on a given Web page, the word Liyang shows up as a given name, and on some other page, the word Yu shows up as a family name, so on and so forth.

For Sig.ma, however, this is not difficult at all. It knows not only that the word Liyang is a given name and the word Yu is a family name, but quite a lot more. For example, it can tell the string `liyang910@yahoo.com` represents my e-mail address. Clearly, since Sig.ma is built upon a Web of Linked Data, it is therefore able to present the result in a way that seems like the machine is able to understand all the data sources it has encountered during its search.

Now since the sigma pane shows the result of data aggregation, it is certainly useful to include the data sources that have been used to obtain the current sigma. This is the right pane, as shown in the Fig. 11.13. For discussion purpose, let us call it the source pane.

It is very easy for Sig.ma to trace the data source for each information segment in the sigma. For example, if you hover your mouse over the given name "Liyang", at least four data sources in the source pane will be highlighted, telling us that this information is included in all these four data sources, as shown in Fig. 11.14.

In fact, each data source in the source pane has a number associated with it, indicating how many facts are collected from this particular data source. For example, data source number 4

`http://www.liyangyu.com/`

has contributed three facts to the current sigma, as shown in Fig. 11.13. To see a detailed list of these three facts, hover your mouse over this document; the facts from this document will be highlighted in the sigma pane, as shown in Fig. 11.15.

Note that there is a pop-up menu showing up when you hover the mouse over data source number 4 (see Fig. 11.15). The first selection in this pop-up menu is called `solo`, which is a very useful tool: if you click `solo`, the current sigma will show only the facts that are collected from this data source, as seen in Fig. 11.16.

To go back to the complete list of facts, simply click `unsolo`, as you can easily tell.

Another useful feature of Sig.ma is the ability to approve and reject data sources. Recall the fact that search in Sig.ma is based on keyword matching, i.e., when it scans a given RDF data source, it looks for the keywords in that document. The keywords themselves can appear in a comment, a label, or the string value of a given subject. They can also show up in a URI that identifies a subject, a predicate, or an object. As the developers of Sig.ma have pointed out, since very simple strategies have been on purpose chosen at this stage to filter data source candidate, it is quite possible that a given data source is in fact not the data source you are looking for.

In the case where a given data source should not be included in the sigma, you can simply click the `reject` button from the pop-up menu in the source pane (you need to hover the mouse over the selected source data document). Once this is done, all

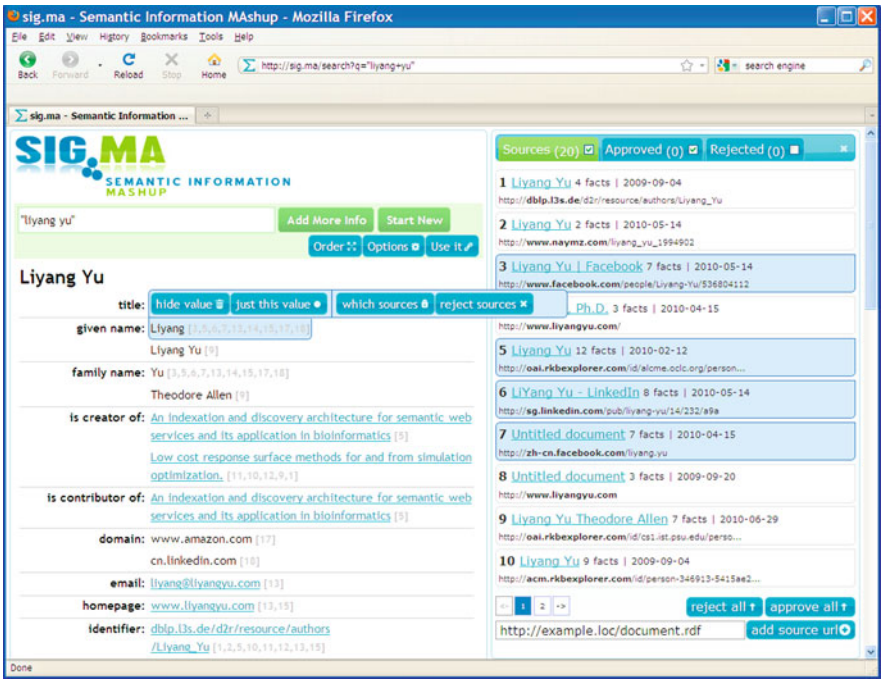

Fig. 11.14 Hovering the mouse over Liyang will show the data sources from where this information is obtained

the facts in the current sigma will be removed and the data source will be removed as well. For example, go back to Fig. 11.13; we can reject document number 4 by easily following these steps.

It is now easy to understand the `approve` selection. Clicking `approve` for a given data resource means that this data source is highly relevant to the search and should stay in the source pane at all times.

Besides rejecting and/or approving the data source files contained in the current source pane, I can click `Add More Info` button in the sigma pane to ask Sig.ma to search more datasets. Once more data sources have been added, I can start to filter them again by applying the same rejecting/approving process until all the data sources are stable in the source pane.

Figure 11.17 shows my final sigma. As you can tell, I have rejected altogether 12 data sources to reach this sigma.

In fact, my final sigma presents a set of entry points to the Web of Linked Data, since each one of the corresponding data sources in the source pane contains links to the Web of Linked Data. It is now time to start our navigation by following these links.

Let us get back to Fig. 11.17. Note the label `topic interest:` and its value `Semantic Web`. Clicking this link brings us to a brand new sigma as shown in Fig. 11.18.

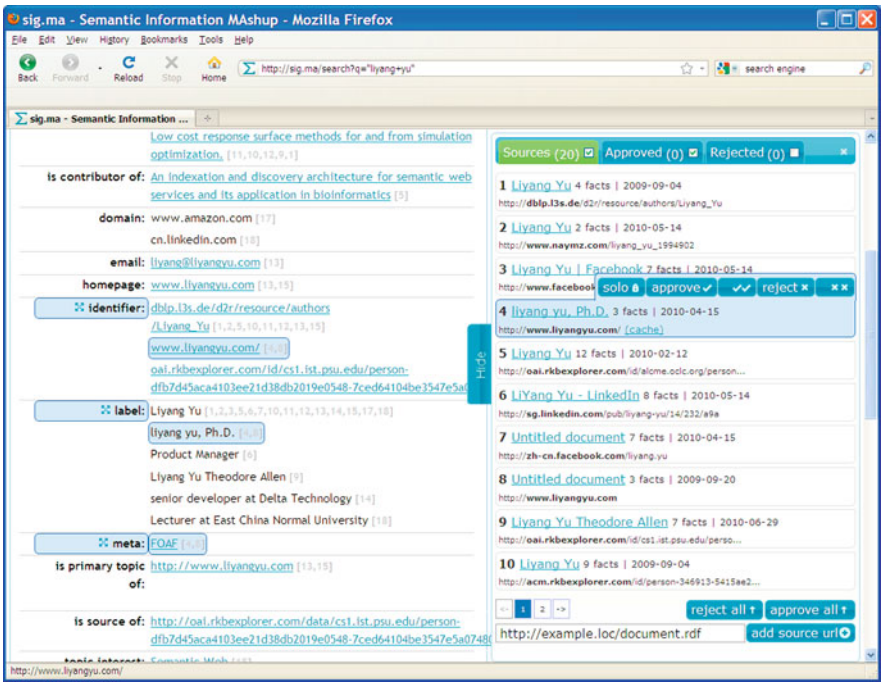

Fig. 11.15 Hover mouse over a data source document, all the facts from this document will be highlighted

As you can tell, the sigma about Semantic Web opens another new entry to the Linked Data Web for you to explore. For example, you can do the following:

- you can start to explore Jena Semantic Web Framework;
- you can start to read more about Resource Description Framework (RDF);
- you can start to understand OWL;
- and more.

I will leave this to you to continue, and Sig.ma is an excellent tool to search and access the Web of Linked Data.

Finally, note that we have only covered some basic functionality provided by Sig.ma. Sig.ma was first released on 22 July 2009, and given the fact that it is experiencing constant development and improvement, at the time when you are reading this book, you will likely see a different version of Sig.ma. However, the basics should remain the same.

11.3.2.2 Using SPARQL Endpoints

We have discussed how to use Linked Data browsers to access the Web of Linked Data manually in the previous section. While it is a useful way to explore the Linked

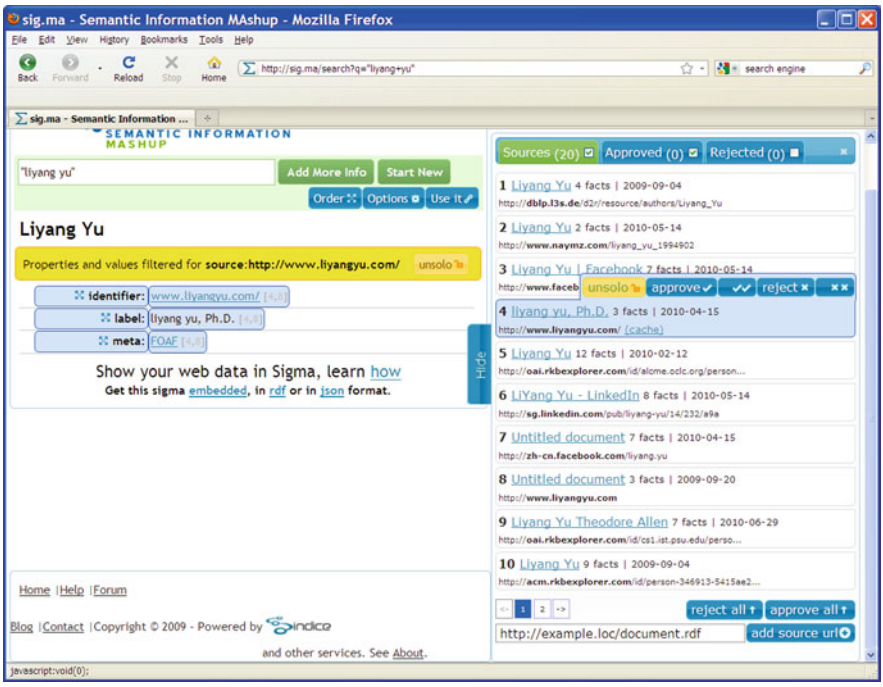

Fig. 11.16 Click `solo` will show only the facts collected from the selected data source in the sigma pane

Data, it does not take full advantage of the fact that the Web of Linked Data contains structured data, which means that we can actually access the Linked Data by using a query language.

In this section, we will use SPARQL to access the Web of Linked Data. In fact, this is not something completely new to us. In Chap. 10, we have discussed how to use SPARQL to access DBpedia. We will expand the same idea and show you how to use SPARQL to access the Web of Linked Data in general.

A good start point is the current Linked Data cloud presented in Fig. 11.7. Again, the benefit of accessing it online is that you can get the version that is clickable: when you click a dataset, it directly takes you to the home site of that dataset.

Now, let us say that we want to understand more about Musicbrainz, which at this point we know nothing about. Clicking this dataset takes us to the home site of Musicbrainz. On its home site, we can find the following SPARQL endpoint (note that not all the datasets provide SPARQL endpoints):

```
http://dbtune.org/musicbrainz/snorql/
```

and opening this endpoint will give us the query interface supported by the dataset.

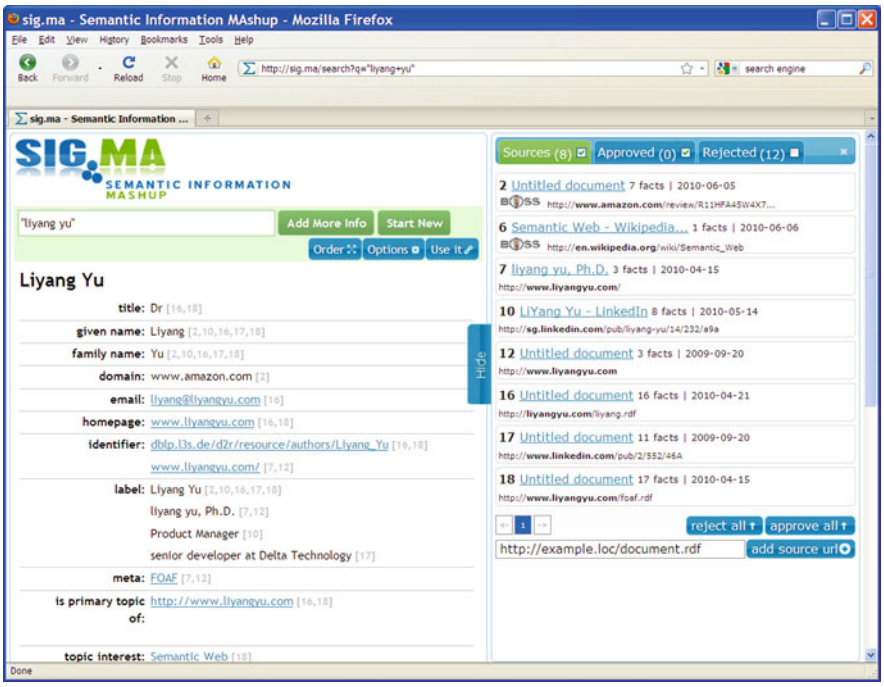

Fig. 11.17 My final sigma after rejecting 12 data sources

Now, to explore this dataset, or rather, to explore any given dataset, we can always start from two general queries. The first query is given below:

```
SELECT DISTINCT ?concept
WHERE
{[] a ?concept}
```

This shows all classes that are used in a given dataset. Note that there might be a large number of classes used, and some of them might look unfamiliar to you. However, this query can give you some feeling about what the given dataset is all about.

Let us try this query on the Musicbrainz dataset. Enter the above query in the query box, but change the query so it looks like this:

```
SELECT DISTINCT ?concept
WHERE
{[] a ?concept}
LIMIT 10
```

Adding `LIMIT 10` is to make sure the query can be executed in a reasonable amount of time. You can change it to another integer number if you prefer, such as `20`.

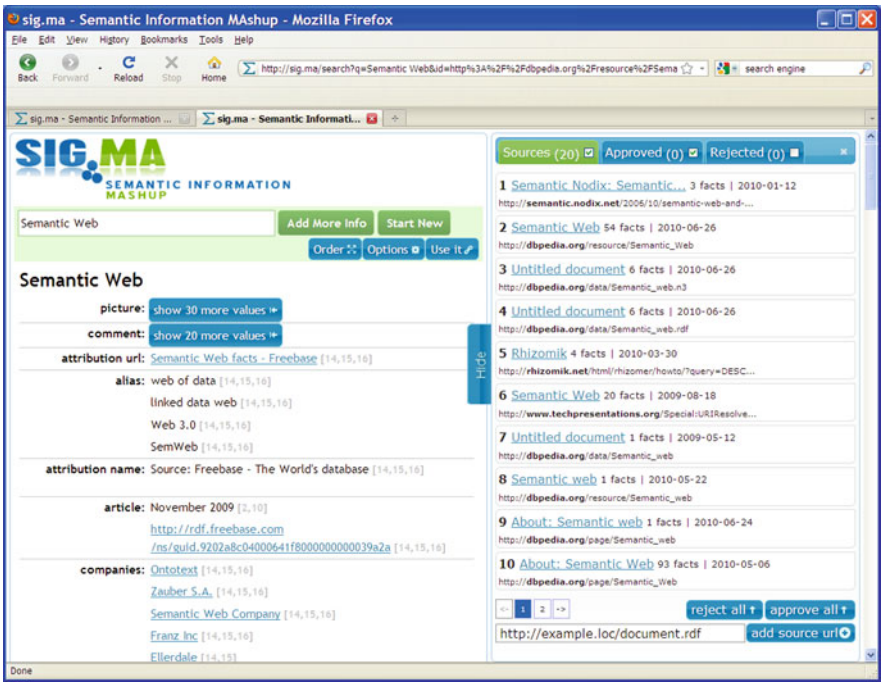

Fig. 11.18 Clicking `Semantic Web` from Fig. 11.17 will take us to this new sigma

Once you submit the query, you should get some results back. For example, part of the classes I got is shown as follows:

```
bio:Birth
bio:Death
db:vocab/puidjoin
db:vocab/l_label_track
db:vocab/lt_artist_label
db:vocab/lt_artist_artist
lingvoj:LinguisticSystem
mo:MusicArtist
mo:Performance
mo:Release
mo:Record
```

Again, by the time you are reading this book, you could get different results back.

Now, the above class list will give us some basic idea about what is covered in this dataset. For example, this dataset is more about some music artists, their albums, their performance, so on and so forth.

The second useful query is similar to the first one. It asks all the properties that are included in a given dataset:

```
SELECT DISTINCT ?property
WHERE
{?sub ?property ?obj}
```

Again, you might want to use it together with `LIMIT 10` constraint, just to make sure the performance of the endpoint is acceptable:

```
SELECT DISTINCT ?property
WHERE
{?sub ?property ?obj}
LIMIT 10
```

The following is the result:

```
rdfs:label
db:vocab/puidjoin_puid
db:vocab/puidjoin_usecount
db:vocab/puidjoin_id
db:vocab/puidjoin_track
rdf:type
db:vocab/l_label_track_enddate
db:vocab/l_label_track_link_type
db:vocab/l_label_track_begindate
db:vocab/l_label_track_modpending
```

As you can tell, the above two queries are very useful when you know nothing about the dataset. In fact, some SPARQL endpoints have included these two queries for you as your default starting point.

After these two general queries, it is up to you to continue your exploration. In most cases, what you will be doing depends on the results from these two queries. For example, for the Musicbrainz dataset, I am interested in `mo:MusicArtist` class, and I want to find who is a member of this class. To do so, I will use the following query:

```
SELECT ?artist
WHERE
{?artist a <http://purl.org/ontology/mo/MusicArtist> }
LIMIT 10
```

And I got 10 instances of `mo:MusicArtist` back. One of them is the following:

```
db:artist/0002260a-b298-48cc-9895-52c9425796b7
```

To know more about this instance, I continue to execute the following query:

```
SELECT ?property ?hasValue ?isValueOf
WHERE {
  { <http://dbtune.org/musicbrainz/resource/artist/
 0002260a-b298-48cc-9895-52c9425796b7> ?property ?hasValue }
  UNION
  { ?isValueOf ?property
    <http://dbtune.org/musicbrainz/resource/artist/
  0002260a-b298-48cc-9895-52c9425796b7> }
}
```

This query will find everything that has been said about this artist. Once you execute the query, you will get the name of the artist, the label, etc. Obviously, we can continue like this by following a number of different directions, and at some point, we will find ourselves moving on to explore other datasets.

The point is clear: besides using Semantic Web browsers or Semantic Web search engines to access the Web of Linked Data, it is also very useful and efficient to access it by using SPARQL queries. After all, search engines will point you to a set of documents that might contain the answer, but SPARQL queries can directly give you the answer you need.

11.3.2.3 Accessing the Linked Data Web Programmatically

The most significant difference between our current Web and the Web of Linked Data is the fact that the Web of Linked Data is processable by machine. Given this, it is certainly possible to access the Web of Linked Data programmatically, and it has already been the backbone of many Linked Data applications (as we will see in the next section).

Different applications may implement different ways of accessing the Web of Linked Data. However, two basic methods of accessing the Linked Data Web should be understood: one is referred to as Follow-Your-Nose method, the other is about issuing SPARQL queries within your application by using supporting tools.

The best way to learn these two methods is by going through some examples. Since these two methods are quite generic, they are discussed in Chap. 14; you can find working example for each method there. For now, we will move on with the discussion of Linked Data applications.

11.4 Linked Data Application

Discovering and accessing Linked Data is only the first step, our ultimate goal is to build applications that make use of Linked Data. In this section, we present one popular example to show you how Linked Data can be used.

11.4.1 Linked Data Application Example: Revyu

11.4.1.1 Revyu: An Overview

Revyu is a Web site that everyone can login to review and rate anything in the world. However, it is not just another review site; it is developed by using the Semantic Web technologies and standards, and by following Linked Data principles and best practices. More importantly, it also consumes Linked Data from the Web to enhance its user experience.

Revyu is implemented in PHP and runs on a regular Apache Web server. It can be accessed at this location:

```
http://revyu.com/
```

A registered user can review and rate things by filling out a Web form, which does not require any knowledge of the Semantic Web. Once finished, the user can submit this review form and the review will show up at the site.

This does not sound too much different from other review sites at all. However, lots of things will then happen inside Revyu. To understand all these, we first need to understand one fact: every review created in Revyu is also expressed as an RDF graph, besides its normal look-and-feel on the Web.

Let us look at one example. From Revyu home page, click `Search Things` link to search for the movie *Broken Flowers*, for which a review has been created as an example by Tom Heath, the creator of Revyu. Figure 11.19 shows the review page of the movie Broken Flowers.

On this page, click the link which identifies the reviewer. In this case, the link reads as `by tom on 30 Jan 2007`. Once you click this link, you will land on the page as shown in Fig. 11.20.

On the right side of the page, you will find a link called `RDF Metadata for this Review of Broken Flowers` (the right-hand side of Fig. 11.20). Clicking this link will take us to the RDF format of this review.

With the understanding that every review in Revyu has its RDF representation, let us now take a look at what will happen inside Revyu when a review is submitted by a user.

- All things represented in Revyu are assigned with URIs.

  At the moment a review is submitted, the reviewer (i.e., the user), the review the user creates, and the resource being reviewed are all assigned with URIs. Also, the tags used when reviewing the resource are assigned with URIs as well (will discuss tags later this section).

  Note that Revyu is designed to follow the four basic principles of Linked Data discussed early in this chapter. To see this, we can open the RDF document which represents the review of *Broken Flowers* and locate the URI Revyu has assigned to Tom Heath:

  ```
  http://revyu.com/people/tom
  ```

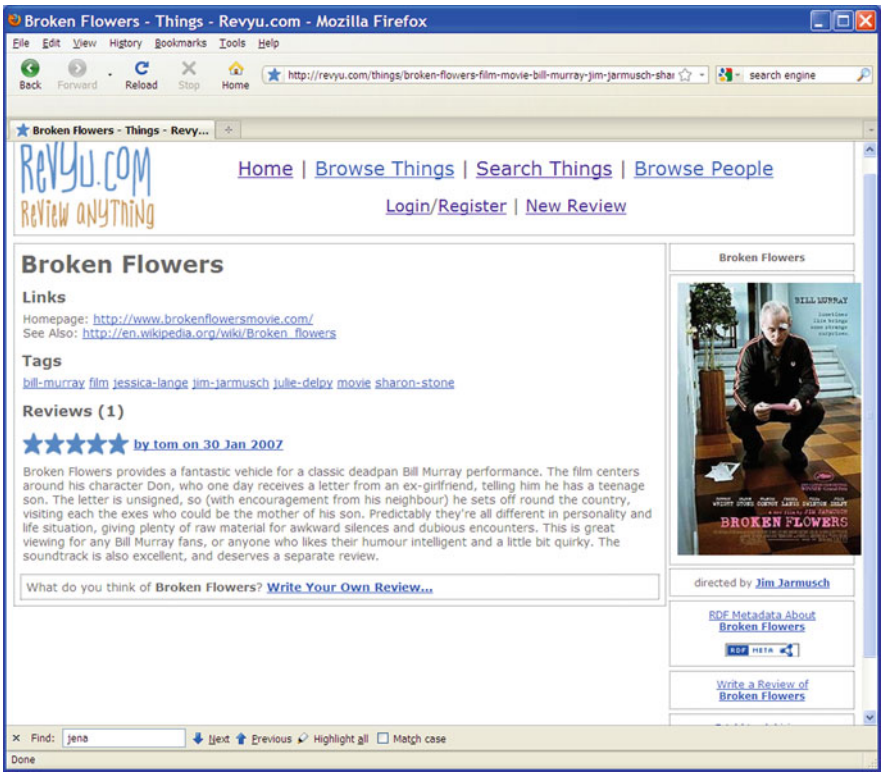

Fig. 11.19 Review page for the movie Broken Flowers

Since this URI represents a non-information resource, if it is dereferenced, our Internet browser should receive a HTTP `303 See Other` response. Furthermore, our browser should also receive a URL pointing to a document that describes the resource, in this case, Tom Heath.

To test this, let us paste the above URI into our Web browser, and we will be taken to another URL given as below:

```
http://revyu.com/people/tom/about/html
```

which contains a HTML description about Tom.

In fact, content negotiation is also supported by Revyu: if a user agent asks for HTML format, it will receive a HTML document located at the above URL, and if it asks for an RDF format, it will receive an RDF description located at this URL:

```
http://revyu.com/people/tom/about/rdf
```

- Tags are used to create links to the datasets on the Web of Linked Data.

Obviously, the collection of reviewed items is at the center of any review site. As we have discussed, every item in this collection has been assigned a URI by

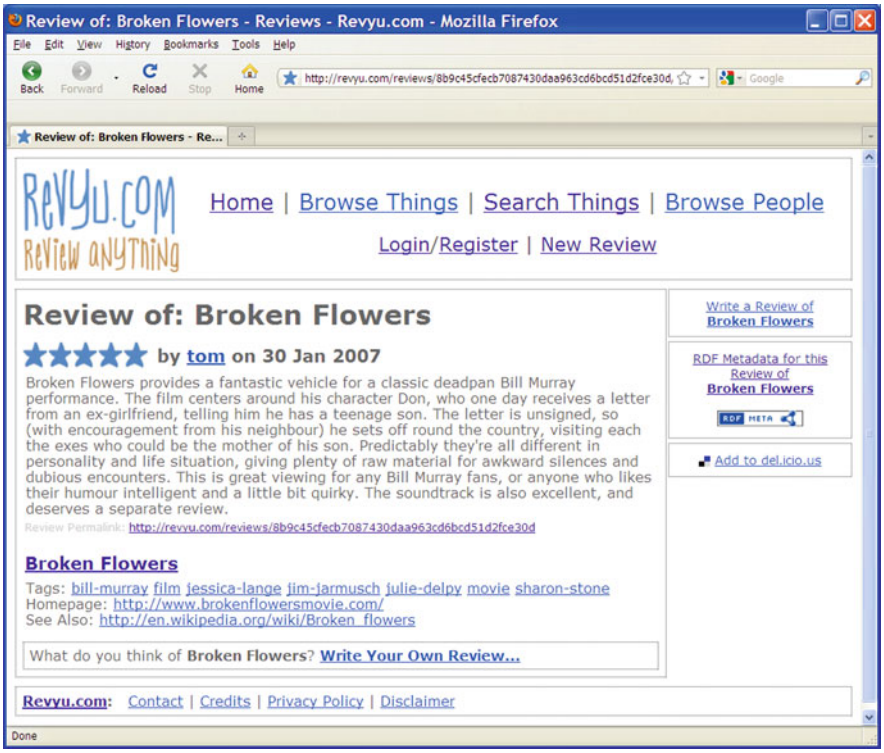

Fig. 11.20 Reviewer page of tom

Revyu. However, an isolated URI will not be of much value unless one of the following two (or both) can happen:

- it is associated with another resource URI contained in another dataset, by using `owl:sameAs` or `rdfs:seeAlso` property;
- it is associated with a type information (a class defined in an ontology), so some application can performance reasoning on this URI.

Clearly, asking the user of Revyu to accomplish either one of these conditions is not feasible: not only has the user to understand the Semantic Web technologies and standards, but also there has to be ontologies readily available which can provide sufficient coverage to any arbitrary item that may receive a review.

The solution taken by the Revyu designers is to use tags. In particular, it is up to the user to associate keyword tags to the item being reviewed. With this tag information, Revyu is then responsible for deriving type information and linking the item to a certain resource described by another dataset.

Currently two domains are covered by Revyu: books and films. More specifically, when Revyu recognizes that a new item is tagged as book, it will examine every Web link provided by the reviewer at the time the review is submitted. For example, the

reviewer may have provided a link from Amazon which contains some information about the book. When examining this link, Revyu parses the Web document downloaded from the link and attempts to extract an ISBN number embedded in the document. If Revyu can find an ISBN number, it will conclude that the reviewed item is indeed a book and will assert a corresponding `rdf:type` statement in the generated RDF statements.

As one example, List 11.7 shows some generated RDF statements for the reviewed book titled *The Unwritten Rules of Ph.D. Research*. The reviewer has provided the related link from Amazon, which contains the ISBN number (line 23). Based on this information, line 26 has been added by Revyu to establish the type information for this item.

List 11.7 RDF statements generated by Revyu (a book review)

```
1: <?xml version="1.0" encoding="UTF-8" ?>
2: <rdf:RDF
3:    xml:base="http://revyu.com/"
4:    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6:    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7:    xmlns:owl="http://www.w3.org/2002/07/owl#"
8:    xmlns:dc="http://purl.org/dc/elements/1.1/"
9:    xmlns:dcterms="http://purl.org/dc/terms/"
10:   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
11:   xmlns:foaf="http://xmlns.com/foaf/0.1/"
12:   xmlns:rev="http://purl.org/stuff/rev#"
13:   xmlns:tag=
13a:        "http://www.holygoat.co.uk/owl/redwood/0.1/tags/"
14:   xmlns:ns1="http://www.hackcraft.net/bookrdf/vocab/0_1/">
15:
16: <rdf:Description
16a:     rdf:about="things/the-unwritten-rules-of-phd-research">
17:   <rev:hasReview rdf:resource=
17a:       "reviews/82825d6cec2a2267c541848397e1605ab0042af0"/>
18:   <tag:tag rdf:resource=
18a:       "taggings/82825d6cec2a2267c541848397e1605ab0042af0"/>
19: </rdf:Description>
20:
21: <owl:Thing rdf:about=
21a:     "things/the-unwritten-rules-of-phd-research">
22:   <rdfs:label>The Unwritten Rules of Phd Research, by Gordon
22a:              Rugg and Marian Petre </rdfs:label>
23:   <rdfs:seeAlso rdf:resource=
23a:                "http://www.amazon.co.uk/Unwritten-Rules-Phd-
23b:                 Research/dp/0335213448/"/>
24:   <foaf:homepage rdf:resource=
24a:        "http://mcgraw-hill.co.uk/openup/unwrittenrules/"/>
25:   <owl:sameAs rdf:resource=
25a:       "http://www4.wiwiss.fu-berlin.de/bookmashup/
25b:        books/0335213448"/>
```

```
26:   <rdf:type rdf:resource=
26a:      "http://www.hackcraft.net/bookrdf/vocab/0_1/Book"/>
27: </owl:Thing>
28:
29: <rdf:Description rdf:about=
29a:     "taggings/82825d6cec2a2267c541848397e1605ab0042af0">
30:  <rdfs:label>A bundle of Tags associated with this Thing, de
30a:         fining when they were added and by whom</rdfs:label>

31: </rdf:Description>
32:
33: </rdf:RDF>
```

If a reviewed item is tagged as movie or film, Revyu will issue a query against DBpedia's SPARQL endpoint with the goal of finding any instance data whose type is given by `yago:Film` and also has the same name as the reviewed item. If this is successful, Revyu will conclude that the reviewed item is indeed a movie, and an `rdf:type` statement will be generated. For example, once the review for movie *Broken Flowers* is submitted, Revyu is able to confirm that this item is the movie named *Broken Flowers*, and List 11.8 shows the RDF statements generated for the item.

List 11.8 RDF statements generated by Revyu (a movie review)

```
1: <?xml version="1.0" encoding="UTF-8" ?>
2: <rdf:RDF
3:    xml:base="http://revyu.com/"
4:    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6:    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7:    xmlns:owl="http://www.w3.org/2002/07/owl#"
8:    xmlns:dc="http://purl.org/dc/elements/1.1/"
9:    xmlns:dcterms="http://purl.org/dc/terms/"
10:   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
11:   xmlns:foaf="http://xmlns.com/foaf/0.1/"
12:   xmlns:rev="http://purl.org/stuff/rev#"
13:   xmlns:tag=
13a:       "http://www.holygoat.co.uk/owl/redwood/0.1/tags/"
14:   xmlns:ns1=
14a:       "http://www.csd.abdn.ac.uk/~ggrimnes/dev/imdb/IMDB#">
15:
16: <rdf:Description rdf:about=
16a:  "things/broken-flowers-film-movie-bill-murray-jim-jarmusch-
16b:   sharon">
17:    <rev:hasReview rdf:resource=
17a:       "reviews/8b9c45cfecb7087430daa963cd6bcd51d2fce30d"/>
18:    <tag:tag rdf:resource=
18a:        "taggings/8b9c45cfecb7087430daa963cd6bcd51d2fce30d"/>
19: </rdf:Description>
20:
```

```
21: <owl:Thing rdf:about=
21a:    "things/broken-flowers-film-movie-bill-murray-
21b:     jim-jarmusch-sharon">
22:    <rdfs:label>Broken Flowers</rdfs:label>
23:    <rdfs:seeAlso rdf:resource=
23a:          "http://en.wikipedia.org/wiki/Broken_flowers"/>
24:    <foaf:homepage rdf:resource=
24a:          "http://www.brokenflowersmovie.com/"/>
25:
26:    <owl:sameAs rdf:resource=
26a:         "http://dbpedia.org/resource/Broken_Flowers"/>
27:    <rdf:type rdf:resource=
27a:  "http://www.csd.abdn.ac.uk/~ggrimnes/dev/imdb/IMDB#Movie"/>
28: </owl:Thing>
29:
30: <rdf:Description rdf:about=
30a:     "taggings/8b9c45cfecb7087430daa963cd6bcd51d2fce30d">
31:   <rdfs:label>A bundle of Tags associated with this Thing,
31a:              defining when they were added and
31b:              by whom</rdfs:label>
32: </rdf:Description>
33:
34: </rdf:RDF>
```

As you can see, line 27 is added to identify the type of the reviewed item.

11.4.1.2 Revyu: Why It Is Different

Revyu is different from other review sites. For books and movies, Revyu assigns a URI to the item being reviewed and also generates an RDF document to represent the review itself. Furthermore, Revyu searches against existing linked datasets and automatically creates links to these external datasets whenever possible. For example, line 25 of List 11.7 and line 26 of List 11.8 are the links to other datasets. As a result, these links have turned both these two RDF documents into newly produced Linked Data on the Linked Data Web.

In fact, changing the submitted review into structured data not only adds new elements to the Linked Data Web, but also makes it much easier for any application that attempts to consume the review results.

For example, to get the review for a given item, all the application has to do is to query the Revyu dataset by issuing SPARQL query via Revyu's SPARQL interface, and the result is an RDF document that can be easily processed. Compared to Amazon, where the review data has to be obtained by using its own APIs (Amazon Web services), the benefit is quite obvious. We will come back to this point later in this chapter.

Besides producing new Linked Data, Revyu also consumes existing Linked Data on the Web to enhance its user experience.

To see this, let us go back to line 25 of List 11.7 and line 26 of List 11.8. Since these statements are links that point to other linked datasets, one can simply apply

the Follow-Your-Nose method to retrieve additional information about the reviewed item. In fact, this is exactly what Revyu has done. For example, by following the link on line 26 of List 11.8, Revyu was able to obtain this movie's entry in DBpedia, which contains the URI of the film's promotional poster and the name of the director, etc. All this additional information has been displayed on the page about this film, as shown in Fig. 11.19.

Clearly, this automatic consumption of the existing Linked Data has greatly enhanced the value of the whole site, without requiring this information to be manually entered by the reviewer.

Similarly, Revyu can fetch more information about the book from RDF Book Mashup dataset (line 25 of List 11.7), such as the book cover and author information, which is also displayed on the Revyu page about the book. Again, all this does not ask any extra work from the reviewer, but is very valuable to anyone who is reading these reviews.

Furthermore, the same idea can be applied to a reviewer. Recall each reviewer is assigned an URI, and a simple RDF document is created for this reviewer. If a given reviewer has an existing FOAF document, an `rdfs:seeAlso` statement will be included in the RDF description. For example, List 11.9 shows the RDF document created for me by Revyu.

List 11.9 RDF statements generated by Revyu for a reviewer

```
1: <?xml version="1.0" encoding="UTF-8" ?>
2: <rdf:RDF
3:    xml:base="http://revyu.com/"
4:    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
6:    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
7:    xmlns:owl="http://www.w3.org/2002/07/owl#"
8:    xmlns:dc="http://purl.org/dc/elements/1.1/"
9:    xmlns:dcterms="http://purl.org/dc/terms/"
10:   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
11:   xmlns:foaf="http://xmlns.com/foaf/0.1/"
12:   xmlns:rev="http://purl.org/stuff/rev#"
13:   xmlns:tag=
13a:        "http://www.holygoat.co.uk/owl/redwood/0.1/tags/">
14:
15: <foaf:Person rdf:about="people/liyang">
16:   <foaf:mbox_sha1sum>
16a:     1613a9c3ec8b18271a8fe1f79537a7b08803d896
16b:  </foaf:mbox_sha1sum>
17:   <foaf:nick>liyang</foaf:nick>
18:   <foaf:made rdf:resource=
18a:        "reviews/cbe1fd43cf7de69ee0530fe65593d6d77d03daed"/>
19:   <foaf:made rdf:resource=
19a:        "reviews/436f699d347d433315507923664cf567fe872a59"/>
20:   <rdfs:seeAlso
20a:        rdf:resource="http://www.liyangyu.com/foaf.rdf"/>
```

```
21: </foaf:Person>
22:
23: </rdf:RDF>
```

Revyu will dereference the URI on line 20 and query the resulting FOAF document for information such as my photo, location, home page, and interests. This information then automatically shows up at my profile page without me entering them at all.

As a summary, Revyu is a simple and elegant application that makes use of existing Linked Data to enhance its user experience without asking extra work from its users. Although it is not a large-scare Linked Data application, it does show us the benefits offered by the Web of Linked Data.

11.4.2 Web 2.0 Mashups vs. Linked Data Mashups

In the previous section, you have seen an interesting application that consumes Linked Data on the Web. However, up to now, consuming Linked Data on the Web in a large scale still remains an open question, and the business value of Linked Data Web can be better appreciated only through these large-scale applications.

However, researchers and developers in the field of the Semantic Web are still very optimistic about its future, and one of the reasons is based on the comparison of the so-called Web 2.0 mashup to semantic mashup. In fact, if Web 2.0 mashups can continue to remain in high demand in the environment of traditional Web, semantic mashups under the concept of the Semantic Web will sooner or later be the real data mashup tool that everyone will use, simple because it is much more easier, much more efficient, and much more scalable. The rest of this section will explain this conclusion in detail.

Exactly what is a mashup? In a very simple sentence, a *mashup* is a Web application that collects structured data produced by third parties through APIs offered by these parties and processes the data in some way and then represents the data back to the user in a form that differs from its original look-and-feel. Normally, a mashup application will either enhance the visual presentation of the data or offer added value to its users by combining the data from different sources or both. This concept is more related to Web 2.0, where more and more Web sites expose their data via their APIs.

A typical mashup could be something like this: a shopbot can be coded to retrieve the price of a given product (such as a camera with a specific make and model number) from Amazon.com by accessing its published APIs. At the same time, the same shopbot can also retrieve the price of the same product from BestBuy.com (assuming BestBuy has also published their APIs), and these two prices can be compared and returned to the user so the user can decide where to buy the product. The shopbot can even retrieve the prices from these two vendors periodically, so the user can see the change of these prices over a certain amount of time, and therefore

can buy the product when its price is going down and reaches a relatively stable stage. Here the added value is obvious, and we can expand this shopbot in many ways, such as including more vendors and more products.

This all sounds correct and feasible. However, when you really set off to construct such a shopbot, you will soon discover its limitations:

- poor scalability of the method itself

  Since different vendors publish different APIs, this is a constant learning process. You will have to learn each set of APIs, and once a new vendor is available, you will have to learn a new set of APIs again. Therefore, the construction of such a mashup is not scalable and its maintenance will be quite expensive as well.

- limited coverage at most

  Obviously, the shopbot only understands the APIs that you have coded for it to understand, it cannot do any simple explore on its own. The data coverage is therefore very limited and any decision based on this shopbot will probably not be optimal either.

- lost links to channel back to the data providers

  Once the data are retrieved and consumed by the shopbot, the link between the shopbot and the original data provider is lost; a user cannot link back to the original data providing site. Even in the case where we have decided to put some links channeling back to the data providers, these links are shallow links at their best, and they will not be able to link back to the precise locations of those particular data components. In addition, a mashup site supported by this shopbot only shows the price. What if the original site offers some free gifts if you buy it now? If there were a link back to this particular product, the user might be able to catch this offer. Even more importantly, the links that channel back to the original data provider mean more incoming traffic, which means significant chances for some potential business value.

Now, with all the above being said, let us take a look at what would be the case if a mashup application is developed under the environment of Linked Data Web. In fact, Revyu is such a mashup: it retrieves data from external Web sites (DBpedia, RDF Book Mashup, etc.) to enhance its user experience, a typical way that a mashup should work. More specifically,

- good scalability of the method itself

  Under the Web of Linked Data, structured data are expressed by using RDF graphs and standards, which is the only set of standards across all the sites, and there is no specific APIs for each site to expose its structure data. Therefore, constructing the mashup and maintaining the mashup is quite scalable, there is no need for constant learning of new APIs.

- unbounded coverage of datasets

Obviously, Web 2.0 mashups work against a fixed set of data sources; Linked Data applications operate on top of an unbound, global data space. This enables them to deliver more complete answers as new data sources appear on the Web.

- crucial links to channel back to the data providers

  In Linked Data mashups, all the items (resources) are identified by URIs, each of which may be minted and controlled by the data provider. If a user looks up one of these URIs, the user may be channeled back to the original data provider; it is then up to the data provider to publish appropriate content to further direct the incoming traffic. This linking-back capability is a key difference between Web 2.0 mashups and Linked Data mashups, and this is where the potential business value could be.

Besides the above, Linked Data mashups also offer a chance to their users to chain up almost unlimited resources. More specifically, each item in the mashup is identified by a URI, which can be linked to other resources in other datasets, and the links themselves are also typed. As a result, you can choose to follow a specific link and visit a specific resource, which further takes you to other resources in other datasets, so on and so forth. As this point, you should be able to appreciate the value of this unlimited linkage, without the need of much explanation at all.

11.5 Summary

In this chapter, we have learned Linked Open Data. It is another example of the Semantic Web technologies at work, and it is quite different from other examples we have learned. Instead of adding semantics to the current Web (either manually or automatically), its idea is to create a machine-readable Web all from the scratch. It is therefore also called the Web of Linked Data, or the Linked Data Web.

First off, understand the following about Linked Open Data:

- its basic concept and basic principles;
- its relationship to the classic view of the Semantic Web, i.e., it can be viewed as an implementation of the vision of the Semantic Web.

Second, understand the two major topics about Linked Open Data: how to publish Linked Data on the Web and how to consume Linked Data on the Web.

About how to publish Linked Data on the Web, you need to understand the following:

- how to mint URIs for resources, what is the difference between 303 URIs and hash URIs;
- how to create links to other datasets, and how to make sure your data is published as Linked Data, i.e., what are the minimal requirements of being Linked Data on the Web;
- remember to use a validator to make sure you have done everything correctly.

About how to consume the Linked Data, you need to understand the following:

- consuming Linked Data means discovering Linked Data on the Web, accessing the Web of Linked Data, and building applications on top of the Web of Linked Data;
- there are Semantic Web search engines you can use to discover Linked Data on the Web;
- there are Semantic Web browsers you can use to access the Web of Linked Data manually;
- you can also use SPARQL endpoints to access the Web of Linked Data, in addition, you can programmatically access the Web of Linked Data from within your applications;
- the ultimate goal is to create powerful applications that make use of the Web of Linked Data.

Finally, to show you how to build applications on top of the Web of Linked Data, we have included Revyu as one such example. Make sure you understand how Revyu makes use of the Linked Data on the Web, and more importantly, hope this can serve as a hint to you, so you can come up with possible applications of your own to show the power of the Web of Linked Data.