The Semantic Web: An Introduction

This document is designed as being a simple but comprehensive introductory publication for anybody trying to get into the Semantic Web: from beginners through to long time hackers. Recommended pre-reading: the Semantic Web in Breadth.

Table Of Contents

What Is The Semantic Web?

The Semantic Web is a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. You can think of it as being an efficient way of representing data on the World Wide Web, or as a globally linked database.

The Semantic Web was thought up by Tim Berners-Lee, inventor of the WWW, URIs, HTTP, and HTML. There is a dedicated team of people at the World Wide Web consortium (W3C) working to improve, extend and standardize the system, and many languages, publications, tools and so on have already been developed. However, Semantic Web technologies are still very much in their infancies, and although the future of the project in general appears to be bright, there seems to be little consensus about the likely direction and characteristics of the early Semantic Web.

What's the rationale for such a system? Data that is geneally hidden away in HTML files is often useful in some contexts, but not in others. The problem with the majority of data on the Web that is in this form at the moment is that it is difficult to use on a large scale, because there is no global system for publishing data in such a way as it can be easily processed by anyone. For example, just think of information about local sports events, weather information, plane times, Major League Baseball statistics, and television guides...

all of this information is presented by numerous sites, but all in HTML. The problem with that is that, is some contexts, it is difficult to use this data in the ways that one might want to do so.

So the Semantic Web can be seen as a huge engineering solution... but it is more than that. We will find that as it becomes easier to publish data in a repurposable form, so more people will want to pubish data, and there will be a knock-on or domino effect. We may find that a large number of Semantic Web applications can be used for a variety of different tasks, increasing the modularity of applications on the Web. But enough subjective reasoning... onto how this will be accomplished.

The Semantic Web is generally built on syntaxes which use URIs to represent data, usually in triples based structures: i.e. many triples of URI data that can be held in databases, or interchanged on the world Wide Web using a set of particular syntaxes developed especially for the task. These syntaxes are called "Resource Description Framework" syntaxes.

URI - Uniform Resource Identifier

A URI is simply a Web identifier: like the strings starting with "http:" or "ftp:" that you often find on the World Wide Web. Anyone can create a URI, and the ownership of them is clearly delegated, so they form an ideal base technology with which to build a global Web on top of. In fact, the World Wide Web is such a thing: anything that has a URI is considered to be "on the Web".

The syntax of URIs is carefully governed by the IETF, who published RFC 2396 as the general URI specification. The W3C maintains a list of URI schemes.

RDF - Resource Description Framework

A triple can simply be described as three URIs. A language which utilises three URIs in such a way is called RDF: the W3C have developed an XML serialization of RDF, the "Syntax" in the RDF Model and Syntax recommendation. RDF XML is considered to be the standard interchange format for RDF on the Semantic Web, although it is not the only format. For example, Notation3 (which we shall be going through later on in this article) is an excellent plain text alternative serialization.

Once information is in RDF form, it becomes easy to process it, since RDF is a generic format, which already has many parsers. XML RDF is quite a verbose specification, and it can take some getting used to (for example, to learn XML RDF properly, you need to understand a little about XML and namespaces beforehand...), but let's take a quick look at

an example of XML RDF right now:-

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:foaf="http://xmlns.com/0.1/foaf/" >
    <rdf:Description rdf:about="">
        <dc:creator rdf:parseType="Resource">
            <foaf:name>Sean B. Palmer</foaf:name>
        </dc:creator>
        <dc:title>The Semantic Web: An Introduction</dc:title>
    </rdf:Description>
</rdf:RDF>
```

This piece of RDF basically says that this article has the title "The Semantic Web: An Introduction", and was written by someone whose name is "Sean B. Palmer". Here are the triples that this RDF produces:-

```
<> <http://purl.org/dc/elements/1.1/creator> _:x0 .
this <http://purl.org/dc/elements/1.1/title> "The Semantic Web:
An Introduction" .
_:x0 <http://xmlns.com/0.1/foaf/name> "Sean B. Palmer" .
```

This format is actually a plain text serialization of RDF called "Notation3", which we shall be covering later on. Note that some people actually prefer using XML RDF to Notation3, but it is generally accepted that Notation3 is easier to use, and is of course convertable to XML RDF anyway.

Why RDF?

When people are confronted with XML RDF for the first time, they usually have two questions: "why use RDF rather than XML?", and "do we use XML Schema in conjunction with RDF?".

The answer to "why use RDF rather than XML?" is quite simple, and is twofold. Firstly, the benefit that one gets from drafting a language in RDF is that the information maps *directly* and *unambiguously* to a model, a model which is decentralized, and for which there are many generic parsers already available. This means that when you have an RDF application, you know which bits of data are the semantics of the application, and which bits are just syntactic fluff. And not only do you know that, *everyone* knows that, often implicitly without even reading a specification because RDF is so well known. The second part of the twofold answer is that we hope that RDF data will become a part of the Semantic Web, so the benefits of drafting your data in RDF now draws parallels with drafting your information

in HTML in the early days of the Web.

The answer to "do we use XML Schema in conjunction with RDF?" is almost as brief. XML Schema is a language for restricting the *syntax* of XML applications. RDF already has a built in BNF that sets out how the language is to be used, so on the face of it the answer is a solid "no". However, using XML Schema in conjunction with RDF *may* be useful for creating datatypes and so on. Therefore the answer is "possibly", with a caveat that it is not really used to control the syntax of RDF. This is a common misunderstanding, perpetuated for too long now.

Screen Scraping, and Forms

For the Semantic Web to reach its full potential, many people need to start publishing data as RDF. Where is this information going to come from? A lot of it can be derived from many data publications that exist today, using a process called "screen scraping". Screen scraping is the act of literally getting the data from a source into a more manageable form (i. e. RDF) using whatever means come to hand. Two useful tools for screen scraping are XSLT (an XML transformations language), and RegExps (in Perl, Python, and so on).

However, screen scraping is often a tedious solution, so another way to approach it is to build proper RDF systems that take input from the user and then store it straight away in RDF. Data such as you may enter when signing up for a new mail account, buying some CDs online, or searching for a used car can all be stored as RDF and then used on the Semantic Web.

Notation3: RDF Made Easy

As you will have seen above, XML RDF can be rather difficult, but thankfully, there is are simpler teaching forms of RDF. One of these is called "Notation3", and was developed by Tim Berners-Lee. There is some documentation covering N3, including a specification, and an excellent Primer.

The design criteria behind Notation3 were fairly simple: design a simple easy to learn scribblable RDF format, that is easy to parse and build larger applications on top of. In Notation3, we can simply write out the URIs in a triple, delimiting them with a "<" and ">" symbols. For example, here's a simple triple consisting of three URI triples:-

```
<http://xyz.org/#a> <http://xyz.org/#b> <http://xyz.org/#c> .
```

To use literal values, simply enclose the value in double quote marks, thus:-

```
<http://xyz.org/#Sean> <http://xyz.org/#name> "Sean" .
```

If you don't want to give a URI for something that you are talking about, then there is a concept for that too (this is like saying "there is someone called... but without giving them a URI). You simply use an underscore and a colon, and then put a little label there:-

```
_:a1 <http://xyz.org/#name> "Sean" .
```

This may be read as "there is something that has the name Sean", or "a1 has the name Sean, for some value of a1". These things are called anonymous nodes, because they don't have a URI, and are sometimes referred to as existentially quantified nodes.

Note how in one of the examples above, we used the URI "http://xyz.org/#" three times, with only the last character changing each time? Notation3 gives us an excellent way to abbreviate this: by giving parts of URIs aliases, and using those aliases instead. This is how you declare an alias in Notation3:-

```
@prefix xyz: <http://xyz.org/#> .
```

Note that you must always declare an alias before you can use it. To use an alias, simply use the "xyz:" bit instead of the URI, and *don't* wrap the resulting term in the "<" and ">" delimiters. For example, instead of writing:-

```
<http://xyz.org/#a> <http://xyz.org/#b> <http://xyz.org/#c> .
```

We can instead do:-

```
@prefix xyz: <http://xyz.org/#> .
:a :b :c .
```

Note that it doesn't matter what alias you use for a URI, as long as you use the same one throughout that document. You can also declare many aliases. The following bits of code are both equivalent to the piece of code above:-

```
@prefix blargh: <http://xyz.org/#> .
blargh:a blargh:b blargh:c .
```

```
@prefix blargh: <http://xyz.org/#> .
@prefix xyz: <http://xyz.org/#> .
blargh:a xyz:b blargh:c .
```

However, it should be noted that we often use a few aliases pretty much standardly, so that when Semantic Web developers exchange code in plain text, they can just leave the prefixes out and people can guess what they're talking about. Note that code should *not* implement this feature. Here is an example of some "standard" aliases:-

```
@prefix : <#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml: <http://www.daml.org/2001/03/daml+oil#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

The empty alias ":" is often used to denote a new namespace that the author has not yet created a URI for (tut, tut). We use it in this introduction.

Notation3 does have many other little constructs including contexts, DAML lists, and alternative ways of representing anonymous nodes, but we need not concern ourselves with them here. Note that a syntax was devised to be an even simpler subset of Notation3, called N-Triples, but it doesn't use prefixes, and hence many of the examples in this article are not valid N-Triples, but are valid Notation3.

Dan Connolly once called Notation3 a "poor-man's RDF authoring tool" (source: RDF IG logs, 2001-06-01 03:55:12). Apparently, it is called Notation3 because "RDF M&S was the first, the RDF strawman was the second and this is the third" (source: RDF IG F2F 2001-02-26).

CWM: An XML RDF And Notation3 Inference Engine

Although we won't be discussing inference engines until later on in this article, we should note at this point that much RDF and Semantic Web processing (albeit often only experimental or demonstrative, at this stage) is done using a Python program called CWM or "Closed World Machine". More information can be found on the SWAP site.

At the moment, the best demonstration of its use can be how it can convert from XML RDF into Notation3 and vice versa. To convert "a.n3" into "a.rdf", simply use the following command line:-

```
python cwm.py a.n3 -rdf > a.rdf
```

CWM is a very powerful Semantic Web toolkit, and we shall be refering to it occasionally

throught this article.

Simple Data Modelling: Schemata

The first "layer" of the Semantic Web above the syntax discussed above is the simple datatyping model. A "schema" (plural "schemata") is simply a document or piece of code that controls a set of terms in another document or piece of code. It's like a master checklist, or definition grammar.

RDF Schema

RDF Schema (also: RDF Schema Candidate Recommendation) was designed to be a simple datatyping model for RDF. Using RDF Schema, we can say that "Fido" is a type of "Dog", and that "Dog" is a sub class of animal. We can also create properties and classes, as well as doing some slightly more "advanced" stuff such as creating ranges and domains for properties.

All of the terms for RDF Schema start with "http://www.w3.org/2000/01/rdf-schema#", which you may have noticed is in our list of "standard" aliases above. The alias "rdfs:" is often used for RDF Schema, and we continue that tradition here.

The first three most important concepts that RDF and RDF Schema give us are the "Resource" (rdfs:Resource), the "Class" (rdfs:Class), and the "Property" (rdf:Property). These are all "classes", in that terms may belong to these classes. For example, all terms in RDF are types of resource. To declare that something is a "type" of something else, we just use the rdf:type property:-

```
rdfs:Resource rdf:type rdfs:Class .
rdfs:Class rdf:type rdfs:Class .
rdf:Property rdf:type rdfs:Class .
rdf:type rdf:type rdf:Property .
```

This simply says that "Resource is a type of Class, Class is a type of Class, Property is a type of Class, and type is a type of Property". These are all true statements.

It is quite easy to make up your own classes. For example, let's create a class called "Dog", which contains all of the dogs in the world:-

```
:Dog rdf:type rdfs:Class .
```

Now we can say that "Fido is a type of Dog":-

```
:Fido rdf:type :Dog .
```

We can also create properties quite easily by saying that a term is a type of rdf:Property, and then use those properties in our RDF:-

```
:name rdf:type rdf:Property .
:Fido :name "Fido" .
```

Why have we said that Fido's name is "Fido"? Because the term ":Fido" is a URI, and we could quite easily have chosen any URI for Fido, including ":Squiggle" or ":n508s0srh". We just happened to use the URI ":Fido" because it's easier to remember. However, we still have to tell machines that his name is Fido, because although people can guess that from the URI (even though they probably shouldn't), machines can't.

RDF Schema also has a few more properties that we can make use of: rdfs:subClassOf and rdfs:subPropertyOf. These allow us to say that one class or property is a sub class or sub property of another. For example, we might want to say that the class "Dog" is a sub class of the class "Animal". To do that, we simply say:-

```
:Dog rdfs:subClassOf :Animal .
```

Hence, when we say that Fido is a Dog, we are also saying that Fido is an Animal. We can also say that there are other sub classes of Animal:-

```
:Human rdfs:subClassOf :Animal .
:Duck rdfs:subClassOf :Animal .
```

And then create new instances of those classes:-

```
:Bob rdf:type :Human .
:Quakcy rdf:type :Duck .
```

And then we can invent another property, use that, and build up even more information...

```
:owns rdf:type rdf:Property .
:Bob :owns :Fido .
:Bob :owns :Quacky .
:Bob :name "Bob Fleming" .
:Quacky :name "Quakcy" .
```

And so on. You can see that RDF Schema is very simple, and yet allows one to build up

knowledge bases of data in RDF very very quickly.

The next concepts which RDF Schema provides us, which are important to mention, are ranges and domains. Ranges and domains let us say what classes the subject and object of each property must belong to. For example, we might want to say that the property ": bookTitle" must always apply to a book, and have a literal value:-

```
:Book rdf:type rdfs:Class .
:bookTitle rdf:type rdf:Property .
:bookTitle rdfs:domain :Book .
:bookTitle rdfs:range rdfs:Literal .
:MyBook rdf:type :Book .
:MyBook :bookTitle "My Book" .
```

rdfs:domain always says what class the subject of a triple using that property belongs to, and rdfs:range always says what class the object of a triple using that property belongs to.

RDF Schema also contains a set of properties for annotating schemata, providing comments, labels, and the like. The two properties for doing this are rdfs:label and rdfs: comment, and an example of their use is:-

```
:bookTitle rdfs:label "bookTitle";
   rdfs:comment "the title of a book" .
```

It is a good best practise to always label and comment your new properties, classes, and other terms.

Ontologies, Inferences, and DAML

DAML is a language created by DARPA as an ontology and inference langauge based upon RDF. DAML takes RDF Schema a step further, by giving us more in depth properties and classes. DAML allows one to be even more expressive than with RDF Schema, and brings us back on track with our Semantic Web discussion by providing some simple terms for creating inferences.

DAML+OIL

DAML provides us a method of saying things such as inverses, unambiguous properties, unique properties, lists, restrictions, cardinalities, pairwise disjoint lists, datatypes, and so on. We shall run through a couple of these here, but armed with the knowledge that you've already gotten from this introduction (assuming that you haven't skipped any of it!), it should

be just as beneficial going through the DAML+OIL Walkthru.

One DAML construct that we shall run through is the daml:inverseOf property. Using this property, we can say that one property is the inverse of another. The rdfs:range and rdfs: domain values of daml:inverseOf is rdf:Property. Here is an example of daml:inverseOf being used:-

```
:hasName daml:inverseOf :isNameOf .
:Sean :hasName "Sean" .
"Sean" :isNameOf :Sean .
```

The second useful DAML construct that we shall go through is the daml: UnambiguousProperty class. Saying that a Property is a daml:UnambiguousProperty means that if the object of the property is the same, then the subjects are equivalent. For example:-

```
foaf:mbox rdf:type daml:UnambiguousProperty .
:x foaf:mbox  .
:y foaf:mbox  .
```

implies that:-

```
:x daml:equivalentTo :y .
```

Don't worry if this is getting all a bit too much... it's not essential to learning about the Semantic Web, but it is useful, since many Semantic Web applications now involve DAML. However, DAML is only one in a series of languages and so forth that are being used.

Inference

The principle of "inference" is quite a simple one: being able to derive new data from data that you already know. In a mathematical sense, querying is a form of inference (being able to infer some search results from a mass of data, for example). Inference is one of the driving principles of the Semantic Web, because it will allow us to create SW applications quite easily.

To demonstrate the power of inference, we can use some simple examples. Let's take the simple car example: we can say that:-

```
:MyCar de:macht "160KW" .
```

Now, to a German Semantic Web processor, the term ":macht" may well be built into it, and although an English processor may have an equivalent term built into it somewhere, it will not understand the code with the term in it that it doesn't understand. Here, then, is a piece of inference data that makes things clearer to the processor:-

```
de:macht daml:equivalentTo en:power .
```

We have used the DAML "equivalentTo" property to say that "macht" in the German system is equivalent to "power" in the English system. Now, using an inference engine, a Semantic Web client could successfully determine that:-

```
:MyCar en:power "160KW" .
```

This is only a very simple example of inference, but you can see immediately how easily the system could scale up. Merging databases simply becomes a matter of recording in RDF somewhere that "Person Name" in your database is equivalent to "Name" in my database, and then throwing all of the information together and getting a processor to think about it.

Indeed, this is already possible with Semantic Web tools that we have at our disposal today: CWM. Unfortunately, great levels of inference can only be provided using "First Order Predicate Logic" languages, and DAML is not a FOPL language entirely.

Logic

For the Semantic Web to become expressive enough to help us in a wide range of situations, it will become necessary to construct a powerful logical language for making inferences. There is a raging debate as to how and even whether this can be accomplished, with people pointing out that RDF lacks the power to quantify, and that the scope of quantification is not well defined. Predicate logic is better discussed in John Sowa's excellent Mathematical Background (Predicate Logic).

In particular, Pat Hayes is hard at work on a model for RDF that may ease the situation (2001-09), but there is still a great amount of uncertainty at this time. Of course, this does not stop us from using a Webized version of KIF or somesuch as a logical language on the Semantic Web.

At any rate, we already have a great range of tools with which to build the Semantic Web: assertions (i.e. "and"), and quoting (reification) in RDF, classes, properties, ranges and documentation in RDF Schema, disjoint classes, unambiguous and unique properties, datatypes, inverses, equivalencies, lists, and much more in DAML+OIL.

Note that Notation3 introduces a "context" construct, enabling one to group statements together and quantify over them using a specially designed logic vocabulary. Using this vocabulary, for example, one can express "or", using NANDs:-

```
{ { :Joe :loves :TheSimpsons } a log:Falsehood .
  { :Joe :is :Nuts } a log:Falsehood .
} a log:Falsehood .
```

Which can be read as "it is not true that Joe does not love The Simpsons and is not nuts". I resisted the temptation to make Joe a universally quantified variable.

Note that the above example does not serialize "properly" into XML RDF, because XML RDF does not have the context construct as denoted by the curly brackets in the example above. However a similar effect can be achieved using reification and containers.

The Power Of Semantic Web Languages

The main power of Semantic Web languages is that any one can create one, simply by publishing some RDF that describes a set of URIs, what they do, and how they should be used. We have already seen that RDF Schema and DAML are very powerful langauges for creating languages.

Because we use URIs for each of the terms in our languages, we can publish the languages easily without fear that they might get misinterpreted or stolen, and with the knowledge that anyone in the world that has a generic RDF processor can use them.

The Principle Of Least Power

The Semantic Web works on a principle of least power: the less rules, the better. This means that the Semantic Web is essentially very unconstraining in what it lets one say, and hence it follows that anyone can say anything about anything. When you look at what the Semantic Web is trying to do, it becomes very obvious why this level of power is necessary... if we started constraining people, they wouldn't be able to build a full range of applications, and the Semantic Web would therefore become useless to some people.

How Much Is Too Much?

However, it has been pointed out that this power will surely be too much... won't people be trying to process their shopping lists on an inference engine, and suddenly come up with a plan for world peace, or some strange and exciting new symphony?

The answer is (perhaps unfortunately!) no. Although the basic parts of the Semantic Web, RDF and the concepts behind it are very minimally constraining, applications that are built on top of the Semantic Web will be designed to perform specific tasks, and as such will be very well defined.

For example, take a simple server log program. One might want to record some server logs in RDF, and then build a program that can gather statistics from the logs that pertain to the site; how many visitors it had in a week, and so forth. That doesn't mean that it'll turn your floppy disc drive into a toaster or anything; it'll just process server logs. The power that you get from publishing your information in RDF is that once published in the public domain, it can be *repurposed* (used for other things) so much easier. Because RDF uses URIs, it is fully decentralized: you don't have to beg for some central authority to publish a language and all your data for you... you can do it yourself. It's Do It Yourself data management.

The Pedantic Web

Unfortunately, there is an air of academia and corporatate thinking lingering in the Semantic Web community, which has lead to the term "Pedantic Web" being coined, and a lot of mis/disinformation and unecessary hype being disseminated. Note that this very document was devised to help clear up some common misconceptions that people may have about the Semantic Web.

For example, almost all beginners to RDF go through a sort of "identity crisis" phase, where they confuse people with their names, and documents with their titles. For example, it is common to see statements such as:-

```
<http://example.org/> dc:creator "Bob" .
```

However, Bob is just a literal string, so how can a literal string write a document? What the author really means is:-

```
<http://example.org/> dc:creator _:b .
_:b foaf:name "Bob" .
```

i.e., that example.org was created by someone whose name is "Bob". Tips like these are being slowly collected, and some of them are being displayed in the SWTips guide, a collection of Semantic Web hints and tips maintained as a collaborative development project.

Education And Outreach

The move away from the "Pedantic Web", to some extent, is all part of a movement to bring the power of the Semantic Web to the people. This is a well documented need:-

> [...] the idea that the above URIs reveal a schema that somehow fully describes this language and that it is so simple (only two {count 'em 2} possible "statements"), yet looks like the recipe for flying to Mars is a bit daunting. Its very simplicity enables it to evaluate and report on just about anything - from document through language via guidelines! It is a fundamental tool for the Semantic Web in that it gives "power to the people" who can say anything about anything.
>
> - *EARL for dummies, William Loughborough, May 2001*

RDF Schema and DAML+OIL are generally languages that need to be *learned*, however, so what is being done to accomodate people who have neither the time nor patience to read up on these things, and yet want to create Semantic Web applications? Thankfully, many Semantic Web applications will be lower end appliactions, so you'll no more need to have a knowledge of RDF than Amaya requires one to have a knowledge of (X)HTML.

Trust and Proof

The next step in the archtecture of the Semantic Web is trust and proof. Very little is written about this layer, which is a shame since it will become very important in the future.

In stark reality, the simplest way to put it is: if one person says that x is blue, and another says that x is not blue, doesn't the whole Semantic Web fall apart?

The answer is of course not, because a) applications on the Semantic Web at the moment generally depend upon context, and b) because applications in the future will generally contain proof checking mechanisms, and digital signatures.

Context

Applications on the Semantic Web will depend on context generally to let people know whether or not they trust the data. If I get an RDF feed from a friend about some movies that he's seen, and how highly he rates them, I know that I trust that information. Moreover, I can then use that information and safely trust that it came from him, and then leave it down to my own judgement just to how much I trust his critiques of the films that he has reviewed.

Groups of people also operate on shared context. If one group is developing a Semantic

Web depiction service, cataloguing who people are, what their names are, and where pictures of those people are, then my trust of that group is dependant upon how much I trust the people running it not to make spurious claims.

So context is a good thing because it lets us operate on local and medium scales intuitively, without having to rely on complex authentication and checking systems. However, what happens when there is a party that we know, but we don't know how to verify that a certain heap of RDF data came from them? That's where digital signatures come in.

Digital Signatures

Digital signatures are simply little bits of code that one can use to unambiguously verify that one wrote a certain document. Many people are probably familiar with the technology: it the same key based PGP-style thing that people use to encrypt and sign messages. We simply apply that technology to RDF.

For example, let's say I have some information in RDF that contains a link to a digital signature:-

```
this :signature <http://example.org/signature> .
:Jane :loves :Mary .
```

To ascertain whether or not we trust that Jane really loves Mary, we can feed the RDF into a trust engine (an inference engine that has a little digital signature checker built into it), and get it to work out if we trust the source of the information.

Proof Languages

A proof language is simply a language that let's us prove whether or not a statement is true. An instance of a proof language will generally consist of a list of inference "items" that have been used to derive the information in question, and the trust information for each of those items that can then be checked.

For example, we may want to prove that Joe loves Mary. The way that we came across the information is that we found two documents on a trusted site, one of which said that ":Joe : loves :MJS", and another of which said that ":MJS daml:equivalentTo :Mary". We also got the checksums of the files in person from the maintainer of the site.

To check this information, we can list the checksums in a local file, and then set up some FOPL rules that say "if file 'a' contains the information Joe loves mary and has the checksum md5:0qrhf8q3hfh, then record SuccessA", "if file 'b' contains the information MJS is equivalent to Mary, and has the checksum md5:0892t925h, then record SuccessB", and

"if SuccessA and SuccessB, then Joe loves Mary".

An example of this in Notation3 can be found in some of the author's proof example experiments, but here is the rules file:-

```
@prefix : <http://infomesh.net/2001/proofexample/#> .
@prefix p: <http://www.w3.org/2001/07/imaginary-proof-
ontology#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

p:ProvenTruth rdfs:subClassOf log:Truth .

# Proof

{ { <a.n3> p:checksum <md5:blargh>;
           log:resolvesTo [ log:includes { :Joe :loves :MJS } ] }
log:implies
{ :Step1 a p:Success } } a log:Truth .

{ { <b.n3> p:checksum <md5:test>;
          log:resolvesTo [ log:includes { :MJS = :Mary } ] }
log:implies
{ :Step2 a p:Success } } a log:Truth .

{ { :Step1 a p:Success . :Step2 a p:Success }
log:implies
{ { :Joe :loves :Mary } a p:ProvenTruth } } a log:Truth .
```

The file speaks for itself, and when processed using CWM, does indeed work, producing the intended output. CWM doesn't have the capability to automatically check file checksums or digital signatures, but it is only a matter of time before a proper Semantic Web trust engine is written.

Ambient Information and SEM

The scope of information was discussed a little, but let's take into consideration what it really means to have a "local" and a "global" system.

In general, there are small and large scale systems, and interactions between the two will most likely form a huge part of the transactions that occur on the Semantic Web. Let's define what we mean by large, medium, and small scale systems.

Large Scale

An example of a large scale system is two companies that are undergoing a merger needing to combine their databases. Another example would be search engines compiling results based upon a huge range of data. Large scale Semantic Web systems generally involve large databases, and heavy duty inference rules and processors are required to handle the databases.

Medium Scale

Medium scale Semantic Web systems attempt to make sense out of the larger scale Semantic Web systems, or are examples of small scale Semantic Web systems joined together. An example of the former is a company trying to partially understand two large scale invoice formats enough to use them together. An example of the latter is of two address book language groups trying to create a super-address book language.

Small Scale

Small scale Semantic Web systems are less widely discussed. By small scale Semantic Web systems, we mean languages that will be used primarily offline, or piles of data that will only be transferred with a limited scope, perhaps between friends, departments, or even two companies.

Sharing data on a local level is a very powerful example of how the Semantic Web can be useful in a myriad of situations. In the next section on evolution we shall be finding out how interactions between the different sized systems will form a key part of the Semantic Web.

SEM - SEmantic Memory

The concept of a SEmantic Memory was first proposed by Seth Russell, who suggested that personal database dumps of RDF that one has collected from the "rest" of the Semantic Web (a kind of Semantic Cloud) would be imperative for maintaining a coherant view of data. For example, a SEM would most likely be partitioned into data which is inherent to the whole Semantic Web (i.e., the schemata for the major languages such as XML RDF, RDF Schema, DAML+OIL, and so on), local data which is important for any Semantic Web applications that may be running (e.g. information about the logic namespace for CWM, which is currently built in), and data that the person has personally been using, is publishing, or that has been otherwise entered into the root context of the SEM.

The internal structure of a SEM will most likely go well beyond the usual triples structure of RDF, perhaps as far as quads or even pents. The extra fields are for contexts (an StID), and perhaps sequences. In other words, they are ways of grouping information *within* the SEM, for easy maintainence and update. For example, it should become simple to just delete any triple that was added into a certain context by removing all triples with that particular StID.

A lot of work on the Semantic Web has concentrated on making data stores (i.e. SEMs) interoperable, which is good, but that has lead to less work being conducted on what actually happens within the SEM itself, which is not good, because the representation of quads and pents in RDF is therefore up in the air. Obviously, statements can be modelled as they would be for reification:-

```
rdf:Statement rdfs:subClassOf :Pent .
_:s1 rdf:type :Pent .
_:s1 rdf:subject :x .
_:s1 rdf:predicate :y .
_:s1 rdf:object :z .
_:s1 :context :p .
_:s1 :seq "0" .
```

But clearly a dedicated pentuples format is always going to be more efficient, and aviod the perils of reification:-

```
:x :y :z :p "0" .
```

This language also needs a default context flag that indicates the root context of the document. The root context of a document is the space to which the (non-quoted) assertions are parsed, the conceptual information space in which all of the assertions are taken to be true. Any quoted information in the document (for example, using the Notation3 context syntax) would be in a different (possibly anonymous) context than the root context of the document.

TimBL appears, gaguing from the CWM source code, to be using "...#_formula" as the root context for the document, which (if true) is a bit of a nasty hack... what if one creates a property with the same URI? Maintaining interoperability at this level of the Semantic Web is an important thing for the Semantic Web developers to be investigating at this stage.

Evolution

A very important concept on the Semantic Web is that of evolution: going from one system into another. Two key parts of evolvability are *partial understanding* and *transformability*.

We will find out next how these manifest themselves naturally when changing the scale of a system.

Partial Understanding: Large Scale to Medium Scale

The concept of *partial understanding* is a very important one on the Semantic Web, and can often be found in older documents that came out about the same time as the Semantic Web was first being theorized.

An example of partial understanding when moving a large scale system to a medium scale system is of a company trying to make sense out of two invoices, one from Company A and one from Company B. The knowledge that both of the companies use similar fields in their invoices is well known, so company trying to make sense out of the invoices can easily compile a master list of expenditures by simply scraping the data from the two invoice languages. Neither Company A nor Company B need to know that this is going on.

Indeed, TimBL included this example in his XML 2000 keynote:-

> [...] what we'll end up doing in the future is converting things, so for example [...] in the Semantic Web we will have a relationship between two langauges so that if you get an invoice in a langauge you don't understand, and you have... some business software which can pay invoices... by following links across the Semantic Web, your machine will be able to automatically convert it from one language to another, and so process it.
> - *Tim Berners-Lee*

Transformability: Small Scale to Medium Scale

An example of a small scale Semantic Web system joined together to make a medium sized Semantic Web system could be two groups that have published address book fomats wanting to make a larger and better address book format by merging the two current formats together. Anyone using one of the old address book formats could probably convert them into the new format, and hence there would be a greater sense of interoperability. That's generally what happens when one goes from a small scale Semantic Web system into a medium scale Semantic Web system, although this is often not without some disadvantages and incompatabilites. The Semantic Web takes the sting out of it by automating 99% of the process (it can convert field A into field B, but it can't fill in any new data for you... of course, new fields can always be left empty for a while).

Facilitating Evolvability

How do we document the evolution of languages? This is a very important and indeed urgent question, and one which TimBL summarized quite neatly:-

> Where for example a library of congress schema talks of an "author", and a British Library talks of a "creator", a small bit of RDF would be able to say that for any person x and any resource y, if x is the (LoC) author of y, then x is the (BL) creator of y. This is the sort of rule which solves the evolvability problems. Where would a processor find it? [...]
>
> - *Semantic Web roadmap, Tim Berners-Lee*

One possible answer is: third party databases. Very often, it is not practical to have (in TimBL's example) either the LoC or or BL record the fact that two of their fields are the same, so this information will have to be recorded by a reputable third party.

One such "third party" that was set up to investigate this is SWAG, the Semantic Web Agreement Group. Co-founded by Seth Russell, Sean B. Palmer, Aaron Swartz, and William Loughborough, the group aims to ensure interoperability on the Semantic Web. They set up what is possibly the first ever third party Semantic Web dictionary, the WebNS SWAG Dictionary.

Intertwingling: Difficult, But Important

Although the Semantic Web as a whole is still very much at a grassroots kind of level, people are starting to take notice; they're starting to publish information using RDF, and thereby making it fit for the Semantic Web.

However, not enough is being done to link information together... in other words, the "Semantic" part of the "Semantic Web" is coming along nicely, but where's the "Web"? People are not using other people's terms effectively; when they use other terms, they often do so because they're aimlessly trying to help, but just generating noice in the process. If you're going to use other people's data, try to find out what the advantage is in doing that beforehand. For example, just because you use the term "dc:title" in your RDF rather than a home brewed ":title", does that mean that suddenly a Dublin Core application is going to be able to "understand" your code? Of course not. What it does mean however is that if the "dc:title" property in your instance is being put to use in such a way that information may be need to repurposed from it in the near future, then you *may* gain some advantage because "dc:title" is such a commonly used term, you may be able to modify a current rules file, or whatever.

Another part of the problem may be due to a problem similar to the one that the early World

Wide Web experienced: why bother publishing a Web site when there is no one else's site to link to or be linked to? Why bother publishing a Web site when so few people have browsers? Why bother writing a browser when there are so few Web sites? Some people have to make the leaps for it all to happen, and that's a slow process.

What can be done about the situation? Well, it may hopefully sort itself out. Another well-known principle that applies very well to Semantic Web applications is that there is no point in reinventing the wheel; viz., if someone has already invented a schema which contains a comprehensive and well understood and used set of terms that you also need to use in your application, then there is no point in trying to redo the work that they have done. At some points this may lead to a form of "schema war", but survival of the fittest should see to it that a core of the best schemata are put to the most use. This is probably what TimBL means when he says that terms will just "emerge" out of the Semantic Cloud, that when people keep using the term "zip", rather than just recording that my term "zip" is equivalent to your term "zip" which is equivalent to someone else's term "zip", we'll all just use the same URI, and hence interoperability will be vastly improved.

Does It Work? What Semantic Web Applications Are There?

I addressed this in a previous article: The Semantic Web: Taking Form, but it does bear repeating: the Semantic Web *already* works, and people are using it.

Semantic Web Applications

Unfortunately, the Semantic Web is dissimilar in many ways from the World Wide Web, including that you can't just point people to a Web site for them to realise how it's working, and what it is. However, there have been a number of small scale Semantic Web applications written up. One of the best ones is Dan Connolly's Arcs and Nodes diagrams experiment:-

> One of the objectives of the advanced development component of the Semantic Web activity is to demonstrate how RDF and Semantic Web technologies can be applied to the W3C Process to increase efficiency, reliability, etc. In the early stages of developing an RDF model of the W3C process, the tools I was using to visualize the model while working on it started working well enough that I started applying them to all sorts of stuff.
>
> - *Circles and arrows diagrams using stylesheet rules, Dan Connolly.*

Of course, this is a rather demonstration-oriented Semantic Web project, but it does illustrate the feasibility of applications being easily built using Semantic Web toolkits.

Another good example of the Semantic Web at work is Dan Brickley et al.'s RDFWeb. RDFWeb is a RDF database driven hypermedia blogspace, a site where all information is stored as RDF, and then that RDF used to render XHTML. Plans are underway to incorporate more advanced Semantic Web principles into the site.

What Can I Do To Help?

There are many ways in which one can contribute to creating the Semantic Web. Here's a few of them:-

- Publish some globally useful data in RDF.
- Write an inference engine in the language of your choice.
- Spread the word: do some education and outreach.
- Help in the developent of RDF Schema and/or DAML.
- Contribute in representing state in RDF, a rather neglected field of research.
- Apply your own development backgrounds to the Semantic Web, give us all a new angle to consider it from.
- Instead of using some proprietary system for your next application, consider making it a Semantic Web project instead.

There are many other ways in which one can help as well: ask in the community for more details.

What Now? Further Reading

As of 2001-09, the amount of Semantic Web Education and Outreach materials can only really be described as "pitiful" (hence this introduction, for a start). Here's a short list of some of the *good* primers and materials currently available, in no particular order:-

- http://www.w3.org/2000/10/swap/Primer (Getting Into RDF & Semantic Web Using N3)
- http://www.w3.org/DesignIssues/Semantic (Semantic Web Roadmap)
- http://purl.org/swag/whatIsSW (What Is The Semantic Web?)
- http://uwimp.com/eo.htm (Semantic Web Primer)
- http://logicerror.com/semanticWeb-long (Semantic Web Introduction - Long)
- http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html (SciAm: The Semantic Web)
- http://www.xml.com/pub/a/2001/03/07/buildingsw.html (Building The Semantic Web)
- http://infomesh.net/2001/06/swform/ (The Semantic Web, Taking Form)
- http://www.w3.org/2001/sw/Activity (SW Activity Statement)
- http://www.w3.org/2000/01/sw/ (SWAD)

For more information, all the latest news etc., Dave Beckett's Resource Description Framework (RDF) Resource Guide is absolutely brilliant.

Many Semantic Web and RDF developers hang out on the RDF IG IRC chatroom, on irc.openprojects.net, #rdfig.

Acknowledgements

Many thanks to Aaron Swartz, Dan Brickley, and Danny Ayers for reviewing this, and pointing out many important little details that I missed.

Sean B. Palmer, 2001-09