Tim Berners-Lee
Date: September 1998. Last modified: $Date: 1998/10/14 20:17:13 $

Status: An attempt to give a high-level plan of the architecture of the Semantic WWW. Editing status: Draft. Comments welcome

Up to Design Issues

Semantic Web Road map

A road map for the future, an architectural plan untested by anything except thought experiments.

This was written as part of a requested road map for future Web design, from a level of 20,000ft. It was spun off from an Architectural overview for an area which required more elaboration than that overview could afford.

Necessarily, from 20,000 feet, large things seem to get a small mention. It is architecture, then, in the sense of how things hopefully will fit together. So we should recognize that while it might be slowly changing, this is also a living document.

This document is 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).

Introduction

The Web was designed as an information space, with the goal that it should be useful not only for human-human communication, but also that machines would be able to participate and help. One of the major obstacles to this has been the fact that most information on the Web is designed for human consumption, and even if it was derived from a database with well defined meanings (in at least some terms) for its columns, that the structure of the data is not evident to a robot browsing the web. Leaving aside the artificial intelligence problem of training machines to behave like people, the Semantic Web approach instead develops languages for expressing information in a machine processable form.

This document gives a road map - a sequence for the incremental introduction of technology to take us, step by step, from the Web of today to a Web in which machine reasoning will be ubiquitous and devastatingly powerful.

It follows the note on the architecture of the Web, which defines existing design decisions and principles for what has been accomplished to date.

Machine-Understandable information: Semantic Web

The Semantic Web is a web of data, in some ways like a global database. The rationale for creating such an infrastructure is given elsewhere [Web future talks &c] here I only outline the architecture as I see it.

The basic assertion model

When looking at a possible formulation of a universal Web of semantic assertions, the principle of minimalist design requires that it be based on a common model of great generality. Only when the common model is general can any prospective application be mapped onto the model. The general model is the Resource Description Framework.

See the RDF Model and Syntax Specification

Being general, this is very simple. Being simple there is nothing much you can do with the model itself without layering many things on top. The basic model contains just the concept of an **assertion**, and the concept of **quotation** - making assertions about assertions. This is introduced because (a) it will be needed later anyway and (b) most of the initial RDF applications are for data about data ("metadata") in which assertions about assertions are basic, even before logic. (Because for the target applications of RDF, assertions are part of a description of some resource, that resource is often an implicit parameter and the assertion is known as a **property** of a resource).

As far as mathematics goes, the language at this point has no negation or implication, and is therefore very limited. Given a set of facts, it is easy to say whether a proof exists or not for any given question, because neither the facts nor the questions can have enough power to make the problem intractable.

Applications at this level are very numerous. Most of the applications for the representation of metadata can be handled by RDF at this level. Examples include card index information (the Dublin Core), Privacy information (P3P), associations of style sheets with documents, intellectual property rights labeling and PICS labels. We are talking about the representation of data here, which is typically simple: not languages for expressing queries or inference rules.

RDF documents at this level do not have great power, and sometimes it is less than evident why one should bother to map an application in RDF. The answer is that we expect this data, while limited and simple within an application, to be combined, later, with data from other applications into a Web. Applications which run over the whole web must be able to use a common framework for combining information from all these applications. For example, access control logic may use a combination of privacy and group membership and data type information to actually allow or deny access. Queries may later allow powerful logical expressions referring to data from domains in which, individually, the data representation language is not very expressive. The purpose of this document is partly to show the plan by which this might happen.

The Schema layer

The basic model of the RDF allows us to do a lot on the blackboard, but does not give us many tools. It gives us a model of assertions and quotations on which we can map the data in any new format.

We next need a schema layer to declare the existence of new property. We need at the same time to say a little more about it. We want to be able to constrain the way it used. Typically we want to constrain the types of object it can apply to. These meta-assertions make it possible to do rudimentary checks on a document. Much as in SGML the "DTD" allows one to check whether elements have been used in appropriate positions, so in RDF a schema will allow us to check that, for example, a driver's license has the name of a person, and not a model of car, as its "name".

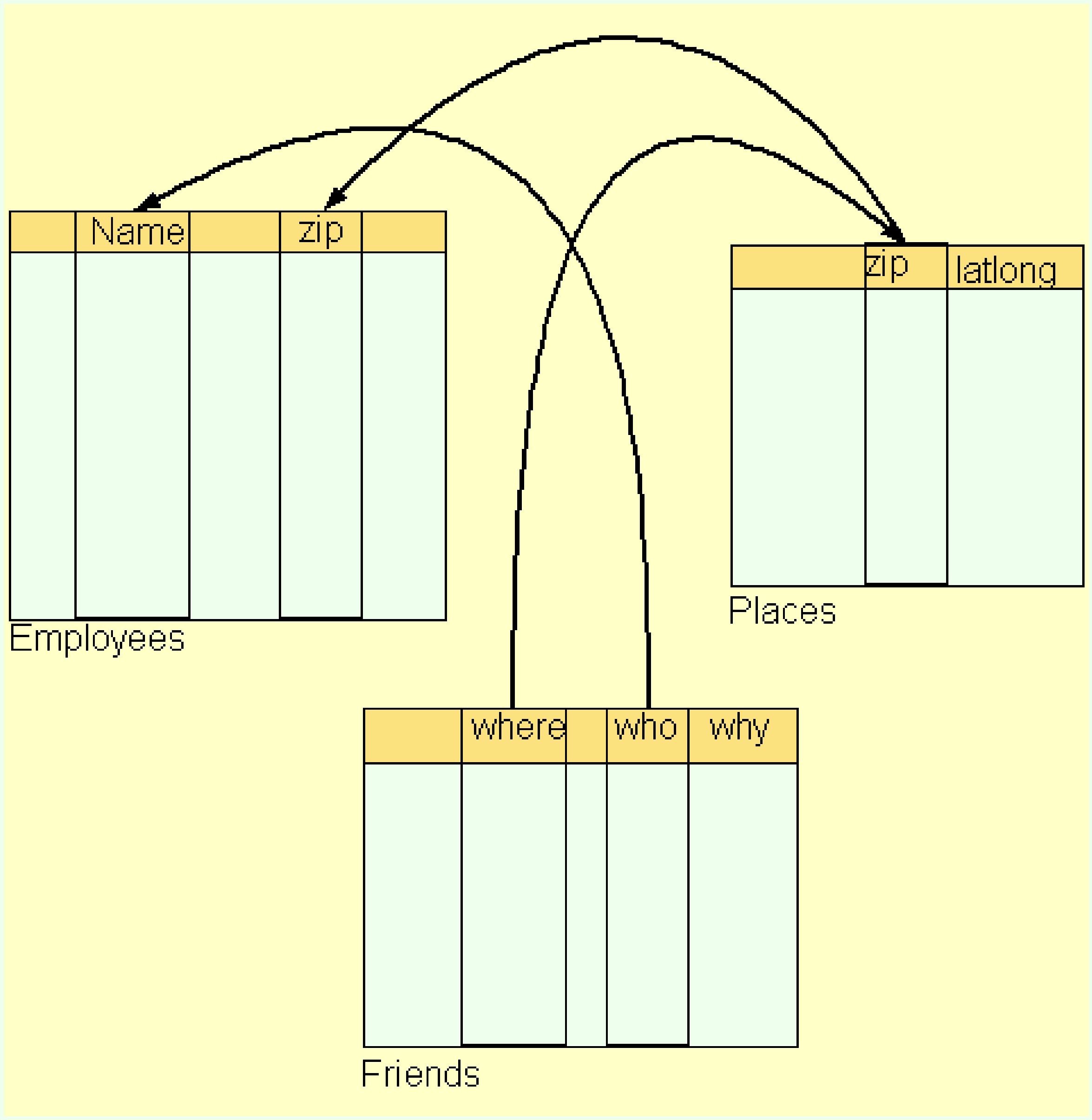
It is not clear to me exactly what primitives have to be introduced, and whether much useful language can be defined at this level without also defining the next level. There is currently a RDF Schema working group in this area. The schema language typically makes simple assertions about permitted combinations. If the SGML DTD is used as a model, the schema can be in a language of very limited power. The constraints expressed in the schema language are easily expanded into a more powerful logical layer expressions (the next layer), but one chose at this point, in order to limit the power, not to do that. For example: one can say in a schema that a property foo is unique. Expanded, that is that for any x, if y is the foo of x, and z is the foo of x, then y equals z. This uses logical expressions which are not available at this level, but that is OK so long as the schema language is, for the moment, going to be handled by specialized schema engines only, not by a general reasoning engine.

When we do this sort of thing with a language - and I think it will be very common - we must be careful that the language is still well defined logically. Later on, we may want to make inferences which can only be made by understanding the semantics of the schema language in logical terms, and combining it with other logical information.

Conversion language

A requirement of namespaces work for evolvability is that one must, with knowledge of common RDF at some level, be able to follow rules for converting a document in one RDF schema into another one (which presumably one has an innate understanding of how to process).

By the principle of least power, this language can in fact be made to have implication (inference rules) without having negation. (This might seem a fine point to make, when in fact one can easily write a rule which defines inference from a statement A of another statement B which actually happens to be false, even though the language has no way of actually stating "False". However, still formally the language does not have the power needed to write a paradox, which comforts some people. In the following, though, as the language gets more expressive, we rely not on an inherent ability to make paradoxical statements, but on applications specifically limiting the expressive power of particular documents. Schemas provide a convenient place to describe those restrictions.)



A simple example of the application of this layer is when two databases, constructed independently and then put on the web, are linked by semantic links which allow queries on one to converted into queries on another. Here, someone noticed that "where" in the *friends* table and "zip" in a *places* table mean the same thing. Someone else documented that "zip" in the *places* table meant the same things as "zip" in the *employees* table, and so on as shown by arrows. Given this information, a search for any employee called Fred with zip 02139 can be widened from *employees* to in include *friends*. All that is needed some RDF "equivalent" property.

The logical layer

The next layer, then is the logical layer. We need ways of writing logic into documents to allow such things as, for example, rules the deduction of one type of document from a document of another type; the checking of a document against a set of rules of self-consistency; and the resolution of a query by conversion from terms unknown into terms known. Given that we have quotation in the language already, the next layer is predicate logic (not, and, etc) and the next layer quantification (for all x, y(x)).

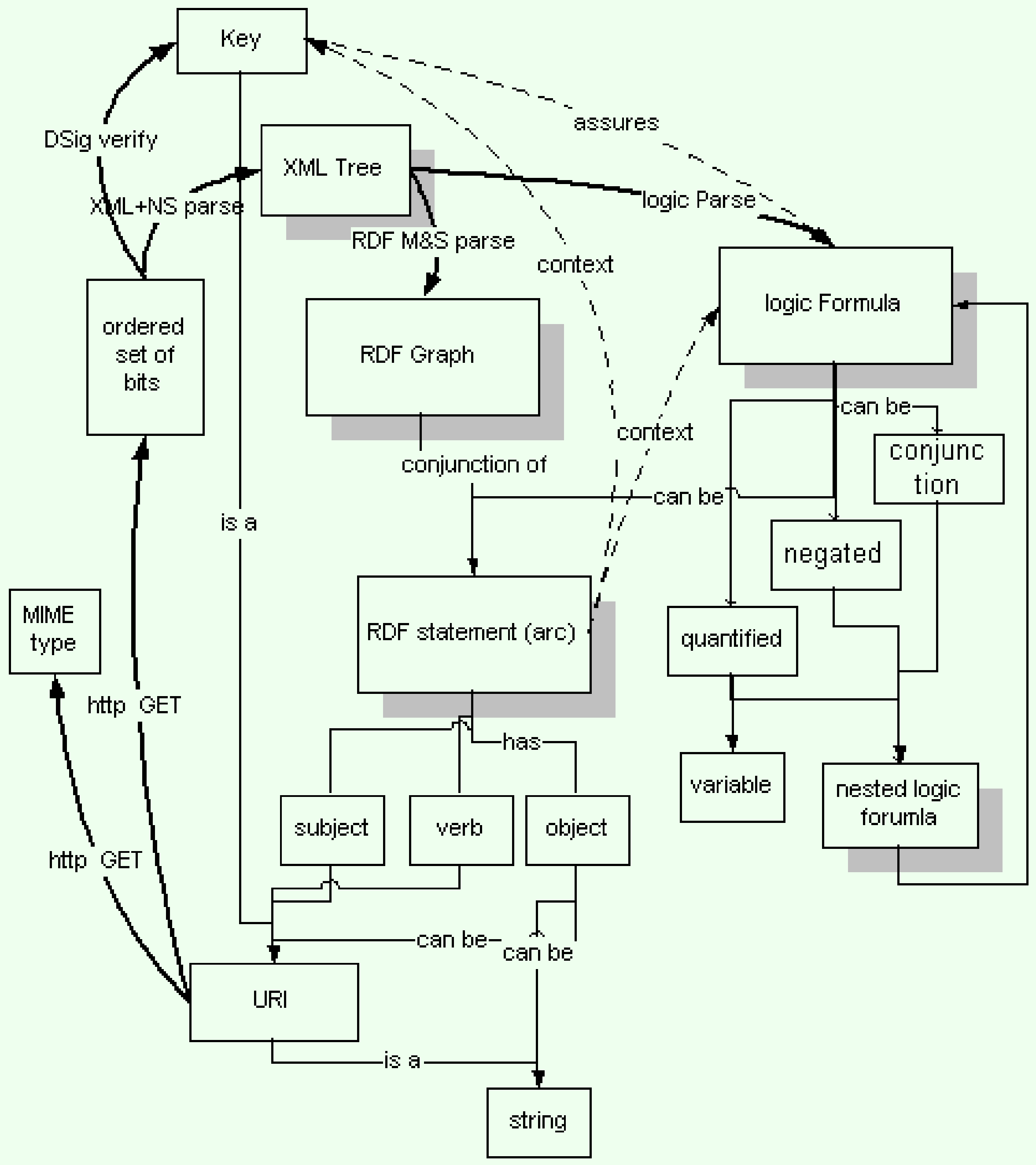
The applications of RDF at this level are basically limited only by the imagination. A simple example of the application of this layer is when two databases, constructed independently and then put on the web, are linked by semantic links which allow queries on one to converted into queries on another. Many things which may have seemed to have needed a new language become suddenly simply a question of writing down the right RDF. Once you have a language which has the great power of predicate calculus with quotation, then when defining a new language for a specific application, two things are required:

- One must settle on the (limited) power of the reasoning engine which the receiver must have, and define a subset of full RDF which will be expected to be understood;
- One will probably want to define some abbreviated functions to efficiently transmit expressions within the set of documents within the constrained language.

See also, if unconvinced:

- *What the Semantic Web is not* - answering some FAQs

The metro map below shows a key loop in the semantic web. The Web part, on the left, shows how a URI is, using HTTP, turned into a representation of a document as a string of bits with some MIME type. It is then parsed into XML and then into RDF, to produce an RDF graph or, at the logic level, a logical formula. On the right hand side, the Semantic part, shows how the RDF graph contains a reference to the URI. It is the trust from the key, combined with the meaning of the statements contained in the document, which may cause a Semantic Web engine to dereference another URI.



Proof Validation - a language for proof

The RDF model does not say anything about the form of reasoning engine, and it is obviously an open question, as there is no definitively perfect algorithm for answering questions - or, basically, finding proofs. At this stage in the development of the Semantic Web, though, we do not tackle that problem. Most applications construction of a proof is done according to some fairly constrained rules, and all that the other party has to do is validate a general proof. This is trivial.

For example, when someone is granted access to a web site, they can be given a document which explains to the web server why they should have access. The proof will be a chain [well, DAG] of assertions and reasoning rules with pointers to all the supporting material.

The same will be true of transactions involving privacy, and most of electronic commerce. The documents sent across the net will be written in a complete language. However, they will be constrained so that, if queries, the results will be computable, and in most cases they will be proofs. The HTTP "GET" will contain a proof that the client has a right to the response. the response will be a proof that the response is in deed what was asked for.

Evolution rules

Language

RDF at the logical level already has the power to express inference rules. For example, you should be able to say such things as "If the zipcode of the organization of x is y then the work-zipcode of x is y". As noted above, just scattering the Web with such remarks will in the end be very interesting, but in the short term won't produce repeatable results unless we restrict the expressiveness of documents to solve particular application problems.

Two fundamental functions we require RDF engines to be able to do are

1. for a version *n* implementation to be able to read enough RDF schema to be able to deduce how to read a version *n+1* document;
2. for a type A application developed quite independently of a type B application which has the same or similar function to be able to read and process enough schema information to be able to process data from the type B application.

(See evolvability article)

The RDF logic level is sufficient to be usable as a language for making inference rules. Note it does not address the heuristics of any particular reasoning engine, which which is an open field made all the more open and fruitful by the Semantic Web. In other words, RDF will allow you to write rules but won't tell anyone at this stage in which order to apply them.

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? In the case of a program which finds a version 2 document and wants to find the rules to convert it into a version 1 document, then the version 2 schema would naturally contain or point to the rules. In the case of retrospective documentation of the relationship between two independently invented schemas, then of course pointers to the rules could be added to either schema, but if that is not (socially) practical, then we have another example of the the annotation problem. This can be solved by third party indexes which can be searched for connections between two schemata. In practice of course search engines provide this function very effectively - you would just have to ask a search engine for all references to one schema and check the results for rules which like the two.

Query languages

One is a query language. A query can be thought of as an assertion about the result to be returned. Fundamentally, RDF at the logical level is sufficient to represent this in any case. However, in practice a query engine has specific algorithms and indexes available with which to work, and can therefore answer specific sorts of query.

It may of course in practice to develop a vocabulary which helps in either of two ways:

1. It allows common powerful query types to be expressed succinctly with fewer pages of mathematics, or
2. It allows certain constrained queries to be expressed, which are interesting because they have certain computability properties.

SQL is an example of a language which does both.

It is clearly important that the query language be defined in terms of RDF logic. For example, to query a server for the author of a resource, one would ask for an assertion of the form "x is the author of p1" for some x. To ask for a definitive list of all authors, one would ask for a set of authors such that any author was in the set and everyone in the set was an author. And so on.

In practice, the diversity of algorithms in search engines on the web, and proof-finding algorithms in pre-web logical systems suggests that there will in a semantic web be many forms of agent able to provide answers to different forms of query.

One useful step the specification of specific query engines for for example searches to a finite level of depth in a specified subset of the Web (such as a web site). Of course there could be several alternatives for different occasions.

Another metastep is the specification of a query engine description language -- basically a specification of the sort of query the engine can return in a general way. This would open the door to agents chaining together searches and inference across many intermediate engines.

Digital Signature

Public key cryptography is a remarkable technology which completely changes what is possible. While one can add a digital signature block as decoration on an existing document, attempts to add the logic of trust as icing on the cake of a reasoning system have to date been restricted to systems limited in their generality. For reasoning to be able to take trust into account, the common logical model requires extension to include the keys with which assertions have been signed.

Like all logic, the basis of this, may not seem appealing at first until one has seen what can be built on top. This basis is the introduction of keys as first class objects (where the URI can be the literal value of a public key), and a the introduction of general reasoning about assertions attributable to keys.

In an implementation, this means that reasoning engine will have to be tied to the signature verification system . Documents will be parsed not just into trees of assertions, but into into trees of assertions about who has signed what assertions. Proof validation will, for inference rules, check the logic, but for assertions that a document has been signed, check the signature.

The result will be a system which can express and reason about relationships across the whole range of public-key based security and trust systems.

Digital signature becomes interesting when RDF is developed to the level that a proof language exists. However, it can be developed in parallel with RDF for the most part.

In the W3C, input to the digital signature work comes from many directions, including experience with DSig1.0 signed "pics" labels, and various submissions for digitally signed documents.

Indexes of terms

Given a worldwide semantic web of assertions, the search engine technology currently (1998) applied to HTML pages will presumably translate directly into indexes not of words, but of RDF objects. This itself will allow much more efficient searching of the Web as though it were one giant database, rather than one giant book.

The Version A to Version B translation requirement has now been met, and so when two databases exist as for example large arrays of (probably virtual) RDF files, then even though the initial schemas may not have been the same, a retrospective documentation of their equivalence would allow a search engine to satisfy queries by searching across both databases.

Engines of the Future

While search engines which index HTML pages find many answers to searches and cover a huge part of the Web, then return many inappropriate answers. There is no notion of "correctness" to such searches. By contrast, logical engines have typically been able to restrict their output to that which is provably correct answer, but have suffered from the inability to rummage through the mass of intertwined data to construct valid answers. The combinatorial explosion of possibilities to be traced has been quite intractable.

However, the scale upon which search engines have been successful may force us to reexamine our assumptions here. If an engine of the future combines a reasoning engine with a search engine, it may be able to get the best of both worlds, and actually be able to construct proofs in a certain number of cases of very real impact. It will be able to reach out to indexes which contain very complete lists of all occurrences of a given term, and then use logic to weed out all but those which can be of use in solving the given problem.

So while nothing will make the combinatorial explosion go away, many real life problems can be solved using just a few (say two) steps of inference out on the wild web, the rest of the reasoning being in a realm in which proofs are give, or there are constrains and well understood computable algorithms. I also expect a string commercial incentive to develop engines and algorithms which will efficiently tackle specific types of problem. This may involve making caches of intermediate results much analogous to the search engines' indexes of today.

Though there will still not be a machine which can guarantee to answer arbitrary questions, the power to answer real questions which are the stuff of our daily lives and especially of commerce may be quite remarkable.

In this series:

- *What the Semantic Web is not* - answering some FAQs of the unconvinced.
- Evolvability: properties of the language for evolution of the technology
- Web Architecture from 50,000 feet

References

The CYC Representation Language

Knowledge Interchange Format (KIF)

@@

Acknowledgements

This plan is based in discussions with the W3C team, and various W3C member

companies. Thanks also to David Karger and Daniel Jackson of MIT/LCS.

[Up to Design Issues]