Tim Berners-Lee

Date started: January 6, 1997

Status: personal view, but corresponds generally to the W3C architecture for metadata.

Editing status: Italic text is rough. Requires complete edit and possibly massaging, but content is basically there.

I just found Ralph's pointer to this so I am adding stuff (which I had accidentally lost), as people might be actually reading this. (27/Feb/97). Additions are at the end about consistency in label/metaset/collection syntax and semantics.

The syntaxes used in this document are meant to illustrate the architecture and be clear but are otherwise random.

Up to Design Issues

Axioms of Web Architecture: 3

Metadata Architecture

Documents, Metadata, and Links

The thing which you get when you follow a link, when you de-reference a URI, has a lot of names. Formally we call it a **resource**. Sometimes it is referred to as a document because many of the things currently on the Web are human readable documents. Sometimes it is referred to as an object when the object is something which is more machine readable in nature or has hidden state. I will use the words document and resource interchangeably in what follows and sometimes may slip into using "object".

One of the characteristics of the World Wide Web is that resources, when you retrieve them, do not stand simply by themselves without explanation, but there is information about the resource. Information about information is generally known as **Metadata**. Specifically, in the web design,

Definition

Metadata is machine understandable information about web resources or other things

The phrase "machine understandable" is key. We are talking here about information which software agents can use in order to make life easier for us, ensure we obey our principles, the law, check that we can trust what we are doing, and make everything work more smoothly and rapidly. Metadata has well defined semantics and structure.

Metadata was called "Metadata" because it started life, and is currently still chiefly, information about web resources, so data about data. In the future, when the metadata languages and engines are more developed, it should also form a strong basis for a web of machine understandable information about anything: about the people, things, concepts and ideas. We keep this fact in our minds in the design, even though the first step is to make a system for information about information.

For an example of metadata, when an object is retrieved using the HTTP protocol, the protocol allows information about its date, its expiry date, its owner, and other arbitrary information to be sent by the server. The world of the World Wide Web is therefore a world of information and some of that information is information about information. In order to have a coherent picture of this, we need a few axioms about metadata. The first axiom is that :

Axiom

metadata is data.

That is to say, information about information is to be counted in all respects as information. There are various parts of this.

One is that metadata can be stored regarded as data, it can be stored in a resource. So, one resource may contain information about itself or about another resource. In current practice on the World Wide Web there are three ways in which one gets metadata. The first is the data about a document contained within the document itself, for example in the HEAD part of an HTML documents or within word processor documents. The second is that during the HTTP transfer the server transfers some metadata to the client about the object which is being transferred. This, during an http GET, is transferred from the server to the client and, during a PUT or a POST, is transferred from the client to the server. One of the things which we have to rationalize in our architecture of the World Wide Web is who exactly is making the statement. Whose statement, whose property is that metadata. The third way in which metadata is found is when it is looked up in another document. This practice has not been very common until the PICS initiative was to define label formats specifically for representing information about World Wide Web resources. The PICS architecture specifically allows for PICS labels which are resources about other resources to be buried within the resource itself, to be retrieved as separate resources, or to be passed over during the http transaction. To conclude,

Metadata about one document can occur within the document, or within a separate document, or it may be transferred accompanying the document.

Put another way, metadata can be a first class object.

The second part of the above axiom is:

Metadata can describe metadata

That is, metadata itself may have attributes such as ownership and an expiry date, and so there is meta-metadata but we don’t distinguish many levels, we just say that metadata is data and that from that it follows that it can have other data about itself. This gives the Web a certain consistency.

The Form of Metadata

Metadata consists of assertions about data, and such assertions typically, when represented in computer systems, take the form of a name or type of assertion and a set of parameters, just as in the natural language a sentence takes the form of a verb and a subject, an object and various clauses.

Axiom

The architecture is of metadata represented as a set of independent assertions.

This model implies that in general, two assertions about the same resource can stand alone and independently. When they are grouped together in one place, the combined assertion is simply the sum (actually the logical AND) of the independent ones. Therefore (because AND is commutative) collections of assertions are essentially unordered sets. This design decision rules out for example, in simple sets of data, assertions which are somehow cumulative or later ones override earlier ones. Each assertion stands independently of others.

We will see below how logical expressions are formed to combine assertions in more varied ways, and syntactic rules which allow the subject at least of the assertion to be made implicit. But neither of these change the basic operation of combining assertions in unordered AND lists.

Attributes

Assertions about resources are often referred to as attributes of the resource. That is, the type of assertion is an assertion that the object, the resource in question, has a particular named property such as it’s author, and in that case the parameter is the name or identity of the author. Similarly, if the attribute is the document’s date of expiry then the parameter is that date.

Often, a group of assertions about the same resource occur together, in which case the syntax generally omits the URI of that resource as it is implicit. In these cases, when it is clear from the context about which resource the assertion is being made, the assertion often takes the form of a list of attributes and values. In RFC822 format messages, such as mail messages and HTTP messages, metadata is transferred where the attribute name is an RFC822 header name and the rest of the RFC822 line is the value of the attribute, such as Date: and From: and To: information. The attribute value pair model is that used by most activities defining the semantics of metadata today.

I use the word "assertion" to emphasize the fact that the attribute value pair when it is transferred is a statement made by some party. It does not simply and directly imply that the resource at any given time has that value for the given attribute. It must be seen as a statement by a particular party with or without implicit or explicit guarantees as to validity. Throughout the World Wide Web, as trust becomes an important issue, it will be important for software -- and people -- to keep track of and take into account who said what in terms of data and metadata. So, our model of data of a resource is something about which typically we know the creator or the person responsible, and typically the date of which the information was created, which implies, in the case of a piece of information which makes an assertion, the date at which the assertion was made.

An assertion

(A u1, p, q...)

typically has as explicit parameters,

- the URI of the resource about which the assertion is made (u1).
- some identifier (A) for the type of assertion being made, such as author or date or expiry date.
- other parameters (p, q,...) according to the type of assertion.

As implicit or explicit or implicit parameters,

- The party making the assertion
- The date/time of the assertion

We can often make an analogy with programming languages. An assertion in metadata can be compared with a function call in a programing language. In object oriented languages, the object of the function has a special place among the parameters just as the subject of an assertion does in metadata. In object oriented languages, though, the set of possible functions depends on the object, whereas in metadata the set of assertion types is more or less unlimited, defined by independent choice of vocabulary.

A space for attribute names

It is appropriate for the Web architecture to define like this the topology and the general concepts of links and metadata. What about the significance of individual relationships? Sometimes, as above, these are special, defined in the architecture, and having an architectural significance or a significance to the protocols. In other cases, the significance of relationships or indeed of attributes is part of other specifications, other design, or other applications, and must be defined easily by third parties. Therefore, the set of such relationship and attributes names must be extremely easily extensible and therefore extensible in a decentralized manner. This is why

the URL space is an appropriate space for the definition of attribute names.

We have now several vocabularies of attribute names: for example, the HTML elements which can occur within the HEAD element o,r as another example, the headers in an HTTP request which specify attributes of the object. These are defined within the scope of particular specifications. There is always pressure to extend these specifications in a flexible way. HTTP header names are generally extended arbitrarily by those doing experiments. The same can also be true of HTML elements and extension mechanisms have been proposed for both. If we look generically at the very wide space of all such metadata attribute names, we find something in which the dictionary would be so large that ad hoc arbitrary extension would be just as chaotic as central registration would be stifling.

> **Aside: Comparison with Entity-Relationship models**.
>
> This architecture, in which the assertion identifier is taken from (basically) URL space differs from the "Entity-relationship" (ER) model and many similar models like it, including most object-oriented programming systems. In an ER model, typically every object is typed and the type of an object defines the attributes can have, and therefore the assertions which are being made about it. Once a person is defined as having a name, address and phone

number, then the schema has to be altered or a new derived type of person must be introduced before one can make assertions about the race, color or credit card number of a person. The scope of the attribute name is the entity type, just as in OOP the scope of a method name is an object type (or interface)By contrast, in the web, the hypertext link allows statements of new forms to be made about any object, even though (before anything other than syntax checking) this may lead to nonsense or paradox. One can define a property "coolness" within one's own part of the web, and then make statements about the "coolness" of any object on the web.

This design difference is in essence a resurfacing of the decision to make links mondirectional, sacrificing consistency for scalability.

An advantage of ER systems is that they allow one to work, in the user interface for example, with a set of properties which "should" be defined for each entity. You can define these in the Metadata's predicate calculus by defining an expression for a "well specified" object. ("For all *X* such that *X* is a customer *X* is well-specified if there exists *n* such that *n* is the name of *X* and there exists *t* such that *t* is the telephone number of *X* and...)

end of aside.

Metadata ("Entity")headers in HTTP

In the above it is important to realize that the HTTP headers which contain what can be considered as metadata ("entity headers") should be separated quite distinctly from HTTP headers which do not. HTTP headers which contain metadata contain information which can follow the document around. For example, it is reasonable for a cache to pass such information on without treatment, it is reasonable for clients or other programs which process data to store those headers as metadata with the document for later processing. The content of those headers do not have to be associated with that particular HTTP transaction. By contrast, the RFC822 headers in HTTP which deal specifically with the transaction or deal specifically with the TCP link between the two application programs have a shorter scope and can only be regarded as parameters of the HTTP method. To make this separation clear will be to make it easier not only to understand HTTP and how it should be processed, it will also make it clear which pieces of HTTP can be used easily and transparently by other protocols which may use different methods with different parameters. The clarification of the architecture of HTTP such that both the metadata and the methods can be extended into other domains is an important part of the work of the World Wide Web Consortium. The Internet protocols SMTP and NNTP and HTTP as well as many new and proposed protocols share much of the semantics of the RFC822 headers. Formalizing the shared space and making it clear that there is a single design for a particular header, rather than four designs which are independent and happen to look very similar, requires a general architecture, some careful thought, and is essential for the future design of protocols. It will allow protocol design to happen in small groups which can take for granted the bulk of previous work and concentrate on independent new design.

Authorship of HTTP entity headers

It may be possible to remove or at least encompass the apparent anomaly of metadata transferred from an HTTP server by creating a special link type which links the document itself to the set of attributes which the server would give in the HTTP headers. In other words, the server would be able to say, "here is a document, here is some metadata about it, and the metadata about it has the following URL". This would allow one, for example, request a signed copy of the HTTP headers. It would allow one to ask about the intellectual property rights of those headers, and the authorship of those headers.

It is important to be completely clear about the authorship of the HTTP headers. The server should be seen as a software agent acting on behalf of a party which is the publisher or document author: the definer of the URI to resource identity mapping. The webmaster is only an administrator who is responsible for ensuing that (through an appropriately configured server) the transactions on the wire faithfully represent the statements and wishes of that party.

Links

An assertion of relationship between two resources is known as a **link**.

In this case, it is a triple

> (*A u1 u2*)

of:

- the type of assertion being made, that is, the relationship which is being asserted,
- the first URI,
- and the second URI.

These sorts of assertions, links, are the basis of navigation in the World Wide Web; they can be used for building structure within the World Wide Web and also for creating a semantic Web which can express knowledge about the world itself. That is to say, links may be used both for the structure of data, in which case they are metadata, but also they may be used as a form of data.

Links, like all metadata can be transferred in three ways. They can be embedded in a document, which is one end of the link, they can be transferred in an HTTP message, for example what is called the header of the document, and they can be stored in a third document. This latter method has not been used widely on the World Wide Web to date.

Goal: Self-describing information

A critical part of the design of the whole system is the way that the semantics of metadata or indeed of data are defined. The semantics of metadata in our RFC822 headers in mail messages and in http messages are defined by hand in english in the specifications of those protocols. The PICS system takes this to one stage further in terms of flexibility by allowing a message to contain a pointer to the document which defines, in human readable terms, the semantics of each assertion made within a PICS label. In the future we would like to move toward a state in which any metadata or eventually any form of machine readable data carries a reference to the specification of the semantics of all the assertions made within it.

For example, suppose that when a link is defined between two documents, the relationship which is being asserted is defined in a such way that it can be looked up on the World Wide Web (i.e. using some form of URI), and someone or some program, which has not come across that relationship before can follow the link and extend its understanding or functionality to take advantage of this new form of assertion.

In the case of PICS, one can dynamically pick up a human readable definition of what that assertion really means. In PICS (and in theory in SGML using DTDs), one can also pick up a machine readable definition of what form that assertion can take, what syntax, what types of parameters it can take. This allows a human interface to a new PICS scheme to built on the fly. To go one step further, one could, given a suitable logic or knowledge representation language, pick up a machine readable definition of the semantics of that assertion in terms of other relationships.

The advantages of such self describing information is that it allows development of new applications and new functionality independently by many groups across the web. Without self-describing information, development must wait for large companies or standards committees to meet and agree on the commonly agreed semantics.

Of course a pragmatic way of extending software to handle new forms of information is to dynamically download the code to support a software object which can handle such data for one. Whereas this is a powerful technique, and one which will be used increasingly, it is not sufficient. It is not sufficient because one has to trust the implementation of the object, and the state.

Goal

As much as possible of the syntax and semantics should be able to be acquired by reference from a metadata document.

Building Applications using Link Relationships

It turns out that a very large number of applications both built on top of the web and also built within the infrastructure of the Web can largely be built by defining new relationship types. Examples of these are the document versioning problem which can be largely solved by defining link values relating documents to previous and future versions and to lists of versions; intellectual property rights, distribution terms, and other labeling which can be solved by making a link from one document to the document containing the metadata.

Summary so far

1. Metadata is data
2. Metadata may refer to any resource which has a URI
3. Metadata may be stored in any resource no matter to which resource it refers
4. Metadata can be regarded as a set of assertions, each assertion being about a resource (A *u1* ...).
5. Assertions which state a named relationship between two resources are known links (A *u1 u2*)
6. Assertion types (including link relationships) should be first class objects in the sense that they should be able to be defined in addressable resources and referred to by the address of that resource A in { u }

7. The development of new assertion types and link relationships should be done in a consistent manner so that these sort of assertions can be treated generically by people and by software.

Rough from here on down

Label syntax: extracting u1

When labeling information, it is often useful to make a lot of statements about one object (whose URI is, say, u1) and it is also useful to be able to make the same set of statements about a set of resources. Hence in the case in which the assertions

```
(A1 u1  a b ... )
 (A2 u1  c d )
 (A2 u1  a f g h )
```

are written

```
(for u1
        (A1 a b ... )
        (A2 c d )
        (A3 a f g h )
 )
```

Therefore in the syntax of an actual assertion the subject is implicit. This is just the case with RFC822 headers which implicitly refer to the following body, with HTML "HEAD" element contents which implicitly refer to the containing document. (Though notice there is a fundamental difference, discussed below, between a general label and a message header because the message header is definitive.)

(Do we want to lose the syntax for making a fully qualified assertion altogether?)

Assertions, when the subject is implicit, are known as attribute value pairs as discussed above. Let's use the term "label" for a set of assertions with the subject extracted. Like the label on a jam jar, it contains information but there must be something else (in this case if its placement on the jar) which tells you to what it applies.

A label is a set of assertions with a common implicit subject. In this architecture it is a set of attribute-value pairs

(There is a convention that you can write "Jam" on a jam jar label. You don't write "Jam jar" or "Jam Jar label". Even though I once saw a label on a cardboard box with the words "Equipment shipping box label" on it)

Authorship of Metadata

It follows from the fact that metadata is data that here can be metadata about it. Some of this metadata becomes crucial when we consider a trust model. The logic we need includes the author of metadata

p1: (A u1 . . .)

where p1 is ,in a system with low trust, the author as stated, but in a cryptographically secure system is a principle represented by a key.

Analysing labels

PICS labels, already developed with a practical application in mind, give us a foil against which to test our notions of metadata architecture. PICS labels contain, basically, 4 types of information:

- Lexicon pointers to define the vocabulary used for later assertions (PICS version, scheme URI)
- Subject specifiers which identify the set of objects (`for`, `generic`,`md5-hash`)
- Qualifiers about the label itself, which in fact are metametadata (by, until,...)
- Actual assertions about the subject (ratings)

When looking at a new metedata format constraints I would include as **requirements for cleanliness** would include:

1. The syntax used for defining the basic PICS version (vocabulary pointer) should be the same as that for importing any other vocabulary. This is consistency. It makes a PICS label self-describing in its own terms. The "vocabulary import" operation becomes axiomatic, but all else follows.
2. The subject specifiers should be URIs and we should be very careful about introducing anything else. In particular, the md5-hash can be regarded as additional information for security about a URI, and the timestamp can be regarded as a timestamp on the statement. This latter we discuss below .
3. The syntax and vocabulary (subject to which lexicons are actually referenced) should be identical for the metadata and the metametadata.

To elaborate on the last point, notice that the PICS "by x" is equivalent of the HTTP "From: x". It is a metadata statement about the following information, ie a header. Now, it is true that the semantics of these "label options" are mandatory (not optional at all for the reader, only for the writer) and have to be understood by the reasoning engine. They should not however be special cases.

Qualifiers: Metametadata

Let's look at the semantics of a typical qualifier for a moment.

(`until` *date assertion1*)

is an assertion that assertion1 is true up to a certain date. Put that way, it looks like a nesting operation. However when we combine two label options, they are in fact unordered.

```
(by fred (until friday itsraining))
```

is the same as

```
(until friday (by fred itsraining)
```

This assumption is made when PICS (or RFC822) declare label options to be an unordered list. (Is it valid? If not, the syntax should indicate that the order is important as in for example:

```
by fred: until friday: assertion1
```

or indeed using nested brackets. But assuming it *is* valid, then in fact the semantics are

```
(you don't believe fred) or (it is after friday) or (itsraining)
```

where the "or" is commutative, so the list is unordered. This last statement is not couched as metametadata - it has been reduced to data. In fact, I believe that this factoring out of the "until" clause is fine -- it is quite equivalent when expressed without metametadata, but the factoring out of the "by" clause is not fine. The "You disbelieve fred OR it is raining" is not equivalent to "Fred says ‘it is raining’". So we do need qualifiers and metametadata in the syntax.

In fact the simplest and most consistent way to put the PICS qualifiers in is to make a label about the label. When we introduce the concept of a message below, the "by fred" part naturally falls into the message header just as with email.

Timestamps

@@@

The concepts of generic resources allows a URI to refer to something which can be a living document or a frozen one. The usual case is that documents are living documents or even if frozen, the server is not aware of this, and so neither can the client be. In any case, rarely for a living document is a server smart enough and wise enough to provide for the referer second URL for the specific version.

Therefore, PICS 1.1 labels allow the subject to be identified by a URI with the optional qualification of a date-time, "for *u* at *t*" meaning "the resource *u* as it was at time *t*". This is used as an identifier which effectively meets the need for a version-specific URI when that is not available.

What is awkward about this is that if that is what is needed to refer to a document, then logically it should be introduced int the URI syntax or (worse) the syntax of every place in which URIs are used, from newspaper articles to bookmark lists. It is much cleaner when a server issues a special version-specific URI. That reduces reference to a simple URI again.

This is one option. The selection clause is just a URI and any other information (at, hash) is just informational stuff for verification.

Subject selection clauses

Working in the other direction, toward more complex subject specifiers, we can few the specification of the subject as an expression of arbitrary complexity:

"For all documents *D* such that URI *u* dereference to *D* and the last modified time of *D* is *t*, fred asserts that *D* is ok"

This is quite familiar as mathematics (though we don't have the formula capability in HTML yet for this document!) not to mention SQL. PICS already has "for" and "at", so perhaps "such that" would in fact be a simplification. You can alternatively transform the statement into the "OR" form above which might be less familiar but is simpler:

"For all documents *D*, either *u* does not dereference to *D* or *D* was not last modified at *t* or *D* is OK"

let's look at what this looks like in a mixed made up syntax (| means or, ~ means not)

(~URI u) | (~date < friday) | (ok)

or if you like with "=>" meaning assertion,

((URI u) & (before friday)) => (ok)

or for that matter, using "if",

if (URI u) if (before Friday) (ok)

These syntaxes are all basically equivalent. They are **consistent** in that the properties ("URI", "date", etc.) are taken from a normal vocabulary, in the same way as for the assertions which are qualified ("ok", etc.).

There is a decision to make as to whether the qualifiers should be separated from the assertions. The qualifiers are typically OR'd together, and the assertions are typically ANDed. So unless you have explicit operators, you need a grouping at the level of PICS "ratings" block to separate implicit OR from implicit AND.

In the breakdown below, subject specifier clauses are called spec-labels: they are labels specify o refer to the

Selection clauses a la PICS ratings

In fact, the PICS system has more than simple binary assertions. It involves values which visibly have floating point values. There is already a defined algebra for a selection clause: the PICS filter. There can be expressions in PICS filters can be "If (sax>5) and (violins < 2)". Assuming the metadata algebra has to have the power of PICS, then the selection cause must have the power of a PICS filter:

```
from webmaster@w3.org
 date  2/2/92
 For all documents such that
        Date is in range [1/1/91 .. 9/9/99]
   and  URI is like "http://www.w3.org/pub/WWW/TR/REC-*"
   and   Expiry date is greater than 3/3/97
 assert
        sex 0
```

```
   and  violence 1
```

The metadata architecture, to be consistent, must use selection clauses with the same power and syntax whenever a selection clause is required.

A clean way to look at a subject specifier is as a label such that if the label applies to any resource, then that resource is included. I refer to this as a spec-label in the summary.

Granularity

At which granularity should one be able to wrap metametadata around metadata? Languages in which program blocks are different from program files are very frustrating, so a cleanliness requirement is that you can recursively put qualifiers around nested metadata at any granularity. This means that within one document you may have assertions by with or from many different parties. If you regard a basic PICS label as using two vocabularies, the scheme vocabulary and the PICS vocabulary, that you can have nesting both ways.

Whenever a bit of metadata is wrapped up in such a way that metametadata can be expressed about it, it is also useful to be able to give it a name, for reference, within the metadata document. For similar reasons,

Mixing vocabularies

It is naive to imagine that metadata resources will each use one vocabulary, or use at any one nesting level a single vocabulary. By analogy with programming languages, this would be like allowing the only one imported module to be in scope at any one place. This would certainly not be powerful enough for programming. It would lead to the creation of dummy modules whose only purpose is to merge access to more than one other module. The same constraint in the metadata architecture would lead similarly to the creation of dummy lexicons whose sole purpose was to merge other lexicons.

A syntax which allows more than one vocabulary to be imported would overcome this. Here is an arbitrary one.

```
(lexicon http://w3.org/voc/v1 as a )
 (lexicon http://oclc.org/voc/dc1 as d)

 (ratings
      (a.author fred)
      (a.until friday)
      (d.loccat4 123.7.123.8)
 )
```

The general requirement is:

It must be possible to mix multiple vocabularies within the same scope.

Messages

A message, in the email sense, is notion which is not covered so far.

A message is a document with an author and a timestamp, and perhaps other definitive information about itself, with a document which is its body.

The header of a message is a set of attribute-value pairs not unlike a label: indeed it is a label, but it is a special label which is definitive. The difference is that a normal "by fred" label means "fred says", a head "by fred" attribute means "fred hereby says".

The header of a message is also special because it applies not to the body of the message, but to the whole message **including itself**. This is what makes a message a particular and special part of the architecture. It is the only place where you can logically put a signature or a from: field.

(Note that in email, often extra headers such as "Received-by" are added to a message but really logically they can be regarded as nested messages. A signature on the whole message would not of course work if you start adding headers. So messages in this document are not exactly the same as email messages -- its just a good analogy.)

Ignoring the punctuation and keywords of a real syntax, here is the breakdown of things we have to date:

```
message    ::   head
                body

 head      ::   label

 label     ::   ( attribute value )*

 body      ::   statement *   | body-label body   (note 1)

 statement  ::  subject label   | message        (note 2)

 body-label ::  label

 subject    ::  URI     |    spec-label

 spec-label ::  label
```

Note 1. We have allowed here the recursive ability to take a body of statements and add a **body-label** to them. This is distinct from a message header: it is simply some information and some information about that information. This is to allow the granularity of metadata discussed above.

Note 2. We have allowed that one form of statement can be a message. That takes the form of an assertion that the enclosed message is or was made. It is different from a body-label.

Algebraic Manipulations

If you can make assumptions about the properties of labels then you can manipulate them, possibly without knowing everything about their meaning. Properties such as commutativity, transitivity and associativity would be very useful to have easily available: perhaps in the syntax, or failing that in the schema.

For example, given a label saying a pair of jeans has a 32" waist and a price of $28, I can deduce a label which just has the price of $28. But given a label which says that the punishment for the crime is a 2 month in jail and a fine of $3000, I can't deduce one that says that that the punishment is 2 months in jail.

A typical use of metadata will be to provide a statement along with its proof to be verified by another party. Being able to process these things efficiently and with limited knowledge will be crucial.

Ordered/Unordered

The axiom of independence of assertions above gives us that in any set of assertions, as assertions are independently true, specific assertions may be removed or reordered, leaving the document just as valid (though possibly less informative).

Examples of unordered things currently are: RFC822 message header lines, SGML attributes. Examples of ordered things are: HTTP header lines and SGML elements.

Do we need a form in which we can make an assertion which has many parameters which are in fact not mutable in any way?

Summary of Requirements

There are ways of representing the above things: messages, labels, specifying labels, and statements and distinguish between them.

As much as possible of the syntax and semantics should be able to be acquired by reference from a metadata document.

It must be possible to mix multiple vocabularies within the same scope.

The syntax and structure should be such that as many manipulations as possible can be done without having to know the semantics of the vocabulary in use.

(list from paper)

Tim BL, January 1997

Last edit $Date: 1998/09/25 03:53:21 $