Web Architecture: Extensible Languages

W3C Note 10 Feb 1998

This Version:
http://www.w3.org/TR/1998/NOTE-webarch-extlang-19980210
Latest Version:
http://www.w3.org/TR/NOTE-webarch-extlang
Authors:
Tim Berners-Lee <timbl@w3.org> W3C
Dan Connolly <connolly@w3.org> W3C

Status of This Document

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE.

This work is related to the Architecture domain of the W3C, and particularly to the XML activity, but is related to HTML, HTTP and Metadata activities.

Comments should be sent to the authors and www-talk@w3.org.

This document is meant to be a fairly explanatory synthesis of the requirements for namespace extension in languages on the web, and in particular for the general language planned to be the common basis of many future applications, XML. It was originally written as part of the "Design Issues" series of notes. Whilst technically the personal opinion of the authors, it their best attempt as technical coordinators at outlining common architectural principles for W3C development.

At the time of writing [1998/02], various drafts in the XML and RDF community address these requirements in various ways. The document may evolve if further clarity is seen to be needed, or further requirements added. Some open issue are noted.

Abstract

Experience with the task of coordinating developments by independent groups has allows us to define properties of languages which will allow the unfettered growth of the Web technology in a chaotic but still well defined way. These take the form of constraints on the language features for making reference to multiple different vocabularies, and on languges for "schema" documents which define those vocabularies.

Contents

Introduction

When the World Wide Web Consortium was first put together, high on the list of goals of the Consortium was making the web "evolvable". At that time, it was a philosophical goal and it wasn't clear what it would mean technically. Since then, W3C has had plenty of experience in the deployment of new technology, particularly in an environment of thousands of independent groups developing in closely related or identical fields.

The HTTP and HTML specifications have both grown rapidly in this environment. The existence of an open and freely usable standard allows anyone in the world to experiment with extensions. Deployment of experimental features was enabled by one simple rule, inherited with care from the Internet email community:

Rule used to date:

Old rule: If you find a language element you don't understand, ignore it.

(The exact definition of "Ignore" varies - HTTP headers are actually ignored and HTML elements are replaced with their contents

(ie unknown tags are ignored) - but the principle has been the same.)

This rule has covered web development from 1989 to the present. The result has been a very high speed of growth. However, a state of ambiguity and lack of interoperability always exists from the introduction of an experimental feature until the later agreement on a common standard . This weakened the reliability and credibility of the Web. Furthermore, there has always been a threat that lack of consensus on new features would lead to a permanent fragmentation of the evolutionary paths.

The problem was that neither the specification of new elements nor the effect of ignoring them was ever clearly defined. Contrast this to the situation in most distributed object systems. In these cases, objects and support classes generally have well defined interfaces. Whilst ensuring interoperability, the rigidity of this system, in which new interfaces had to be explicitly agreed between parties, has been one of the factors inhibiting such systems from spreading in web-like or virus-like manner. As discussed in [Ascent]:

> And yet the ability to combine resources that were developed independently is an essential survival property of technology in a distributed information system.

Can we have the best of both worlds, and have clearly defined interfaces, but also allow systems from different communities to communicate when having only a partial understanding of each other's specifications? This need has surfaced from many areas from HTTP extensions (see the PEP requirements) to Metadata (see design notes onMetadata architecture).

Requirements

The need is for two systems to be able to communicate when they have a common vocabulary but not complete understanding of all the features they each use. As these requirements are derived from experience across many different systems, we will have to chose which words to use in this document.

Glossary

For the purposes of this document, words are used as follows:

element
: A range text within of a document, identified by a local identifier.

vocabulary
: a set of local identifiers in a document, (which identify parts of the document), and whose meaning (at some level) is defined by generic resource. The namespace resource conceptually represents the vocabulary in general, which may be represented by one or more schemata.

schema
: A specific document which defines a vocabulary (at some level)

Although this is a general document, it is hoped that these terms are not used inconsistently with their use in XML (element) and RDF (schema).

There is some rough correspondence in the soup of terms as follows.

This document	SGML	HTTP	Programming languages	RDF
Element	Element	Header	Function/Procedure/Method call	Element
Binding	-	(PEP header)	"Import", external declaration	
-	Entity declaration	-	#Include	
Declaration	Element declaration	(http spec)	Function declaration	
Schema	DTD	(none!)	Module interface definition	Schema
	Content model		Parameter type	
	Attributes		Parameter type	

Mixing vocabularies

When a message is sent across the Internet as part of a Web communications protocol, it is tempting as above to compare the message with a remote procedure call, and to adopt the characteristics of a procedure/method call from distributed OO systems. A procedure call identifies the target object, one of a finite number of methods from the exported interface, and a set of typed parameters.

However, this analogy is not powerful enough. A message should be considered an expression and, if one takes an analogy with programming languages, the analogy should be with an expression or program rather than with a function call. [Or, if considered a function call, strictly, the parameters have to be extended to allow other nested function calls]. In this case, there may be many functions identified, in many interfaces. In other words, don't think of an HTTP message or an HTML document as an RPC call, but rather as the transmission of a expression in some language.

In the case of an XML document, this corresponds to a document which contains elements whose declarations occur in many different specifications (SGML: many different DTDs). This is the requirement brought out under "Metadata architecture", of

mixing vocabularies:

It must be possible at one point in a document for more than one vocabulary to be in scope.

Scenario

Imagine that I send you an invoice for an aeroplane part I am shipping to you. The invoice is mostly in common business language, and the vocabulary such as item, cost, quantity, authorizing signature, total cost and due date are well known to both of us. However, the item is specified in an expression which details exactly which engine lower inspection hatch door mount bracket lock nut is involved. Neither you nor I actually have to understand this vocabulary and references to part numbers and the like. Only the person or machine loading the part onto the truck, and the person or machine installing the part in the aircraft need to know it. It is true that we need to agree about the cost and the significance of the signing authority, as that is part of the protocol between us.

This sort of thing happens all the time in real life. Documents mix vocabularies defined in different places. We are always making decisions about which of the myriad of things we don't understand are important to us. We are constantly handling information with partial understanding. Imagine if an old version of a word processor could read a file written by a new version with partial understanding, rather than panicing that it had met a being from the future. It also happens all the time on the web, as people bury private elements such as index tags and editing information inside HTML files.

The requirement is for the new vocabularies to be well defined, like the basic vocabulary.

By analogy with a programming language, a Web document or protocol message should be able to include expressions combining calls to functions from many modules. This is so fundamental to programming languages that it has gone without saying, but it has not been possible in SGML.

Same scope

What does "within the same scope" mean? It means that just nesting one sort of document inside another is not good enough. It means that I must be able to write an expression or compound element which combines elements from two vocabularies. (In fact, strictly, wherever there is an expression tree which combines identifiers from more than one vocabulary, one can in theory break it down to a set of nested subtrees each of which only uses one vocabulary and could be considered a "subdocument", but in practice this is impractically cumbersome.) For example, if I can extend HTML to include Math, in this way one is able to use HTML bold tags still within a Math expression.

Local scope

There is a practical requirement that it must be possible to introduce a new vocabulary in part of a document in a way that requires changes only locally within the document. This means that for example it must be possible to introduce a new vocabulary within a local block. Here is an example in an arbitrary syntax, where "NS:using" is the ***binding*** of local identifiers starting with "f" to a schema http://blah/currency

```
  <a:details>
   <NS:using href="http://blah/currency" as="f">
<a:price>
 <f:chf>4.00</f:chf>
</a:price>
   </NS:using>
  </a:details>
```

The binding between the local identifier and the schema is textually local. There is no need to a binding in the document's head. In general this makes document management much easier. It makes checking a document easier, as you can in some cases verify an embedded piece without having to check the whole document.

Why?

A specific need for local scoping comes from the fact that many documents are generated (for example by CGI scripts) by calling programs to output parts in context, and the program which generates the parts has no access to the rest of the document.

In theory it would always be possible to take such a document with nested bindings of namespaces, and find all those bindings, and generate new local prefixes for each so that they are unique, and then move all the bindings to the top of the document. Therefore, a document using local scope can be converted into one which only uses global scoping. However, this requires buffering of all the document, and so cannot be done in pipelined systems, and pipelined systems are often a necessity in the Web in order to achieve acceptable response times.

Another case involves very long documents using many namespaces. Typically web applications have to be able to cope with documents of arbitrary length. Imagine a document which, every paragraph, refers to a new name space. (A proof by example would be a document documenting many namespaces.. but image also a list of suppliers each of which has its own catalog schema.). As processing of the document continues, if the bindings of namespaces are local, then each is made and discarded. The working set needed for processing the document is finite. In the case in which the bindings are global in scope, then the working set size increases linearly with the length of the document, and the product of resource utilization and processing time then rises as the square of the document size.

A third example of a need for local scoping is that for many uses of XML (take SMIL for example) concatenation of two

documents to make one document should be a simple process. Indeed, a worthy design goal would be to require that the concatenation of any two XML documents be an XML document. If local scoping is not available, the concatenation function requires the rewriting of one document from beginning to end changing local identifiers where they clash.

In general, one can call on all the design experience of the computer science community which, over the years, has seen the need for block structured languages with local scoping. There have been many factors influencing this, but one unmentioned to date has been the maintainability of programs/documents. When the binding of a name and its use can be close together, for human-maintained documents, mistakes are much less likely.

It must be possible to introduce a new vocabulary in part of a document in a way that requires changes only locally within the document.

Lack of ambiguity

Some programming languages allow one to introduce identifiers from new name spaces in such a way that it is not possible to know which namespace a local identifier belongs to without accessing both the module interface specifications and checking which one has with the highest priority, or most recently in the document, redefined a given local identifier.

This may have some uses in a programming language such as Java[Java], but it has a serious flaw in that when one module changes (without the knowledge of the designers of the other module), it can unwittingly redefine a local identifier used by the second module, completely changing the meaning of a previously written document. Clearly, in the Web world in which modules evolve but documents must have clearly defined meanings, this is unacceptable. Contrast with Modula-3, where all names are either lexically scoped or fully qualified [SPwM3].

The syntax must unambiguously associate an identifier in a document with the related schema without requiring inspection of that or another schema.

This is the reason for the use of a prefix in the XML namespace proposal to tie the use of an identifier directly to the specification of the name space. Notice that in the example above, the fact that the binding element actually creates a new level of nesting removing all ambiguity.

Evolving new schema languages

In SGML the "DTD" defines, for an SGML element, what possible other elements may be nested inside it. For example, on an invoice, it may specify that the signing authority must be either Tom or Joe. It may specify that an item can be any part number or any accessory number or any book number. Checking the SGML validity of a document is a process which can be done automatically from the DTD. This is a check at a certain low level in that it does not verify semantic correctness, only structural correctness. But the structural constraints alone are useful in many ways. For example, a user interface for constructing a document can be generated automatically from the structural constraints.

We plan to introduce more powerful languages for describing not only the structure of a document, but the semantics to an extent that not only can checking be automated to a higher level, but also so can the processing of a document and reasoning about its contents be automated. Therefore it is essential that when a document is written to refer to a namespace, the name space definition should be a generic resource whose instances may include schemas in various languages at various levels of sophistication. This is an essential growth point for the web.

The resource defining a namespace may be generic and allow definitions of the namespace in varying present or future languages.

Correctness of documents with multiple vocabularies

How does one check the validity/correctness of a document with multiple namespaces? Clearly one must be able to find definitions of the namespaces at the appropriate level, and combine them. Looking at the example above of the invoice, we notice a difference.

In the case of the "content model" for an authorizing person, the designer of the invoice intended that in fact the schema should be extensible so that any new object could be included as an item. For example, one could use a part number system from any new supplier, just by incorporating the namespace. However, when it came to the "content model" for an authorizing person, only Tom or Joe should be able to sign. No namespace extension should be allowed to redefine the permissible content model

There must be a way of indicating when a given content model may be extended by new schemas.

There must be a way, in a new schema, of specifying that a given new content model is designed an extension to the existing content model of an existing schema.

These are constraints on the schema language. (They are addressed by the XML-DATA discussion NOTE.)

Granularity

With what granularity should one be able to define new vocabularies in XML? The analogy with programming languages suggest that we can understand how to add new elements (functions) but that adding new attributes to existing elements (parameters to

existing functions) is difficult to define when one gets above the structural level.

Although scheme languages do not yet exist to define semantic relations and typing, clearly there will be need for extension of concepts to type. Perhaps the need for content model extension will in fact represent the same need.

Incorporation into the language

The namespace functionality is a very fundamental part of the language. A language processor which does not understand it can check what in XML is called "well-formedness", ie basic syntactic correctness, of a document, but can do no more.

A fundamental processing need outlined above is "partial understanding". I envisage three ways in which partial understanding can be accomplished, when a document in an "original" schema's vocabulary includes some of a "new" schema's vocabulary:

1. It may be possible to mathematically deduce what information can be ignored from properties of the original schema;
2. At a simple level this could be built into the language itself so that it can be expressed in the document itself; (analogy with PEP extensions to HTTP).
3. The "new" schema may allow one to deduce what can be ignored. It may even give mappings which allow expressions in the new schema's vocabulary to be replaced with simpler expressions in better known vocabularies.

Notice that the first two ways do not require one to be able to access or understand the "new" schema in order to decide whether to ignore it. This is a powerful and important feature. Taking against the invoice example above, it is essential to be able to process the invoice at some level without even looking up on the Web any definition of the part numbers. It is sufficient for the invoice itself declare that the item specifications don't matter as far as the validity of the invoice as an invoice.

It should be possible to create an original document schema such that one can determine, without access to the extension schema, which uses of extensions to that document can be ignored.

The difference between the first two ways above is whether some functionality is regarded as basic to the language or part of a very commonly understood namespace of elements for document construction. This design decision is not currently clear.

Revision and evolution of namespaces

This document does not define the requirements of schema languages, nor of languages with which to assert the equivalence of assertions made using different vocabularies. However it is worth noting that the architecture expects machine-readable documents to describe the relationship between different schemas, including between a schema and later evolved versions of the schema. The namespace functionality itself is not required to address that issue directly.

References

[Ascent]
The Evolution of Web Documents: The Ascent of XML
Dan Connolly, Rohit Khare, and Adam Rifkin, W3J special Issue on XML, Vol 2, Number 4, Fall 1997, Pages 119-128

[Java]
The Java Language Specification, James Gosling, Bill Joy, Guy Steele, Edition 1.0, (Converted from the printed book, August 1996, first printing) esp. Section 6.5 *Determining the Meaning of a Name*

[SPwM3]
Systems Programming with Modula-3, November 1989. esp. Section 2.5 Modules and interfaces

[SMIL]
Synchronized Multimedia Integration Language W3C Working Draft 2-February-98 Philipp Hoschka

Bibliography

Type Modelling for Document Transformation in Structured Editing Systems
E. Akpotsui, V. Quint and C. Roisin. Mathematical and Computer Modelling, Volume 25, Number 4, Pages 1-19, 1997.

Theory of Semiotics
Umberto Eco Indiana Univ Press February 1979 ISBN: 0253202175

The electronic hypermedia encyclopædia: transcending the constraints of the "authoritative work"?
Patrick J. COPPOCK
The University of Trondheim
College of Arts and Science
Dept. of Applied Linguistics
N-7055 Dragvoll Norway
e-mail: patcop@alfa.avh.unit.no
e-mail: coppack@bo.nettuno.it

Authoritative Sources in a Hyperlinked Environment
IBM Research Report RJ 10076 (91892) May 29, 1997
Jon M. Kleinbert <kleinber@cs.cornell.edu>

The Component Object Model Specification
Draft Version 0.9, October 24, 1995 Microsoft Corporation and Digital Equipment Corporation. (esp Objects and Interfaces)

HTML Dialects: Internet Media and SGML Document Types
W3C Working Draft 06-Mar-96 Daniel W. Connolly

Access-Limited Logic: A Language for Knowledge Representation.
James Crawford. 1990. Doctoral dissertation, Department of Computer Sciences, University of Texas at Austin, Austin, Texas. UT Artificial Intelligence TR AI90-141, October 1990. (Algernon and Access-Limited Logic)