The Semantic Web and an Introduction to RDF*

Thomas Krichel

August 8, 2002

Abstract

The semantic web is an effort promoted by the World Wide Web Consortium (W3C) to make more machine-processable information available on the web. The key infrastructure to build the Semantic Web is the Resource Description Framework (RDF). This is a proposed standard for metadata encoding. This chapter was mainly composed from documentation released by the W3C. It's prime purpose is to provide an introduction to RDF for a non-technical audience, augmented by a—somewhat personal—discussion of the value of RDF. I argue that although RDF is technically highly competent, it is not sure whether it will take off, because RDF implementations have too many problems as to their social and economic viability.

1   Introduction

This is an introduction to the Resource Description Framework (RDF) as proposed by the World Wide Web consortium. At the time of writing, the W3C has an RDF-Core working group concerned with RDF. In its charter at http://www.w3.org/2001/sw/RDFCoreWGCharter, we read:

> Implementor feedback concerning the RDF Model and Syntax Recommendation points to the need for a number of fixes, clarifications and improvements to the specification of RDF's abstract model and XML syntax.
>
> ...
>
> The role of the RDF Core WG is to prepare the way for such work by stabilizing the core RDF specifications. The RDF Core WG is neither chartered to develop a new RDF syntax, nor to reformulate the RDF model. However, the group is expected to re-articulate the RDF model and syntax specification in such a way as to better facilitate future work on alternative XML encodings for RDF.

What these words mean precisely is subject to interpretation, but I think we can be on the safe side to say that an introduction to the topic—as this chapter aims to provide—is not going to be fundamentally altered by what the working group will decide. Indeed, the purpose of this chapter is twofold. As an introduction, it is supposed to give you, the reader, an idea of whether you should make use of RDF in your work as an information professional. But even if you do decide not to use it, I hope that the chapter stimulates your interest in the underlying principles that motivate RDF, or that the concepts discussed here will be of some use to you.

Personally, I was introduced to RDF by the co-chair of RDF-Core working group, Dan Brickley, in a London pub on the September 2, 1998. A written chapter can not reach the same same quality of immediacy. However, I will stay faithful to Dan's ways by adopting an informal style to the discussion of these issues. They tend to be rather dry and abstract, anyway. Some familiarity with XML or a similar markup language is assumed . As far as the structure is concerned, I will proceed in three sections. In a Section 2, I shall explain the motivation behind RDF. In Section 3 I shall be studying the RDF model. In a Section 4 I turn to RDF Schema. In Section 5, I will discuss the significance of RDF, mainly from the point of view of web annotations. A final Section 6 concludes.

---

2 Motivation

The W3C has been promoting the idea of a semantic web since the late 90s. Semantic means—according to the Web edition of Webster's at http://www.webster.com— "of or relating to meaning in language". The semantic web thus relates to the idea that there should be meaning in the web. This may be puzzling to you since you use the web everyday to gain information from it. This web that you use today can be thought of as the first stage of the web. In its most complete form, the web offers a universal communication medium. It allows users to transport any digital entity of information. Any device that is used to produce and receive such digital information may be connected to the web. Thus the web has a universal character for human communication. It is not meant for machine communication. Thus the web only transports raw data, and the interpretation of that data is made by humans. The web itself does not carry any meaning.

The idea of the semantic web is referred to by Berners-Lee (1998) as "a plan for achieving a set of connected applications for data on the Web in such a way as to form a consistent logical web of data (semantic web)", The goal—as outlined there—is to move away from the present web—where pages are essentially constructed for use by human consumption—to a web where more information can be understood and treated by machines. In that case, machines may make assertions out of primary data found on the web. Suppose, for example, that we fit each car in New York City with a device that lets a reverse geographical position system reads its movements. Suppose in addition, that another machine knows what the weather will be like or some other phenomenon that impacts on traffic. Assume that a third has the public transport timetables. Then, data from a collaborative knowledge picture of these machines can be used to advise me on what type of means of transport I should take to reach a certain destination within the next few hours. Such advice can not be given by the current web because of several limitations.

- The data is not being made available.

- The data is not being entered in machine-readable form.

- The data is not being laid out in machine processable form.

- There is no browser where you can enter your problem easily, but precisely.

It is clear that when the data is not there, nothing can be done. But even if the data is there, it can not be used until it is machine processable. The computer systems doing the calculations required for the traffic advisory are likely to be controlled by different bodies, such as the city authority or the national weather service. Thus there must be a way for software agents to process the information from the machine where it resides, to proceed with further processing of that information to a form in which a software agent of the final user can be used to query the dataset. At this level another standard comes into play, the XML markup language. Since you are assumed to be familiar with XML, you will wonder why we need a thing called RDF to encode the information.

Could the information not be encoded in XML. The answer is "yes, but..." To understand this, let us move away from the real-life large-scale example of the traffic advice system to a another real-life example, albeit on a more intimate scale. Let us suppose, for example, that we want to express the fact that Thomas Krichel has a former lover, called Sophie C. Rigny. There are several ways to do this in XML. What about

```
<person><name>Thomas Krichel</name><hasformerlover>
Sophie C. Rigny<hasformerlover></person>
```

or

```
<person><name>Thomas Krichel</name><hasformerlover><person>
<name>Sophie C. Rigny</name></person><hasformerlover></person>
```

or

```
<person>Thomas Krichel<relationship type="former lover">
<person>Sophie C. Rigny</person></relationship></person>
```

2

I think you start to see the problem. If I meet you on the street, and I tell you, "Show me your list of former lovers and I show you mine", and you agree, then we are fine. We can agree on an XML schema—something that formalizes any of the above syntaxes. I send you my data, you send me yours. A human observer would be able to get some idea if she understands English and if the tags are as self-explanatory as in the example above. But a machine would not be able to make anything out of this, because it would not be able to figure out which representation we have agreed upon.

Yet if the machine could understand it, it could process the data in a sensible way. For example, since the relation "former lover" is symmetric, a machine would conclude that Sophie C. Rigny has a former lover called Thomas Krichel. If, in addition, the machine knows that Thomas Krichel is a human being of male sex, it would conclude that it is highly likely Sophie Rigny is a human too, and that her sex is probably female. RDF itself does not provide a framework for deductive reasoning, but it provides a framework by which such information can be channeled to a machine.

To understand the message above, a machine would need to know what "Thomas Krichel" is, what "former lover" is, and what "Sophie Rigny" is. RDF provides tools to make that happen. Note, that RDF does not provide a vocabulary of relationship term such as "hasformerlover", "livesataddress" etc. Instead it provides for a framework where such vocabulary can be built, and exchanged by communities. Thus it is not a metadata format, but provides a framework that communities can use to *express* metadata. In other words, it provides a system of logic in data-organizing, which, when applied to examples such as the one that I gave above, would always result in the use of ways of expressing the information that would ease its processing by machines. This in turn would make data understandable by different machines. An extension of RDF, RDF Schema, can be used to *build* metadata formats. It will be introduced in Section 4.

3 RDF Model

RDF is an abstract model. As such it is syntax independent. The model has three types of things that it is interested in.
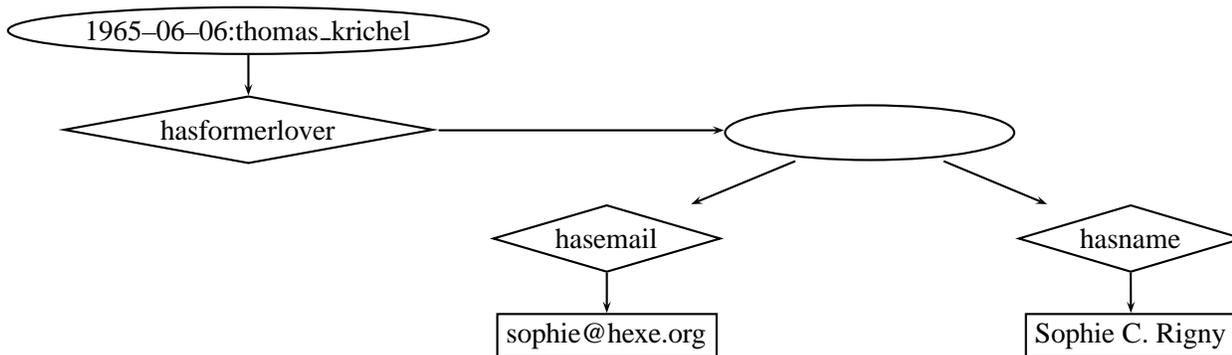
First, there are resources. A resource can first be thought of as something that can be retrieved from the web, such as a web page, or a email, or a picture. However, anything else that has a URI identifier, see Berners-Lee, Fielding, and Masinter (1998), can be a resource. A URI is a rather general identifier. Many existing identifiers can be restated as a URI. For example, the RePEc digital library, see http://repec.org, has an identifier for me as a person, I am RePEc:per:1965--06--05:thomas˙krichel. This identifier could be registered as part of a RePEc URI scheme, or as part of an oai:RePEc URI scheme. I abstract from these administrative details. I will assume that the RePEc identifier above is a URI reference to me. For reasons of space, I shall abbreviate it as "1965–06–05:thomas_krichel". This code is a URI reference to me. I become a resource through the reference. Since URI references can be defined for anything, anything can be a resource.

Second, there are properties. Properties are specific aspects of a resource. Each property has its own meaning. It might only admit a certain range of values, or be attached only to certain resources, but such constraints are not part of the RDF model. These constraints are part of RDF Schema.

Third, there are the values that the property takes. There are two kinds of values. First the value may be a literal, i.e. a string that means itself. Or, second, it may be a new—possibly anonymous—resource. Let us first look at the first case, where the value is a string. I use an entity-relationship diagram with the resource in an oval box, the property name in a diamond box and the value in a rectangular box.
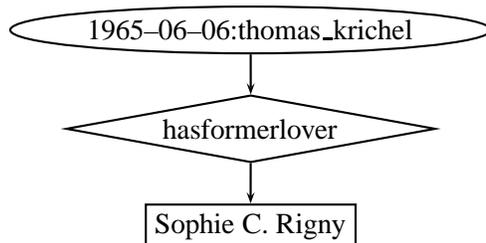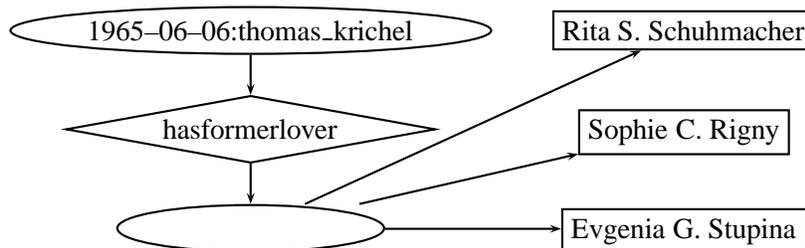
In this case, we see one RDF statement. An RDF statement is a combination of a resource in the oval, a property in the diamond, and a value in the rectangular box. Now let us illustrate the case where value is an anonymous resource, that has further properties.

```
      ( 1965–06–06:thomas_krichel )
                  |
                  v
       < hasformerlover >---------------->  (        )
                                            /          \
                                           v            v
                                   < hasemail >    < hasname >
                                        |               |
                                        v               v
                                [ sophie@hexe.org ]  [ Sophie C. Rigny ]
```
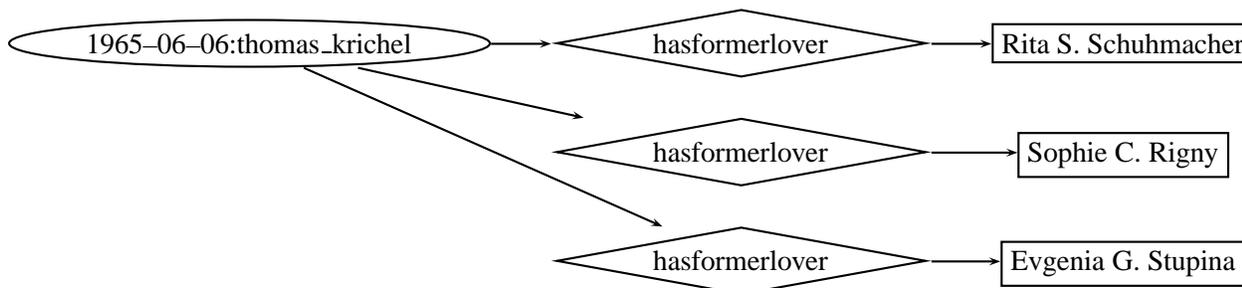
Here the value of the statement is an anonymous resource, that is the resource described in two other statements.

That is almost the whole basic RDF model. There are two more things to cover. First, there is a container structure that covers repetitions. There are three types of containers. The first one, a "bag", is an unordered list of values. The second, a "sequence" is used when order is significant. The third, an "alternative" is used to represent alternatives for a single value.

```
    ( 1965–06–06:thomas_krichel )      [ Rita S. Schuhmacher ]
                  |
                  v
       < hasformerlover >                [ Sophie C. Rigny ]
                  |
                  v
              (        )--------------->  [ Evgenia G. Stupina ]
```

With several former lovers, problems can not be far away. Assume that we know that the values are a sequence. What does that actually mean? The above example does have an order, but which one it is remains a mystery that I shall not reveal. Assume that it would be an alternative? What would that mean? I suppose it starts dawning an the reader that we go down a slippery slope here...
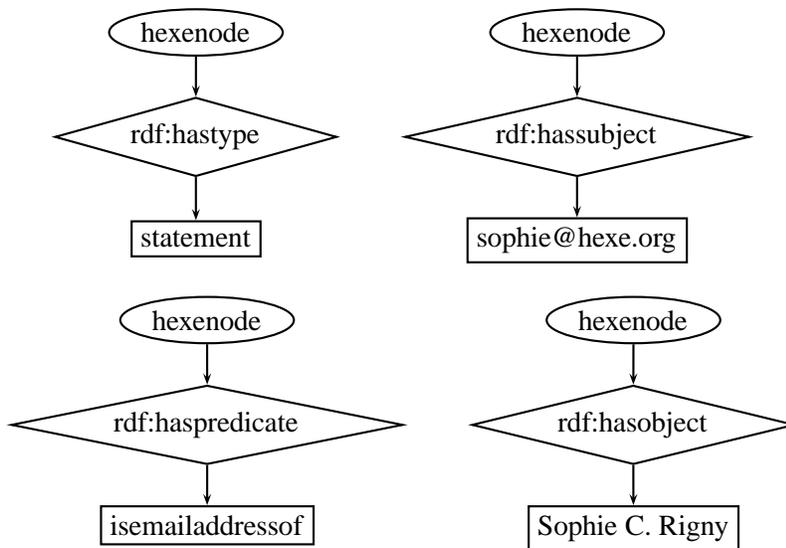
Another tricky issue is not really obvious in the example above, it is the one regarding the status of the container. A human will tend to assume that the four values are meant to be distinct former lovers of our hero. However a machine could also adopt the view that the four women, collectively form a group that is the former lover of "1965–06–06:thomas_krichel". Unfortunately, the default interpretation, in RDF, is the latter one, rather than the former. To express what I really want to say in RDF, I must use something like

```
  ( 1965–06–06:thomas_krichel )--->< hasformerlover >---->[ Rita S. Schuhmacher ]

                               ---->< hasformerlover >---->[ Sophie C. Rigny ]

                               ---->< hasformerlover >---->[ Evgenia G. Stupina ]
```

How easy it is to get this wrong! And this is only the container architecture, a first level of complication. At the next level up, there is what is known as "higher-order" statements. These are statements about statements. RDF aficionados refer to such statements as "reifications". The reified statement is considered to be an anonymous resource.

sophie@hexe.org

isemailaddressof

Sophie C. Rigny

hasboolean → false
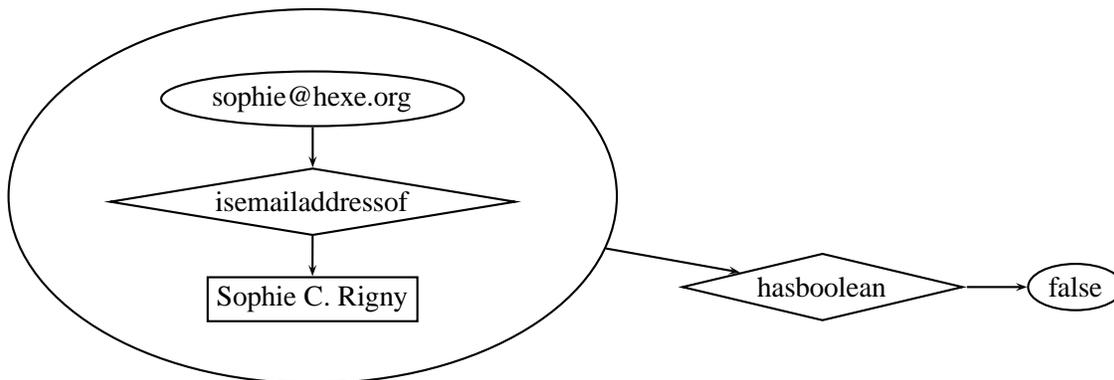
In fact, as far as RDF is concerned, this statement above is transformed into four basic statement. The resource that is the reified statement is decomposed, using RDF reserved vocabulary in an with an "rdf" namespace prefix. If we label the reified statement as "hexenode"—any internal resource identifier would do—we get

hexenode

rdf:hastype

statement

hexenode

rdf:hassubject

sophie@hexe.org

hexenode

rdf:haspredicate

isemailaddressof

hexenode

rdf:hasobject

Sophie C. Rigny

This somewhat convoluted transformation is used so that the initial statement can be reconstructed from the RDF logic. The fact that a statement is added saying that it is fictitious, does not imply a change in the nature of the original statement. The idea is that statements added to the existing RDF data should not be able to change the nature of the original statement. Note that a special vocabulary has to be defined, that expresses the concept of "subject", "predicate" and "object" in such a way that all RDF applications that can read a certain application will understand it. Fortunately, there is only one standard syntax for expressing RDF, and that is XML[1]. Therefore referring to these special vocabularies is easy, we can use XML namespaces for that. Readers who are not familiar with XML namespaces may consult, Bray (1999) for a good introduction.

To summarize, RDF is a syntax-independent way to describe things that are identified. Its mathematical structure is a graph. A graph, in mathematical terminology, is something that is composed of nodes, some of which are connected by lines. Formally, a graph is a binary relation on a set of nodes. If this relation is symmetric, the graph is said to be undirected. This is not the case in RDF, which strictly distinguished between subjects (resources) and objects (property values). Therefore RDF is a directed graph. There are three kinds

---

[1]To be precise, there are a couple of ways of writing down RDF in XML, but I will skip over this, because I don't want to go into the details of the RDF XML syntax.

of node in any RDF graph: urirefs, literals, and blank nodes. A uriref is a URI referring to something that is to be described. A literal is basically just a string of characters. Blank nodes are assumed to be unique to the graph. Blank nodes have no label and are unique to the specific graph. The lines that connect the nodes within the graph correspond to the properties of resources. These lines are always directed, they go from the resource to the property value.

4   RDF Schema

RDF does not provide a way to specify resource and property types. Once you have some instance of a resource, you can associate a property with it. But you can not express what kind of property it is, and you can not specify what kind of resource it is. Doing that is the job of RDF Schema. Much of the inspiration behind RDF schema comes from object-oriented programming. For those who are unfamiliar with the concept, let me explain it a little, those who are familiar with it just skip to the next paragraph. In conventional programming, you define variables, and you define functions that operate on the variable. In object-oriented programming, the variables and the functions that belong together are tied together in an object class. Thus a class "humans" groups all human beings in the in the word, and each person would be an object in that class. Its properties are like the "size", "name", "place of birth", and the things that it can do, like "make love", "teach" would be the methods, i.e. functions that are internal to the object. One important feature of object-oriented programming is that it is much easier to reuse code. If I define a new object "composer" and say it is a human, then the composer immediately inherits all the methods of the human, to which specific methods like "orchestrate", that only composers do, could be added. The principle that objects of the subclass have all the properties of the objects in the higher class is called inheritance.

RDF schema develops classes for both resources and properties. Defining objects in classes is very attractive for resource description because it allows the more special classes of resources to be derived from more general ones. Since everything that RDF describes is a resource rdfs:Resource is the most basic resource class. Every resource is an rdfs:Resource. The property rdf:type is used to say that a resource is of a certain type. The idea of a type of resources is expressed in the concept of a class. A class is a resource whose type is rdfs:Class. Every resource belongs to the class rdfs:Resource and all the classes used in RDF schema are sub classes of resources. Here is a—hopefully—self-explanatory graphical example, for classes and subclasses.
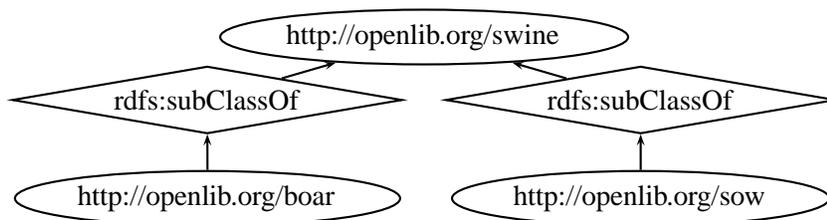

This is the kind of information that is conveyed by RDF schema. It says that there is a class "swine" and two subsclasses "boar" and "sow". It implies that every resource that is of type "boar" or "sow" is also a resource of type "swine".

RDF Schema also allows one to specify properties. In fact, as far as RDF Schema is concerned, properties are a special type of resources. The most important property is rdf:type which says that a resource is a member of a certain class. rdf:subClassOf is another important property that applies to RDF resources that are of rdf:type class. This property has already been referred to in the last graph. A simple hierarchy of properties and sub-properties can be specified, see the following graph for an example.
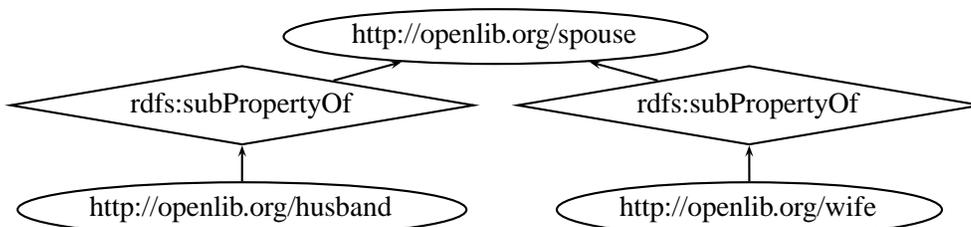
In this case, if there is a resource with the property "husband" that takes the value "Justin", then the same resource also has a spouse that take the value "Justin", by virtue of "husband" being a subproperty of "spouse".

Properties have two important properties themselves. One is the domain of a property. The "domain" of a property means the—zero or more—classes of resources to which the property may be applied. For example, the property "sex" may be applied to man and beast and nouns nouns in many languages. If a property has no explicit domain, it may be applied to any resource. A second important property of a resource is the range of values that it can take. The range constraints the values that the property can take. If the property has no range, it can take any value. If the property has one or more range properties, each the values of these range properties must be of type rdfs:Class. In that case the classes indicate the types of objects that the value of the property can take. A graphic illustration of domain and range may make this a little more clear.
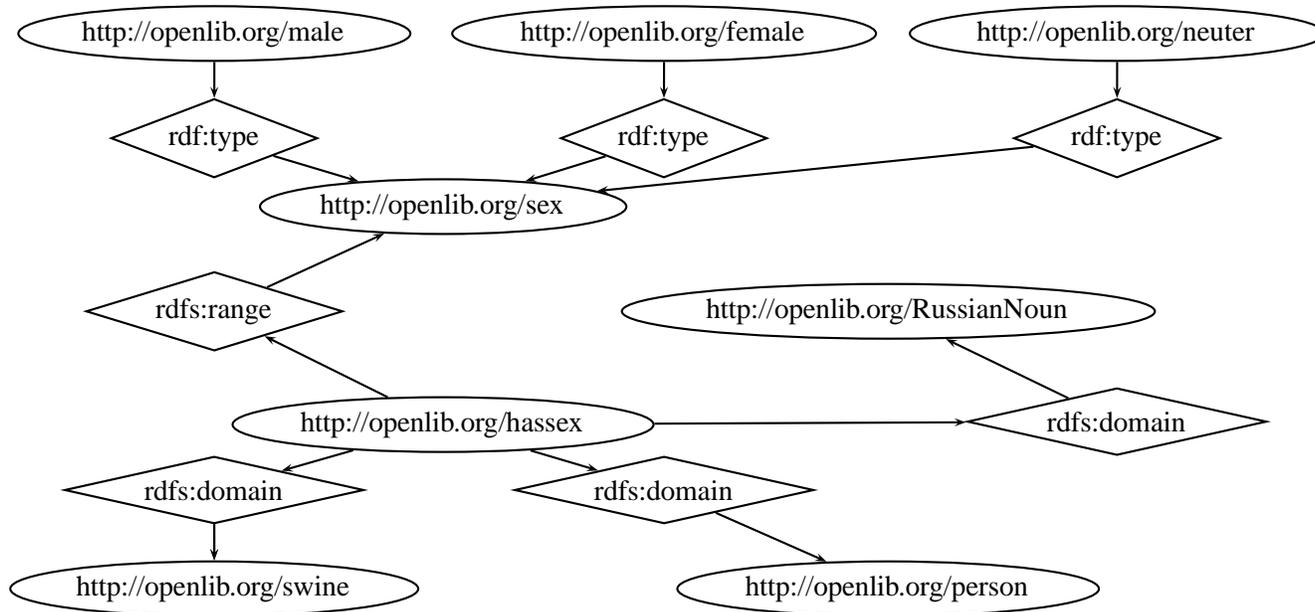
Here I have defined a property "hassex" the value of which is something on the type "sex". I then say that three resource, "male", "female" and "neuter" are of type "sex". Finally, I say that anything that is a swine, a person or a Russian noun can have a sex. Note that the description of the class structure and property structures is done in entirely in RDF. This shows how general RDF is as a descriptive structure.

Finally, those who are familiar with object-oriented programming should notice that there is an important difference in the way RDF schema treats inheritance. Typically, within object-oriented programming, when we define a class and the attributes that its members may take the attributes are local to the object. In contrast, properties in RDF schema have global scope. This is in keeping with the general philosophy that within the web anybody can make a new statement about any resource. Thus, nothing prohibits anyone to define a new resource, say "shmoo" and say that it is also of type "http://openlib.org/sex". The universal nature of the web is the basis for this philosophy behind this—admittedly rather liberal—way to proceed.

5 Discussion

The creation of RDF seemed to be primarily motivated for the purpose of describing web resources. The idea was that more metadata will make the web a place where it is easier to find things. If this was the implicit promise of RDF, then clearly RDF has not lived up to it.

As far as the web is concerned, most of the search engines that are on the market right now, do not use any metadata. In particular, Google, at http://www.google.com seems to be the most widely used engine, does not use metadata. There are two problems with metadata for the web. First, there is the problem of the vocabulary to use. Second, there is the problem of encoding that vocabulary. In this chapter, we are concerned with the latter problem, rather than the former. However, the two problems are related. If there is no meaningful and useful vocabulary, you can not encode anything. But even if there is a useful and meaningful vocabulary (say Dublin

Core) and if there is a way to encode it (say RDF), that still does not make for a meaningful collection until there are incentives for a majority of providers to produce good metadata. Good metadata has to be meaningful and factually correct. Finally, the user must be willing, and able to use that metadata. It is in the latter stage, the good metadata stage, that web resource description falls down, and RDF, as a tool meant to realize it, loses a good deal of its appeal. Let us dwell a little on these points.

Even assuming that RDF has been a miraculous success and the web is now annotated with good metadata—whatever that requires, we will look at these requirements shortly—it is only successful if the user uses it. But if anything can sum up the studies that look at how people use the web, it is that people spend a short amount of time on any one page, and that the queries they issue are extremely primitive. Making use of all this metadata—even if it were assembled—would require a much more complicated search page, and it it unlikely that the average user will want to spend time to learn it. Sure, users tend to get more sophisticated the longer they use a certain information system, but since the information needs answered by the web are so broad, there is not much hope in users learning to express their queries with sufficient structure such that a machine can make sense it. Just look at the problem of shopping. I want to know where I can get convenient access to an item that I can buy. I need to tell the machine where I am, I need to tell the machine what I want to buy, and I need to tell the machine what transport options I have, and, hang-on, where my friends are who could get the item for me and give it to my next time I see them. . . Entering all that data into a computer is a daunting task that is likely to cost me more effort than going to a nearby store and have a look if the item is not there by any chance. And even if I could enter the data, the privacy issues with letting a third party know all these details about me should raise more than a couple of eyebrows.

A further constraint on the usefulness of RDF is that we would need to have machines that can extract a lot of information out of very limited data. Progress on that front depends on developments in artificial intelligence. My guess is that it will take a few more years to achieve results in that area, by which time RDF may have been forgotten. The progress of artificial intelligence to date has been very slow.

For a moment assume that a semantic web browser could be constructed. Now turn to the problem of good metadata. Metadata is good if (1) it is factually correct and (2) it is meaningful. First the metadata has to factually correct. A whole industry is now laboring to get sites to the top of the search engines. If a certain term is known or suspected to be asked for a lot, such as "sex", for example, every web page owner will have an incentive to use this term in his metadata, even if this is not relevant. They can even be honest and write "Esta página no habla de sex." in the summary, they will still be indexed under that term. This is the main reason why metadata does not work. All too often users are frustrated when the page that they wish to access does not contain the term that was searched, because that term was in the metadata only. This is a severe problem that limits the usefulness of any metadata that is provided by non-trusted parties. The web, whose ultimate strength is that anyone can create pages, will have to be limited in the trust that it can place on metadata.

Recall now that metadata does not need only to be factually correct but also to be meaningful. To create meaningful metadata is difficult, even for Dublin Core or RDF aficionados. For example, in many cases, it is difficult to choose between the container architectures "bag", "sequence" and "alternative". Look at the case of authors writing an academic paper. In some disciplines, it is customary to sequence all authors by the order of importance. In others, authors are ordered by alphabetical order. In orders, like in High Energy Physics, there are large collaboratories where scores of authors can appear on the same paper. Does the order matter there? Maybe for the first few, but as we go down the list, there is a chance we reach folks who have not even read the paper. Thus the distinction between bag and sequence can be a difficult one. The same holds for the distinction between alternative and bag and sequence and alternative. Finally the question of using a container is difficult too. If we use the container then the paper has been written by its authors acting as a committee. That may not be realistic a view as far as academic papers are concerned. But now assume that we don't use the container structure. In that case all persons have the author property, and they all appear in different RDF triples. Now we have to fiddle all this back together to get the sequence/bag of authors that we need for our bibliographic service, and, since there is no semantics in the position of RDF statements in the file, any information that may have been intimated in the ordering may get lost. It's a recipe for a fine mess. Another illustration of the problem of meaningful metadata is the meaning of metadata terms. For example, what does the Dublin Core community mean by 'date'? If I have a web page of the "Mona Lisa" and the "Venus of Milo", what date do I

associate with it? The date when the "Mona Lisa" was painted? When the Venus of Milo was sculpted? When the pictures were taken? When the pictures where scanned in? When the web page was created or modified? Clearly, for a restricted set of resources it is possible to agree on meaningful qualifications for 'date' but at the scale of web resource in general, consensus on these matters would be difficult to achieve, would produce a cumbersome set of rules, and would be costly to implement.

That leads me to the next point, the point about costs. Given what we have noted above, adding good metadata, encoded in RDF to web pages, or to any other identified object—aka "resource" in RDF speak—is not a job description for the mentally retarded. The labor force of smart people is a rare and useful resource, and therefore it is expensive. This is the biggest hurdle to overcome for RDF. It makes RDF a non-starter. Its demands are way beyond the reasoning skills of the average information professional, and its implementation requires computing skills that are beyond the scope of the average computing service staff. With these demands, it is likely that RDF will stay, for the foreseeable future, confined to lab experiments[2].

To summarize the discussion in this section, there is one expression that elegantly captures the problems of RDF and Dublin Core. Berners-Lee (1999) talks about "web architecture from 50,000 feet". This is the where the current efforts have serious shortcomings. They try to build a top level application for the exchange of structure metadata between communities, before there are communities that have organized internal exchanges of metadata to any significant extent. Such organized exchanges require the construction of communities. That is a social and economic process that will depend on community leaders appearing and on the plans that they make. Given the cost of implementing RDF infrastructures, it is more likely that it will be done for high-quality resources of a specific domain. The data that will be assembled within specific domains may not need RDF for its representation and exchange. However RDF will be a useful reference model for all communities who wish to exchange structured data.

6   Conclusions

Berthold Brecht has a famous poem about the tailor of Ulm. Brecht tells us that this man threw himself from the cathedral spire—the highest church spire in the world—in 1592, and when he landed dead on the square, the bishop said to his flock "man will never fly".[3] Thus, I really want to resist the temptation to say that RDF will never work. It is not likely that the whole of the web will be covered with an RDF layer. Instead, there will be communities that use RDF, and if these communities want to open their metadata for processing by other communities—they may not wish to do that—they will surely take a close look at RDF. Thus it is in local domains where RDF will be used first.

References

Arms, William Y. (2000). Automated Digital Libraries How Effectively Can Computers Be Used for the Skilled Tasks of Professional Librarianship? *D-lib Magazine 6*. available at http://www.dlib.org/dlib/july00/arms/07arms.html.

Berners-Lee, Tim (1998). Semantic Web Road map. available at http://www.w3.org/DesignIssues/Semantic.

Berners-Lee, Tim (1999). Web Architecture from 50,000 feet. available at http://www.w3.org/DesignIssues/Architecture.

Berners-Lee, Tim, Roy T. Fielding, and Larry Masinter (1998). Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 available at ftp://ftp.isi.edu/in-notes/rfc2396.txt.

Bray, Tim (1999). XML Namespaces by Example. available at http://www.xml.com/pub/a/1999/01/namespaces.html.

---

[2]See Arms (2000) for an highly interesting discussion of this point.

[3]The poem is a bad distortion of historical facts. The real tailor of Ulm was Albrecht L. Berblinger, 1770–1822. He made his attempt at flying on 31 May 1811. He landed in the Danube and was rescued by a fisherman. Recent research suggests that Berblinger's device could actually fly.