The Semantic Web In Breadth

by Aaron Swartz (with much assistance)

Note: This piece speaks about the different parts of the Semantic Web and how they fit together. For a high-level interview, take a look at Sandro Hawke's The Semantic Web (Put Simply). On the other hand, if you're a Web developer who's interested in building Semantic Web Sites or Semantic Web Services, check out The Semantic Web (for Web Developers). Now back to your regularly scheduled article.

Identifiers: Uniform Resource Identifier (URI)

If I want to discuss something, I must first identify it. How else will you know what I'm referring to? I might do this in an indirect manner: "The North Star." "The strange man at the grocery store." "Those really sour candies that Bob always eats." I might also choose to be more direct: "Polaris." "Jonathan Roberts." "Mega Warheads."

To identify items on the Web, we also use identifiers. Because we use a uniform system of identifiers, and because each item identified is considered a "resource," we call these identifiers "Uniform Resource Identifiers" or URIs for short. We can give a URI to anything, and anything that has a URI can be said to be "on the Web": you, the book you bought last week, the fly that keeps buzzing in your ear and anything else you can think of -- they all can have a URI.

The URI is the foundation of the Web. While nearly every other part of the Web can be replaced, the URI cannot: it holds the rest of the Web together.

You're probably already familiar with one form of URI: the URL or Uniform Resource *Locator*. A URL is an address that lets you visit a webpage, such as: `http://www.w3.org/Addressing/`. If you break it down, you can see that a URL tells your computer where to find a specific resource (in this case, the W3C's Addressing website). Unlike most other forms of URIs, a URL both identifies and locates. Contrast this with a "mid:" URI. A "mid:" URI identifies an email message, but it isn't able to locate a copy of the message for you.

Because the Web is far too large for any one organization to control it, URIs are decentralized. No one person or organization controls who makes them or how they can be used. While some URI schemes (such as `http:`) depend on centralized systems (such as DNS), other schemes (such as `freenet:`) are completely decentralized.

This means that you don't need anyone's permission to create a URI. You can even create URIs for things you don't own. While this flexibility makes URIs powerful, it brings with it more than a few problems. Because anyone can create a URI, we will inevitably end up with multiple URIs representing the same thing. Worse, there will be no way to figure out whether two URIs refer to exactly the same resource. Thus, we'll never be able to say with certainty exactly what a given URI means. But these are tradeoffs that must be made if we are to create something as enormous as the Semantic Web.

A common practice for creating URIs is to begin with a Web page. The page describes the object to be identified and explains that the URL of the page is the URI for that object. For example, I wanted to create a URI for my copy of "Weaving the Web" by Tim Berners-Lee. First, I created a Web page that describes my copy. Second, I noted on that page that the URL for the page serves as the URI for my copy of the book. By doing this, I've associated that URI (http://logicerror.com/myWeavingTheWeb) with my copy of "Weaving the Web." Creating a URI can be just this simple.

You may have noticed that in this instance the URI (http://logicerror.com/myWeavingTheWeb) is doing double duty: it represents both the physical book, as well as the Web page that describes it. This is an area of some discussion -- it's called The Semantic Web Identification Problem, and it's a recurring point of discussion for Semantic Web workers.

This is an important fact to understand. A URI is not a set of directions telling your computer how to get to a specific file on the Web (though it may also do this). It is a name for a "resource" (a thing). This resource may **or may not be** accessible over the Internet. The URI may **or may not** provide a way for your computer to get more information about that resource. Yes, a URL is a type of URI that does provide a way to get information about a resource, or perhaps to retrieve the resource itself, and other methods for providing information about URIs and the resources they identify are under development. It is also true that the ability to say things about URIs is an important part of the Semantic Web. But you should not assume that a URI does anything more than provide an identifier for a resource.

Documents: Extensible Markup Language (XML)

XML was designed to be a simple way to send documents across the Web. It allows anyone to design their own document format and then write a document in that format. These document formats can include markup to enhance the meaning of the document's content. This markup is "machine-readable," that is, programs can read and understand it. By including machine-readable meaning in our documents, we make them much more powerful.

Consider a simple example: if a document contains certain words that are marked as "emphasized," the way those words

are rendered can be adapted to the context. A Web browser might simply display them in italics, whereas a voice browser (which reads Web pages aloud) might indicate the emphasis by changing the tone or the volume of its voice. Each program can respond appropriately to the meaning encoded in the markup. In contrast, if I simply marked the words as "in italics", the computer has no way of knowing *why* those words are in italics. Is it for emphasis or simply for a visual effect? How does the voice browser display this effect?

Here's an example of a document in plain text:

```
I just got a new pet dog.
```

As far as your computer is concerned, this is just text. It has no particular meaning to the computer. But now consider this same passage marked up using an XML-based markup language (we'll make one up for this example):

```
<sentence>
<person href="http://aaronsw.com/">I</person> just got a new pet <animal>dog</animal>.
</sentence>
```

Notice that this has the same content, but that parts of that content are labeled. Each label consists of two "tags": an opening tag (e.g., <sentence>) and a closing tag (e.g., </sentence>). The name of the tag ("sentence") is the label for the content enclosed by the tags. We call this collection of tags and content an "element." Thus, the sentence element in the above document contains the sentence, "I just got a new pet dog." This tells the computer that "I just got a new pet dog" is a "sentence," but -- importantly -- it does not tell the computer what a sentence is. Still, the computer now has some information about the document, and we can put this information to use.

Similarly, the computer now knows that "I" is a "person" (whatever that is) and that "dog" is an "animal."

Sometimes it is useful to provide more information about the content of an element than we can provide with the name of the element alone. For example, the computer knows that "I" in the above sentence represents a "person," but it does not know which person. We can provide this sort of information by adding *attributes* to our elements. An attribute has both a name and a value. For example, we can rewrite our example thus:

```
<sentence>
<person href="http://aaronsw.com">I</person> just got a new pet <animal type="dog"
href="http://aaronsw.com/myDog">dog</animal>.
</sentence>
```

As you might have guessed already, there is a problem here. I've used the words "sentence," "person," and "animal" in my markup language. But these are pretty common words. What if others have used these same words in their own markup languages? What if those words have different meanings in those languages? Perhaps "sentence" in another markup language refers to the amount of time that a convicted criminal must serve in a penal institution. How is my computer to keep these straight?

To prevent confusion, I must uniquely **identify** my markup elements. And what better way to identify them than with a Uniform Resource Identifier? So I assign a URI to each of my elements and attributes. I do this using something called XML Namespaces. This way, anyone can create their own tags and mix them with tags made by others. A namespace is just a way of identifying a part of the Web (space) from which we derive the meaning of these names. I create a "namespace" for my markup language by creating a URI for it. (As we discussed earlier, I'll probably create a Web page to describe my markup language and use the URL of my Web page as the URI for my namespace.)

Since everyone's tags have their own URIs, we don't have to worry about tag names conflicting. XML, of course, lets us abbreviate and set default URIs so we don't have to type them out each time:

```
<sentence
   xmlns="http://example.org/xml/documents/"
   xmlns:c="http://animals.example.net/xmlns/"
><c:person c:href="http://aaronsw.com/">I</c:person> just got a new pet <c:animal>dog</c:animal>.</sentence>
```

Note that "http://example.org/xml/documents/" is the default namespace for my document. That is where all elements and attributes not preceded by c: are defined (in this instance "sentence" is the only element thus defined).

Statements: Resource Description Framework (RDF)

Now we start to get into the meat of the Semantic Web. It's wonderful that we can create URIs and talk about them with our web pages. However, it'd be even better if we could talk about them in a way that computers could begin to process what we're saying. For example, it's one thing to say "I really like 'Weaving the Web.'" on a web discussion forum. But what would this mean to a computer?

RDF gives you a way to make statements that are **machine-processable**. Now the computer can't actually "understand" what you said, of course, but it can deal with it in a way that makes it seem like it does. For example, I could search the Web for all book reviews and create an average rating for each book. Then, I could put that information back on the Web. Another website could take that information (the list of book rating averages) and create a "Top Ten Highest Rated Books" page.

RDF is really quite simple. An RDF statement is a lot like a simple sentence, except that almost all the words are URIs. Each RDF statement has three parts: a subject, a predicate and an object. Let's look at a simple RDF statement:

<http://aaronsw.com/> <http://love.example.org/terms/reallyLikes> <http://www.w3.org/People/Berners-Lee/Weaving/> .

Can you guess what this says? The first URI is the subject. In this instance, the subject is me. The second URI is the predicate. It relates the subject to the object. In this instance, the predicate is "reallyLikes." The third URI is the object. Here, the object is Tim Berners-Lee's book "Weaving the Web." So the RDF statement above says that I really like "Weaving the Web."

You may notice that RDF statements can say practically anything, and that it doesn't matter who says them. There is no one official website that says everything about Weaving the Web, or about me. This leads us to an important RDF principle, namely "anything can say anything about anything". Information is spread across the Web, and two people can even say contradictory things -- Bob can say that Aaron loves Weaving the Web and John can say that Aaron hates it. This is the freedom that the Web provides.

The statement above is written in N-Triples, a language that allows you to write simple RDF statements. However, the official RDF specification defines an XML representation of RDF, which is a bit more complicated, but says the same thing:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:love="http://love.example.org/terms/"
>
    <rdf:Description rdf:about="http://aaronsw.com/">
        <love:reallyLikes rdf:resource="http://www.w3.org/People/Berners-Lee/Weaving/"
    </rdf:Description>
</rdf:RDF>
```

Now, to write RDF like that is not the easiest thing in the world, and it seems unlikely that everyone will start speaking this strange new language anytime soon. So where do we expect all this RDF information to come from? The most likely source is databases.

In the world there are thousands of databases, most containing interesting machine-processable information. Governments store arrest records in databases; companies store part and inventory information in a database; most computerized address books store people's names and phone numbers in -- you guessed it! -- a database. When information is stored in a database, it's very easy to ask the computer certain questions about the data: "Show me everyone that was arrested in the past 6 months." "Print a list of all parts we're running low on." "Get me the phone numbers of the people whose last name is Jones."

RDF is ideally suited for publishing these databases to the Web. And when we put them on the Web, we give everything in the database a URI, so that other people can talk about it too. Now, intelligent programs can begin to fit the data together. Using the available information, the computer can begin to connect the Bob Jones whose phone number is in your address book with the Bob Jones who was arrested last week and the Bob Jones who just ordered 100,000 widgets. Now, we can ask questions of all these databases at once: "Get me the phone number of everyone who ordered more than 1,000 widgets and was arrested in the last 6 months."

Schemas and Ontologies: RDF Schemas, DAML+OIL, and WebOnt

All the work on databases assumes that the data is nearly perfect. Few (if any) database systems are ready for the messiness of the Web. Any system that is "hard-coded" to understand certain terms will likely go out of date, or at least have limited usefulness, as new terms are invented and defined. What if someone comes up with a new system that rates books on a scale of 1-10 instead of just saying that someone "reallyLikes" them. Programs built based on the old system won't be able to process the new information.

Worse, there's no way for a computer or human to figure out what a specific term means, or how it should be used. The use of all these URIs is useless if we never describe what they mean. This is where schemas and ontologies come in. A schema and an ontology are ways to describe the meaning and realtionships of terms. This description (in RDF, of course) helps computer systems use terms more easily, and decide how to convert between them.

Two closely related systems, RDF Schemas and the DARPA Agent Markup Language with Ontology Inference Layer

(DAML+OIL) have been developed to solve this problem. For example, a schema might state that:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# A creator is a type of contributor:
dc:creator rdfs:subClassOf dc:contributor .
```

(Here we're using Notation3, as superset of N-Triples that allows us to use more abbreviations.)

This says that a creator is a subclass (type) of contributor. Of what use is this? Well, let's say for example that you build a program to collect the creators and contributors to various documents. Your program uses this vocabulary (dc:creator and dc:contributor) to understand the information it finds. One day, a vast influx of newbies from AOL start creating RDF documents. None of them know about dc:creator, so they make up their own term: ed:hasAuthor.

```
# The old way:
<http://aaronsw.com/> is dc:creator of <http://logicerror.com/semanticWeb-long> .

# The new way:
<http://logicerror.com/semanticWeb-long> ed:hasAuthor <http://aaronsw.com/> .
```

Normally, your program would simply ignore these new statements, since it can't understand them. However, one kind soul was smart enough to bridge the gap between these two worlds, by providing information on how to convert between them:

```
# [X dc:creator Y] is the same as [Y ed:hasAuthor X]
dc:creator daml:inverse ed:hasAuthor .
```

This tells your program that ed:hasAuthor is the inverse of dc:creator. That means that all your program has to do is swap the subject and object and change ed:hasAuthor to dc:creator. Since your program understands DAML ontologies, it can now take this information and use it to process all of the hasAuthor statements it couldn't understand before.

As of 2001-09-03, the W3C is preparing to start a Web Ontology (WebOnt) Working Group (which some call the WOW-G). This group is chartered to prepare a Web Ontology language that builts upon the work done by RDF Schema and DAML+OIL. It will be interesting to see what this group develops.

Logic

From this point on, I'll be discussing parts of the Semantic Web that haven't been developed yet. Unlike the discussion above, I'm not discussing specific systems, but instead a general concept that could become (and are becoming) many different systems.

While it's nice to have systems that understand these basic concepts (subclass, inverse, etc.) it would be even better if we could state **any** logical principle and permit the computer to reason (by inference) using these principles.

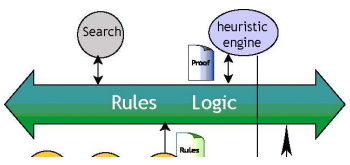
Here's an example: Let's say one company decides that if someone sells more than 100 of our products, then they are a member of the Super Salesman club. A smart program can now follow this rule to make a simple deduction: "John has sold 102 things, therefore John is a member of the Super Salesman club."

Proof

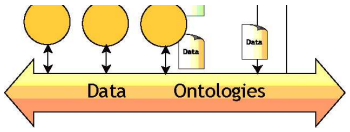
Once we begin to build systems that follow logic, it makes sense to use them to prove things. People all around the world could write logic statements. Then your machine could follow these Semantic "links" to construct proofs.

Example: Corporate sales records show that Jane has sold 55 widgets and 66 sprockets. The inventory system states that widgets and sprockets are both different company products. The built-in math rules state that 55 + 66 = 121 and that 121 is more than 100. And, as we know, someone who sells more than 100 products is a member of the Super Salesman club. The computer puts all these logical rules together into a proof that Jane is a Super Salesman.

While it's very difficult to create these proofs (it can require following thousands, or perhaps millions of the links in the Semantic Web), it's very easy to check them. In this way, we begin to build a Web of information processors. Some of them merely provide data for others to use. Others are smarter, and can use this data to build rules. The smartest are "heuristic engines" which follow all these rules and statements to draw



conclusions, and kindly place their results back on the Web as proofs, as well as plain old data.



Trust: Digital Signatures and Web of Trust

Now you've probably been thinking that this whole plan is great, but rather useless if anyone can say anything. Who would trust such as system? So you don't let me into your site? Ok, I just say I'm the King of the World and that I have permission. Because of the "anything can say anything about anything" rule, who's to stop me?

That's where Digital Signatures come in. Based on work in mathematics and cryptography, digital signatures provide proof that a certain person wrote (or agrees with) a document or statement. Aha! So I digitally sign all of my RDF statements. That way, you can be sure that I wrote them (or at least vouch for their authenticity). Now, you simply tell your program whose signatures to trust and whose not to. Each can set their own levels or trust (or paranoia) the computer can decide how much of what it reads to believe.

Now it's highly unlikely that you'll trust enough people to make use of most of the things on the Web. That's where the "Web of Trust" comes in. You tell your computer that you trust your best friend, Robert. Robert happens to be a rather popular guy on the Net, and trusts quite a number of people. And of course, all the people he trusts, trust another set of people. Each of those people trust another set of people, and so on. As these trust relationships fan out from you, they form a "Web of Trust." And each of these relationships has a degree of trust (or distrust) associated with it.

Note that distrust can be as useful as trust. Suppose that your computer discovers a document that no one explicitly trusts, but that no one explicitly distrusts either. Most likely, your computer will trust this document more than it trusts one that has been explicitly labeled as untrustworthy.

The computer takes all these factors into account when deciding how trustworthy a piece of information is. It can also make this process as transparent or opaque as you desire. For example, you might be happy with a simple "thumbs up/thumbs down" display. Someone else might insist on a complex explanation, including a description of some or all of the trust factors involved in the decision.

Tim Berners-Lee has proposed an "Oh, yeah?" button, that when clicked would have your computer attempt to provide reasons to trust the data. But whether you decide for yourself or leave it up to your computer, the information necessary to make an informed decision is available to you via the Web of Trust.

Conclusion: The Grand Vision

One of the best things about the Web is that it's so many different things to so many different people. The coming Semantic Web will multiply this versatility a thousandfold. For some, the defining feature of the Semantic Web will be the ease with which your PDA, your laptop, your desktop, your server, and your car will communicate with each other. For others, it will be the automation of corporate decisions that previously had to be laboriously hand-processed. For still others, it will be the ability to assess the trustworthiness of documents on the Web and the remarkable ease with which we'll be able to find the answers to our questions -- a process that is currently fraught with frustration.

Whatever the cause, almost everyone can find a reason to support this grand vision of the Semantic Web. Sure, it's a long way from here to there -- and there's no guarantee we'll make it -- but we've made quite a bit of progress so far. The possibilities are endless, and even if we don't ever achieve all of them, the journey will most certainly be its own reward.

Acknowledgements

Thanks to Sean B. Palmer for looking over a first draft of this article. He has since published The Semantic Web: An Introduction, which builds upon this piece. Jim Hendler and I published a paper based on this piece as "Swartz, A. and Hendler, J. The Semantic Web: A Network of Content for the Digital City, Proceedings Second Annual Digital Cities Workshop, Kyoto, Japan, October, 2001." Charles F. Munat* kindly cleaned up my writing, this latest version is updated with more and newer information.

Linked from: RDF Issues | The Semantic Web
Links to: Aaron Swartz | America On-Line (AOL) | Anatomy of an URL | DARPA Agent Markup Language with Ontology Inference Layer | Digital Signatures | Extensible Markup Language | Jim Hendler | N-Triples | Newbie | Notation3 | Ontology | RDF Schema | Resource Description Framework | Sandro Hawke | Schema | Sean B. Palmer | The Semantic Web (for Web Developers) | Tim Berners-Lee | URI Schemes | Weaving the Web | WebOntology (WebOnt) | World Wide Web Consortium | XML Namespaces
Last Modified: 2002-05-01 19:12:37