Names, Namespaces, XML Languages and XML Definition Languages

Henry S. **Thompson**

23 November 2005

Abstract

The construction of URIs to identify named constituents of arbitrary XML languages is explored, as a step towards managing the versioning of language definitions.

Table of Contents

1. Introduction

The problem of understanding and managing the versioning of XML languages is growing in importance. In this paper I try to make a start on what I take to be a pre-requisite for making progress on the versioning problem, namely identifying exactly what a version of a language is. This in turn means exploring what is need to document a language, and *this* turns in to a matter of clarifying just what names in a language are and what they name. Since XML languages are constituents of the Web, we approach the task by way of considering how to construct URIs for the named constituents of XML languages.

2. XML Namespaces: An evolving understanding

The recent discussion [http://lists.w3.org/Archives/Public/www-tag/2005Feb/0017.html] about whether the [xml:id] Recommendation 'changes' the XML namespace by 'adding' a new name to it helped clarify that what I'll call the *minimalist* reading of the [XML NS 1.1] Recommendation has achieved dominance in the intellectual marketplace. By "the minimalist reading" I mean the reading on which an XML namespace is primarily a syntactic mechanism for distinguishing one class of uses of a particular simple name from all other uses thereof. This means a namespace is just an infinite set of expanded names [1], *not* a finite set of names, nor a more complex structured object as suggested by the (in)famous now-deleted non-normative Appendix A: The Internal Structure of XML Namespaces [http://www.w3.org/TR/REC-xml-names/#Philosophy] of [XML NS]

The minimalist reading is the only one consistent with actual usage -- people mint new namespaces by simply *using* them in an expanded name or namespace declaration, without thereby incurring any obligation to define the boundaries of some set. You could say that a namespace springs into life the first time anyone uses a URI as a namespace name, but on balance I prefer an understanding which doesn't reify a namespace as such at all. I don't object to using phrases such as "[some name] in the [some URI] namespace", but that's just another was of saying "the expanded name `< some_URI, some_name >`".

On this account it makes sense to ask questions about namespace names, e.g. "What namespace name will XSLT 2.0 use?" and about expanded names, e.g. "Does the definition of the element type named `< http://www.w3.org/Style/1998/Transform, output >` change between XSLT 1.0 and XSLT 2.0?", but questions about namespaces as such are rarely if ever useful (unless of course they're understood as questions about namespace *names* or about some otherwise-defined set of expanded names with a namespace name in common).

3. From namespaces to languages

Taking the argument one step further, it is a necessary consequence of the position outlined above that it is incoherent to understand e.g. "Such-and-such an element type is defined in the XSLT namespace" to mean that the XSLT namespace contains element types (or element declarations). Considering things carefully, we must understand this sentence as meaning that the XSLT *language* uses the expanded name `< http://www.w3.org/Style/1998/Transform, such-and-such >` for some element type for some purpose. This perspective fits particularly well with a schema language such as W3C XML Schema in which a schema document for a particular target namespace corresponds to a schema which assigns element declarations, type definitions, etc. to expanded names all of whose namespace name is that target namespace, but any schema language which supports namespaces at all must in the end function to associate one or more definitions with a particular expanded name.

So it's *languages* (or as we used to say, *applications*, in the SGML sense) which provide definitions for expanded names. A language might provide one and only one definition for a particular expanded name, but evidently in many cases a particular expanded name may have more than one definition, because it gets used in the language in more than one sort of way, e.g. to name an element type *and* an attribute type. Note I'm using 'definition' here to cover

[1] I use this term to mean a pair of a namespace name (or nothing) and a local name, as defined by XML Namespaces 1.1 [http://www.w3.org/TR/xml-names11/#dt-expname])

everything a language has to say about a particular use of an expanded name -- syntax, semantics, whatever -- I'll come back to this below.

It's important to distinguish three uses of expanded names:for classes of infoitems in XML languages in general, and the use of expanded

1. As names for (classes of) infoitems in XML languages in general, without explicit formal definitions. Every well-formed XML document has named element types, even without a DTD or other form of schema, and many have attribute types, and some have named anchors.

2. As names for things in application data models in general. Even from the XML-is-for-human-documents perspective, there's an application data model distinct from the 'raw' infoitem model, and in most data models some of the constituents have names.

3. As names for things in the application domain of what we might call XML *definition* languages such as W3C XML Schema, Relax NG, OWL, SVG or WSDL. Such languages have as an important part of their overall semantics the assignment of names to constructs in their domain or data model, e.g. simple type definitions for W3C XML Schema, patterns for Relax NG, classes for OWL, views for SVG and interfaces for WSDL.

In what follows I'm going to focus on the somewhat convoluted case where the application domain is *itself* the definition of XML languages. Thus languages such as W3C XML Schema, Relax NG and WSDL are the primary focus hereafter.

4. Ambiguity

When a word has more than one meaning in a natural language, we say it's ambiguous. The same thing happens with respect to an XML language, that is, it uses the same expanded name for more than one thing. Ambiguous expanded names are a problem for Web Architecture, which says [http://www.w3.org/TR/webarch/#qname-mapping] "A specification in which QNames serve as resource identifiers MUST provide a mapping to URIs". One concrete goal of the analysis given here is, then, to try to address this problem.

Any given language only gives names to certain sorts of things. Some languages only give names to one sort of thing, while others give names to more than one sort of thing. For example, in valid XML, element types, attribute types, notations and two types of entity are all given names.

Although it's in principle possible, I'm not aware in practice of any XML definition languages which name only one kind of thing. Indeed very few XML languages of any kind, which name anything, name only one kind of thing. I haven't located any real examples, but it's easy to imagine one -- for example an AddressBookML, which would only provide for naming address book entries.

On the other hand some languages, although naming more than one sort of thing, constrain their use of names to be unambiguous. Typically this is manifest in that they use XML IDs in their XML representations, as for example [SVG] and [XEnc], but sometimes it is achieved by explicit requirement, as in [RDFS] and [OWL].

In the unambiguous cases, just an expanded name is sufficient to identify something, and constructing a URI for something is therefore straightforward (provided there's a functional mapping from namespace name to language), but in the in-principle ambiguouscases, where there are multiple sort of things being named, and no uniqueness constraint, this is not the case.

Looking more closely at XML as defined by a DTD, there are in principle an unbounded number of things which might share a name, only distinguishable by context: we have element declarations (max. one per expanded name), and attribute declarations (max. as many as there are element declarations). For example, there are four distinct definitions for **align** and five distinct definitions for **type** as attributes in the XHTML transitional DTD [http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd], and the name **style** is ambiguous between element and attribute in the same language. W3C XML Schema also has a substantial set of what it calls "symbol spaces" (what

I've been calling "sorts"). There are seven things whose definitions can be named (it adds types, attribute and element groups, notations and identity-constraints along side elements and attributes), it also allows elements as well as attributes to be defined in context. These examples of context-dependence mean that any general approach to providing unambiguous names will have to accommodate some means for distinguishing between contexts.

All of the above potential for ambiguity regarding expanded names is confined to a *single* language understood very narrowly. But of course not only do languages vary over time, as new versions of a language are released, but some languages encompass alternative variants which are all current at the same time. For example the HTML `P` element has a long and complex history, and even the XHTML `p` element has three distinct variants in version 1.0 (strict, transitional and basic), none of which is exactly the same as the one in version 1.1.

Sometimes we may want names which abstract over such differences, and other times we may need to be very precise.

None of this should come as a surprise. Ordinary language uses names in ways which are both ambiguous and context-determined, and whose use changes over time. But its consequence for the Web are more serious, particularly as we consider the use of names for things on the Web intended for automatic processing, where appeal to context for disambiguation may not be straightforward at all. At the very least it is clear that it is no longer trivial to specify an approach to constructing URIs for things which will cover all the cases just discussed.

5. Dimensionality

To summarise where we are so far, we've identified five dimensions which have to be fixed to identify a named thing: language (XHTML vs. XSLT), variant (1.0 traditional vs. 1.1), sort (Template rule vs. attribute set; context-dependency), namespace name (really distinct from language? See below), local name. The obvious approach to constructing a URI for resource which has a name within a language is to combine a URI for the language with a fragment identifier for the name. We've clearly got several too many dimensions to make this possible straight away.

Broadly speaking there are three ways one could respond to this:

1. Only expect to have a systematic approach to naming things with URIs when the URIs associated with a language are unique to it and the language has a single flat story about naming, that is, local names are always unambiguous, and you don't care about variants. We might call this the **simple** (or **simplistic**) view.

2. Demand a systematic approach in all cases, and over all variants, but acknowledge that this means that in complex cases (e.g. WSDL, XML Schema) the resulting URIs will themselves be complex, requiring new media types and/or using new XPointer schemes. We might call this the **rich** (or **overkill**) view, exemplified by [SCDs] and [WSDL_IRIs].

3. Look for a middle ground, which adopts the simple view wherever possible, otherwise an approximation to it which ignores variants and as much application-specific detail as possible, with the option to fall back to the rich view as and when this is necessary. We might call this the **middle** (or **80/20**) view.

6. Architectural principles

It's important to note that there's an unspoken common assumption to all three of these approaches to URI construction: We're going to construct the URI for some named thing by adding some variety of fragment identifier to the namespace name of its expanded name. There is no space here for the possibility that two distinct languages or language variants might use the *same* expanded name for two evidently distinct things.

This is intimately bound up with another assumption with respect to variation, namely that it's possibly to tell reliably when a change in something counts as a variation, as opposed to a fundamental change of identity. If I change the named definition of a type by nudging its min or max a bit, that pretty clearly just produces a variant of the same type.

But if I change the definition assigned to a name from being an integer to being a date, it's equally pretty clear that that's no longer the same type at all. Those are the easy cases, there will be many which are much harder to call.

I expect that both of these assumptions will want to be recast as Good Practice notes going forward (i.e. "Don't use the same expanded name for two different things of the same sort in different languages under your control"; "As a language evolves, use new expanded names for new things, don't recycle old ones").

7. Abstraction and friends

There are at least four things that "ignore variants" might mean:

- Pick a variant and stay with it. That is, the constructed URI names something in a distinguished variant, for example the first variant.
- Collect all variants. That is, the constructed URI names the set of things named across *all* variants in which the name is used.
- Abstract over all variants. That is, the constructed URI names whatever is common across over all the members of the above set.
- Accept that the name is contingent. This means accepting that the constructed URI will name different things at different times. It requires imposing an order, typically temporal, across all possible variants and then interpreting the URI to mean the largest member of the above set with respect to that order.

The last of the above alternatives is, of course, similar to the way most URIs already function. The resource identified by `http://www.guardian.co.uk/` is time-varying -- if you want a particular edition of the Guardian newspaper, you have to use a much more complex URI.

8. The starting point

What is the starting point for URI construction? Clearly if there is a one-to-one relation between language and namespace name (ignoring variants), then that's the starting point. What other cases are there?

1. There is no namespace name. [DocBook] and [SpecProd] are widely used languages for document markup which define elements and attributes in no namespace. The obvious choice of starting point in such cases is the URI of the official language definition.
2. There are multiple namespace names, all specific to the language. Many languages defined using W3C XML Schema are in this category, e.g. [UBL], [JDF], [ESIDEL]. Fortunately, in all but one pathological case ([FAndO]) there is a functional mapping from namespace name to language, so the namespace name is a usable starting point.

All this adds up to saying there is a single starting-point URI we can use for all names, whether the above story leads us to a namespace name or a language definition URI. Sometimes this URI will also encode some variant information, sometimes it won't. It would still be a good idea to have a single unchanging URI which names a language independent of variation, but that's for another discussion.

9. From rich to middle ground

We've already established that there are five dimensions along which the constituents of a language need to be identified: language, variant, sort, namespace name, local name. The previous section effectively covers language and namespace name, leaving variant, sort and local name. I will assume without argument that `http:` URIs are the goal. This gives us three syntactic mechanisms to exploit to produce a name from our starting point, which we'll schematise as `http://starting/point/`:

1. Additional path components, i.e. `http://starting/point/more/goes/here/`

2. Parameters, i.e. `http://starting/point/?more=this&other=that`

3. Fragment identifiers. This case sub-divides based on whether we use a new media type for the representations retrievable via our constructed URIs or not:

Existing XML media type(s)	Either a shorthand pointer, i.e. `http://starting/point/#ncname` or an XPointer using a new scheme, i.e. `http://starting/point/#more(goes,here)`
New media type(s)	Wide-open, only subject to `http:` syntax rules, i.e. `http://starting/point/#more;goes~here`

The rich approach needs to pick from the above mechanisms to encode all three of variant, sort and local name. The middle sometimes needs all three, sometimes variant or sort or both are not needed, but it always needs the local name. It follows that choosing a syntax which allows the encoding or variant or sort to be easily elided would be a good thing.

In cases such as XML attributes or elements whose identity is determined by context, the space of sorts is open-ended, so for the rich approach some form of path-based syntax seems inescapable.

Insofar as it makes sense to describe a generic solution, independent of the details of particular languages, then I think it looks like this:

variant	Encode as a numeric+optional alphabetic path component, e.g. `http://www.w3.org/1999/xhtml/1.1/`, `http://www.w3.org/1999/XSL/Transform/2.0/`
simple sorts	Encode as an alphabetic path component, e.g. `http://www.w3.org/2001/XMLSchema/simpleType/`, `http://www.w3.org/1999/XSL/Transform/attribute/`
local name	Encode as a shorthand fragment identifier, e.g. `http://www.w3.org/1999/xlink/#href`, `http://www.w3.org/2005/xpath-functions/#tokenize`
context-specific sorts	Encode as a fragment identifier using an XPointer scheme, existing if possible, otherwise new, e.g. `http://www.w3.org/1999/xhtml/#xpath(//hr/@align)` (the **align** attribute of the **hr** element in XHTML), `http://www.w3.org/2001/04/xmlenc/#xscd(/~EncryptedType/EncryptedMethod)` (the **EncryptedMethod** element as defined by the W3C XML Schema schema for the XML Encryption language.

The intention is that where necessary and/or appropriate, these can all be combined. When anything necessary to uniquely identify something is left out, the alternative interpretations discussed above at the end of Section 5, “Dimensionality” come in to play. Consider the case of the W3C XML Schema language itself. The expanded name `< http://www.w3.org/2001/XMLSchema, attribute >` has definitions therein as four different sorts of things: a key, a complex type, a top-level element type and an context-specific element type. Not all of these definitions stayed the same between the original Recommendation and the second edition. Accordingly we could establish a naming convention which yielded all the following URIs for things with that expanded name:

`http://www.w3.org/2001/XMLSchema/key/#attribute`	The key, no variant specified

`http://www.w3.org/2001/XMLSchema/1.0.2/complexType/#attribute`	The complex type, as defined in the second edition
`http://www.w3.org/2001/XMLSchema/1.0/element/#attribute`	The top-level element type, as defined in the first edition
`http://www.w3.org/2001/XMLSchema/csElement/#xscd(/group::attrDecls/attribute)`	The context-specific element type, as it appears in the `attrDecls` model group definition, no variant specified.
`http://www.w3.org/2001/XMLSchema/#attribute`	If we chose to, we *could* say this was the most recent top-level element. That is, for each dimension, it's open to us to say how to disambiguate in ambiguous cases.

10. A missing dimension

The detailed example in the previous section was looking at names for things defined in the W3C XML Schema language. But the things there were to name depended on the choice of W3C XML Schema as the definition language with which to define the target language. If we had used DTDs, or Relax NG, to define the target language, we would have had different sorts of things to name. For the moment, I'm assuming the right way to accommodate this dimension is just to consider it as just another source of variants. In other words, we consider the Relax-NG-defined language and the DTD-defined language to be variants, which might or might not be (provably) equivalent. It's for this reason that the examples above put the encoding of variant above the encoding of sort in the path, since the latter may depend on the former.

11. The resources identified

The proposal outlined above results in a large number of URIs for any given language -- perhaps as many as `(S x V) + S + V + 1` (that's 1 for the language, abstracting over variants and sorts, one for each sort, abstracting over variants, one for each variant, abstracting over sorts, and one for each sort with respect to each variant. What kind of resource should be identified by each such URI? Clearly, at least, one with anchors for all the barenames appropriate to the URI. And what should each such anchor correspond to? A definition of the thing named, of course. Presumably that should include connections to any published definitions, either formal or in natural language. But the details of how this should be done, and what further information should be provided, that is, the design of a generic language definition information document, although the original goal of this work, will have to be left for another day.

One thing we'd like to find in a definition is a set of variants with respect to which it's valid. This in turn would support a minimal coherence condition, given that the discussion above implies the existence of a partial (not all sorts exist in all variants, and not all names name things of all sorts) function from name plus sort plus variant to definitions. Informally, we would then hope that `defn(name,sort,variant)=defn implies variant in defn.variants`.

Acknowledgements

My thanks to the members of the W3C Technical Architecture Group and to Richard Tobin, who have contributed to my understanding of the issues addressed here in many patient conversations. The misunderstandings which remain are, of course, solely my own responsibility.

Bibliography

[DocBook] *DocBook [http://www.docbook.org/]*

[ESIDEL] *European Steel Industry Data Exchange Language [http://www.eurofer.org/edifer/publications_ESIDEL1_0.htm]*, Eurofer.

[FAndO] *XQuery 1.0 and XPath 2.0 Functions and Operators [http://www.w3.org/TR/2005/WD-xpath-functions-20050915/]*, Ashok Malhotra, Jim Melton and Norm Walsh, eds.

[JDF] *JDF Specification Version 1.3 [http://www.cip4.org/documents/jdf_specifications/JDF1.3.pdf]*, CIP4.

[OWL] *OWL Web Ontology Language Reference [http://www.w3.org/TR/owl-ref/]*, Mike Dean and Guus Schreiber, eds.

[RDFS] *RDF Vocabulary Description Language 1.0: RDF Schema [http://www.w3.org/TR/rdf-schema/]*, Dan Brickley and R. V. Guha, eds.

[SCDs] *XML Schema: Component Designators [http://www.w3.org/TR/2005/WD-xmlschema-ref-20050329/]*, Mary Holstege and Asir Vedamuthu, eds.

[SVG] *SVG [http://www.w3.org/TR/SVG/]*, Jon Ferraiolo, Jun Fujisawa and Dean Jackson, eds.

[SpecProd] *The XML Spec Schema and Stylesheets [http://www.w3.org/2002/xmlspec/]*, Norm Walsh and Eve Maler maintainers.

[UBL] *Universal Business Language 1.0 [http://docs.oasis-open.org/ubl/cd-UBL-1.0/]*, Bill Meadows and Lisa Seaburg, eds.

[WSDL_IRIs] *IRI References for WSDL 2.0 Components [http://www.w3.org/TR/wsdl20/#wsdl-iri-references]*, Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman and Sanjiva Weerawarana, eds.

[XEnc] *XML Encryption Syntax and Processing [http://www.w3.org/TR/xmlenc-core/]*, Donald Eastlake and Joseph Reagle, eds.

[XLink] *XML Linking Language (XLink) Version 1.0 [http://www.w3.org/TR/xlink/]*, Steve DeRose, Eve Maler and David Orchard, eds.

[XML NS] *XML Namespaces [http://www.w3.org/TR/REC-xml-names/], version 1.0*, Tim Bray, Dave Hollander and Andrew Layman, eds.

[XML NS 1.1] *XML Namespaces [http://www.w3.org/TR/xml-names11/] version 1.1*, Tim Bray, Dave Hollander, Andrew Layman and Richard Tobin, eds.

[xml:id] *xml:id [http://www.w3.org/TR/xml-id/]*, Jonathan Marsh, Daniel Veillard and Norm Walsh, eds.

Biography

Henry S. **Thompson**
University of Edinburgh
ICCS/HCRC, Division of Informatics [http://www.iccs.inf.ed.ac.uk/]
Edinburgh
United Kingdom
Markup Technology [http://www.markup.co.uk/]
World Wide Web Consortium [http://www.w3.org/]

Henry S. Thompson divides his time between the School of Informatics at the University of Edinburgh, where he is Reader in Artificial Intelligence and Cognitive Science, based in the Language Technology Group of the Human Communication Research Centre, and the World Wide Web Consortium (W3C), where he works in the XML Activity.

He received his Ph.D. in Linguistics from the University of California at Berkeley in 1980. His university education was divided between Linguistics and Computer Science, in which he holds an M.Sc. While still at Berkeley he was affiliated with the Natural Language Research Group at the Xerox Palo Alto Research Center, where he participated in the GUS and KRL projects. His research interests have ranged widely, including natural language parsing, speech recognition, machine translation evaluation, modelling human lexical access mechanisms, the fine structure of human-human dialogue, language resource creation and architectures for linguistic annotation. His current research is focussed on articulating and extending the architectures of XML.

He was a member of the SGML Working Group of the World Wide Web Consortium which designed XML, is the author of the XED, the first free XML instance editor and co-author of the LT XML toolkit and is currently a member of the XML Core and XML Schema Working Groups of the W3C, and has recently been elected to the W3C TAG (Technical Architecture Group). He is lead editor of the Structures part of the XML Schema W3C Recommendation, for which he co-wrote the first publicly available implementation, XSV. He has presented many papers and tutorials on SGML, DSSSL, XML, XSL and XML Schemas in both industrial and public settings over the last eight years.