Making the Infoset Extensible

Erik **Wilde** <net.dret@dret.net>

Abstract

The XML Infoset defines the data model of XML, and it is used by a number of other specifications, such as XML Schema, XPath, DOM, and SAX. Currently, the Infoset defines a fixed number of Information Items and their Properties, and the only widely accepted extension of the Infoset are the *Post Schema Validation Infoset (PSVI)* contributions of XML Schema. XML Schema demonstrates that extending the Infoset can be very useful, and the PSVI contributions of XML Schema are being used by XPath 2.0 to access type information in a document's Infoset.

In this paper, we present an approach to making the Infoset generically extensible by using the well-known Namespace mechanism. Using Namespaces, it is possible to define sets of additional Information Items and Properties which are extending the core Infoset (or other Infoset extensions, defining a possibly multi-level hierarchy of Infoset extensions). Basically, a Namespace for an Infoset extension contains a number of Information Items, which may have any number of Properties. It is also possible to define an Infoset extension containing only Properties, extending the Information Items of other Infosets.

Further elaborating on this method, many of the XML technologies currently using the Infoset could be extended to support the Infoset extensions by importing Infoset extension using the extension's Namespace name. To illustrate these concepts, we give an example by defining the *XML Linking Language (XLink)*, the XML vocabulary for hyperlinking information, in terms of Infoset extensions. We show how the proposed ways of supporting Infoset extensions in XML technologies such as XPath, DOM, and CSS could pave the path to a better support (and hopefully faster adoption) of XLink than we see today. XLink serves as one example, but the proposed extensions and techniques are not limited to this particular technology.

The content of this paper is work in progress, contributing to the ongoing debate on how to deal with different XML vocabularies and their usage in other XML technologies. We believe that making the Infoset extensible would provide a robust and flexible way of making the data model of XML-based data more versatile, and creating an accepted way of making the data available through standard interfaces such as DOM and XPath.

Table of Contents

1. Introduction 2
2. Motivation 3
3. The XML Infoset 3
4. Requirements 4
5. Infoset Extensions 4
    5.1. Modularization 4
    5.2. Datatypes 4
    5.3. Infoset Extension Support 5
6. Extension Examples 5
    6.1. XML Schema 5
    6.2. XLink 6
7. Opportunities for Using Extensions 6
    7.1. DOM EXIS Module 6
    7.2. XPath 2.0 7
    7.3. XHTML 2.0 7
8. Open Issues 8

9. Conclusions 8
Bibliography 8

1. Introduction

The *Extensible Markup Language (XML)* [XML1.0Sec] is the foundation of a number of technologies which are used by other technologies and applications. Even though XML itself was the language that started it all, the core of many XML applications is not XML itself, but the *XML Information Set (XML Infoset)* [XMLInfoset], the data model of XML. Examples for technologies built on top of the XML Infoset are the *Document Object Model (DOM)* [DOM3Core], and the *XML Path Language (XPath)* [XPath1.0], which in turn serves as a foundation for other technologies, for example the *XML Transformations (XSLT)* [XSLT1.0] programming language and the *XML Query (XQuery)* [XQuery1.0] language.

In all these applications of the XML Infoset, the Infoset is assumed to be identical to the model specified in the XML Infoset specification, even though the specification explicitly states that applications may subset or extend the Infoset if they want to. The first major application of this principle of Infoset extensibility is *XML Schema* [XMLSchema1] with its definition of *Post Schema Validation Infoset (PSVI) Contributions*. In essence, XML Schema defines the validation of a document as a process of Infoset augmentation. XML Schema thus cleverly uses the Infoset's extensibility, but looking at the current state of XML technologies, this also creates some problems, such as

- the inability to access the PSVI contributions in a standardized way from within applications, because of the DOM's limitation to the "core Infoset", and
- the necessity of other specifications using the PSVI contributions to have them "hardcoded" into the specification.

The goal of this paper is to describe a way how Infoset extensions could be specified generically and in a modular fashion, so that specifications or applications built on top of extended Infosets could be built in a standardized and re-usable way, and could use established technologies for dealing with extended Infosets, such as APIs and programming languages. In general, this paper should be one step in the direction of filling in the gaps which are still present in XML technologies as shown in table Table 1.

	Data Model	Interface(s)	Representation(s)
Documents	XML Infoset	DOM, SAX, JDOM	XML 1.0, Canonical XML
Schema Languages	DTD	n/a	XML 1.0
	Schema components	(Apache Schema component API)	XML Schema XML representation
	RELAX NG	n/a	RELAX NG XML syntax, RELAX NG compact syntax
Formatting	(CSS)	SAC, CSS-OM (DOM)	CSS
	(XSL-FO)	n/a	XSL-FO
Hyperlinking	(XLink)	n/a	XLink, HLink
HTML	HTML 4.01	HTML DOM	XHTML 1.0, XHTML 1.1
Selection	XPath Node Tree	XPath DOM, XSLT	XPath

Table 1. Status of some XML-related Technologies

The underlying idea is that of XML and the XML Infoset as the foundation, which provides applications with the ability to exchange and manipulate structured data in a very generic way, while certain specifications (such as XML Schema) may add additional semantics to this data (such as type information), which also should be accessible to applications. As a case study, we will present the *XML Linking Language (XLink)* [XLink1.0], which in a way is very similar to XML Schema, because it also augments the data of an XML document with additional information, in XLink's case with link information rather than type information.

2. Motivation

XML technologies are still an evolving field, and a particularly interesting one because the inputs and demands for the further development come from many different application areas and backgrounds. Even though this makes XML an exciting field, many people agree that there is a lack of coherence in the foundations for all these technologies, and it is our motivation to point out an area where a more clearly and generically defined model would be beneficial.

As an example of the current state of XML technologies it is interesting to look at XML and hyperlinking. After a long time of development, the *XML Linking Language (XLink)* [XLink1.0] has been defined as a W3C recommendation in June 2001, but so far support for it is very weak. There are two important reasons for this lack of support,

- the first is the lack of supporting standards, for example for programming with links (something like a DOM XLink module), link styling (link support in CSS and XSL-FO), and a clearly defined processing model for links;
- the second is the fact that XLink is defined in terms of syntax rather than a data model, and some possible applications of XML linking, such as XHTML, have problems with integrating XLink-style markup with their own philosophy or tradition of markup design.

This problem could have been avoided if XLink had been designed in a more abstract way, so that the first problem could be tackled more easily by applying well-known mappings of the Infoset to new mapping of an extended Infoset, and the second problem could have been dealt with by defining alternative syntaxes for the data model.

Of course, alternative syntaxes also introduce problems such as how to recognize them and how to avoid the uncontrolled spread of syntaxes for the same data model. Arguments about the pros and cons of what should come first, a data model or a syntax, very rapidly are very often fought with a lot of zeal und fundamentalism. The approach presented in this paper is meant as an argument that for the sake of a clearly layered architecture it may be a good thing to first define a data model and allow different syntaxes for it.

3. The XML Infoset

In this section, we give a very brief overview of the *XML Information Set (XML Infoset)* [XMLInfoset], the data model of XML. The Infoset is an abstract model of the content of an XML document, it is neither an API nor a representation. The Infoset is defined in terms of *items* having *properties*, and there are eleven types of items (document, element, attribute, processing instruction, ...). The Infoset also states that applications are free to extend or restrict that model, but it is unclear how exactly this may be done. In communications on the *xml-dev* mailing list, **Cowan**, one of the authors of the Infoset specification, stated that

- "there are only items and properties;
- every property has a type given, either 'set of items', 'list of items', or a simple type; and
- 'no value' is a special value, and 'unknown' is used when we are dealing with partial infosets."

However, all these facts are not clearly defined in the Infoset specification, as well as some more specific questions such as "is it possible to restrict the allowed item types in sets or lists of items", or "what exactly are 'simple types', which type system is supported"? Looking from today's perspective, the simple types may be XML Schema's simple or atomic types, but even if this is the case (which is just an assumption), are only XML Schema built-in types allowed, or is it possible to use any derived type? And if derivation is allowed, may it be done by restriction, list, and/or union? Thus, even though the Infoset serves a very useful and important purpose, it is severely underspecified when looking at it from the perspective of how to extend it.

4. Requirements

If we attempt to make the Infoset extensible in a clearly defined way, then the Infoset specification itself must be much more precise in terms of how to use it as a foundation for derived Infoset specifications. In particular, the Infoset specification must be extended in the following ways:

- Items and properties must be specified in a formal notation that makes it easy for Infoset extensions to reference them. The formal notation is also required to specify the types of properties.
- The "core" Infoset as well as Infoset extensions must have an identifier assigned that makes it possible to refer to this particular Infoset extension (or the core itself). In essence, this will enable a hierarchical structure of Infoset extensions built on each other, which can most easily be regarded as a dependency graph (consequently, it must be acyclic).
- Other specifications being based on the Infoset should be designed in a way which supports Infoset extensions. If not, Infoset extensions will not be accessible by these specifications.

These requirements will be described in more detail in the following section. We call the new version of the Infoset, which should be designed according to these requirements, the *Extensible XML Information Set (EXIS)*. It should be noted that the requirements as well as the concepts described in the following section are work in progress, so they may be incomplete or need some adjustment. However, we believe that the general idea is valid and should be implemented in the W3C standardization process.

5. Infoset Extensions

In the following sections, we describe in greater detail how EXIS Infoset extensions are designed. The three main areas (as already pointed out in the previous section) are modularization (described in Section Section 5.1), a datatype concept (described in Section Section 5.2), and support for EXIS Infoset extensions in specifications using the Infoset (described in Section Section 5.3).

5.1. Modularization

One of the main points of EXIS is that Infoset extensions should be regarded as *modules*, and that modules depend on other modules, so that a complete picture of a number of Infoset extensions will be a dependency graph (the root of which is the core Infoset itself). To make this possible, Infoset extensions must be identifiable, and because we essentially want to identify a set of names (items and properties), the mechanism which is mostly used for this (and therefore also employed by us) is *XML Namespaces* [XMLNamespaces].

The XML Namespaces recommendation does not make any assumption about the names being identified by a namespace name, different users of namespaces can use them differently, as demonstrated by the DTD-oriented *namespace partition* concept introduced in a non-normative part of the XML Namespaces recommendation, and by the *symbol space* concept introduced by XML Schema [XMLSchema1]. In the same way, EXIS namespaces have two partitions, one for items and another for properties. Each name must be unique in its partition.

When defining new Infoset extensions, it is necessary to specify which other extensions (or the core) this new extension is based on. Items in this new extension then may be defined and are within the namespace of this extension, while properties may be defined which either belong to items of the same extension, or to items that have been defined in extensions which serve as a foundation for the newly defined extension. If a property of an Infoset extension is defined for an item of another extension, then this extension can be uniquely identified by its namespace name, and the item by its name (relative to this namespace name). This Infoset extension then must be (directly or indirectly) a base for the newly defined extension.

5.2. Datatypes

If Infoset extensions should be openly definable, then there must be a set of types to choose from for the properties. Items are defined by the properties that they have, but properties often carry values that must be defined according to some type system. In principle, properties may have three different kinds of values:

- *Special values*: These are values such as 'no value' and 'unknown' used in the current Infoset specification, which indicate special cases which may occur for any property.
- *References to items*: Properties may be simple references to items, sequences of references to items, or sets of references to items. All three cases are used in the current Infoset specification, and these three variants of references to items should be sufficiently versatile for Infoset extensions.
- *Simple values*: Many properties will have simple values such as strings, numbers, or one of a number of predefined keywords. All these cases are used in the current Infoset specification, but it is unclear from which basic set of datatypes these have been chosen.

Looking at this list of values for properties, it seems as if the first and the last of them could be easily handled by using *XML Schema Datatypes* [XMLSchema2]. XML Schema's simple types provide an application-oriented set of datatypes which can be derived by restriction, list, or union. Moreover, references to items could also be modeled by using XML Schema-based types, but is is unclear whether this would include too much dependency into the EXIS architecture. However, XML Schema provides an interesting set of built-in data types (on the contrary, *RELAX NG* [RELAXNG] does not define such a set of built-in data types), and using this as the foundation for the datatypes to be built into EXIS probably will prove useful. This is also demonstrated by the fact that XML Schema datatypes are already part of the data model of XPath 2.0 (and thus XSLT 2.0 and XQuery 1.0, see Section Section 7.2 for an explanation), so that properties from Infoset extensions will fit in seamlessly with the existing data model.

5.3. Infoset Extension Support

The whole effort of making the Infoset extensible is pointless if there is no benefit to users. Modeling extensions of the XML data model as Infoset extensions can have some value in itself by forcing the extension author to make as many of his assumptions and constraints as explicit as possible, but the goal of EXIS is not to support modeling for the sake of modeling. Instead, EXIS is aimed at making life easier for specification authors as well as users, and therefore the concept of Infoset extensions has to be supported by other specifications as well.

The two most prominent examples where Infoset extensions would have to be taken into account are the two most popular tree models of XML, DOM and XPath. While DOM is mainly used as an interface from a large variety of programming languages, XPath is used as a foundation for a number of XML-specific technologies, most notably XSLT and XQuery. In both cases, the tree model exposes an XML document to an application (and ultimately to a user). In order to make EXIS immediately useful to users, Infoset extensions would have to be supported by DOM and XPath, thus exposing the extended data model through the tree model to the user. In the following section, we give some examples of where this could be of immediate benefit.

6. Extension Examples

While having a well-defined framework for modeling extensions of the XML data model is of value in itself, in this section we give some examples where existing XML technologies would have been improved by an underlying model for Infoset extensions.

6.1. XML Schema

So far, XML Schema is the only XML specification extending the XML Infoset. XML Schema does so by introducing a number of *Post Schema Validation Infoset (PSVI) Contributions*, which are defined in Section C.2 of the XML Schema Specification [XMLSchema1]. Interestingly, XML Schema depends very strongly on the PSVI model, because in XML Schema the whole process of validation is defined as a process of Infoset augmentation. Basically, the input for an XML Schema validator is an Infoset, and the output is the Infoset augmented with PSVI information.

Unfortunately, there is no defined interface for PSVI (the DOM activity within W3C once planned to create a DOM PSVI module, but so far nothing has been published). As a result of this situation, using an XML Schema processor is easy as long as one is only interested in a binary result (validation as a whole succeeded or failed), but when it comes to accessing the result of the validation process in a more detailed way, each XML processor has its own way of making them accessible to the user (if at all).

With EXIS, the PSVI information would have been modeled as an Infoset extension, and thus would have been accessible through any interface mechanism generically supporting EXIS (such as a hypothetical DOM EXIS module, described in more detail in Section Section 7.1). If the developer community decided that accessing PSVI through the generic interface was too cumbersome, it would still have been possible to define a dedicated DOM module, but this could have been made on demand and without any period of time where PSVI information was not accessible at all.

If we think of advanced XML programming as being based on a DOM pipeline model, than the current situation is that a DOM pipeline including an XML Schema processor can either (a) not use the PSVI information produced by the XML Schema processor, or is (b) depending on the particular way the XML Schema processor makes the PSVI information accessible and is therefore tied to this particular XML Schema processor. Both alternatives have drawbacks, and a DOM pipeline using an EXIS-enhanced DOM could benefit from providing access to the PSVI information while still being able to plug-in any other EXIS-enhanced XML Schema processor.

6.2. XLink

Another interesting example is the *XML Linking Language (XLink)* [XLink1.0], which is a vocabulary for embedding hyperlink information in XML documents. First approaches to define XLink based on a data model rather than syntax have been published, one in a W3C note by **Walsh** [XLinkStyle], as well as a more general approach [TIKrep148]. In both cases, XLink is defined in terms of Infoset items and properties, which comes very close to the notion of an Infoset extension as presented in this paper.

One of the more frustrating experiences for the hypermedia community has been the lack of support for XLink. Even though XLink has been around for some time now, it is not supported by any major software, and also enjoys only partly support in the W3C itself. Interestingly, the recent debate around the question whether *XHTML 2.0* [XHTML2.0] should support almost exclusively revolved around the fact that XLink syntax does not go very well with HTML-style syntax. While this is certainly true, the deeper and more fundamental question of whether XLink's linking model was appropriate or not was only rarely touched. In Section Section 7.3, we go into more detail about the question of XLink and XHTML 2.0.

If XLink were to be revised, a new version should define only the data model in terms of Infoset extensions, and leave open the syntactic representation to additional specifications. This would leave open the question of how XLinks are actually encoded ("How can XLinks be recognized?" is the question of heard when proposing this architecture), but rather than tying the data model to one particular representation (which may be disliked by some people, as the XLink vs. XHTML 2.0 debate proves), it would be wise to allow an additional degree of freedom.

Basically, all that the foundation for Web hyperlinking should do is define a linking model. It is very questionable whether purely syntactical aspects such as XLink's simple and extended links should be part of the linking model itself. To put it differently, the "new Web hyperlinking standard" could be something like *WLink (Web Hyperlinking)*, and XLink would be just one syntax for it, based on XML and a specific set of objectives (such as providing a simpler syntax and a more elaborated one). From an application point of view, it would be unimportant whether a particular link structure had been created based on XLink markup, on HTML markup, or on some completely different method, as long as the resulting link structure was compliant with WLink.

7. Opportunities for Using Extensions

In this section, we describe some scenarios demonstrating how EXIS could be used to make life with XML technologies easier. This may not be the case for all XML users, but it will certainly appeal to users having additional data models on top of XML, which up to now are forced to handle these additional data models in a completely proprietary way.

7.1. DOM EXIS Module

The *Document Object Model (DOM)* and its current variant *DOM Level 3* [DOM3Core] is one of the more successful Web standards. It provides the foundation for many applications dealing with structured documents (either HTML or XML). DOM3 defines a rather complex module structure, with dependencies between a number of modules. This modular approach makes DOM more manageable, because most DOM applications do not require all parts of DOM.

Starting from this existing module structure of DOM, it would be easy to think about a DOM EXIS module, providing generic access to extended Infosets. Basically, a programmer would have to specify an Infoset extension's namespace name, and, if it supported this Infoset extension, the DOM EXIS module would provide access to the respective Infoset extensions. A cleverly designed XML parser could even be configurable with respect to Infoset extensions, so that for example it would be possible to configure the parser how to harvest XLink Infoset extensions from XLink syntax (or maybe XLink Infoset extensions from XHTML syntax).

The main point is that a DOM module for generic access to EXIS information would make it possible to work with extended data models without the need to have a newly designed DOM module. For example, to work with XLink using DOM, one currently needs to implement a proprietary "XLink module" sitting on top of the regular DOM and collecting XLink information by searching for attributes being in the XLink namespace. This way, the expenses of transforming XLink syntax to XLink information have to be carried by everybody wanting to work with XLink.

7.2. XPath 2.0

In the same way as DOM provides access to Infosets in many programming languages, XPath provides access to Infosets in some XML technologies, most notably XSLT and XQuery. The *XQuery 1.0 and XPath 2.0 Data Model* [XPath2.0DataModel] defines the data model for the latest version of XPath (in *XPath 1.0* [XPath1.0], the data model was an integral part of the XPath standard itself).

XPath syntax is very compact and powerful, but unfortunately it is not very consistently designed. For example, axes are used to identify positional properties of nodes (for example, the `following-sibling` axis) as well as node types (with the `attribute` axis). Names of nodes are referenced in different ways, either as name-only node tests (for elements and attributes), or as parameters of node type node tests (for `processing-instruction` nodes). This makes it hard to extend XPath in a logical way to cover extensions of the underlying Infoset. However, we believe that *extension axes* could be one solution to the problem of how to access Infoset extensions in XPath. In the same way as XPath allows *extension functions* to be accessed using their namespace-prefixed names, it would be a logical extension of XPath to allow *extension axes*. Another way to go would be to extend the *node test* to also support extension nodes (i.e., items). The exact mechanics of how to extend XPath to cover Infoset extensions have to be examined in more detail, but the benefits are obvious.

If XPath supported EXIS, it would be possible to write an XSLT program to handle hyperlinks in documents. For example, it would be possible to include a statement like `select="//xlink:link::*"` (or maybe `select="/descendant::xlink:link()"`), selecting all links within the document and then doing something with them. Using this approach, it would be possible to use the high-level language of XPath (which is a very powerful way for selecting certain parts of an XML Infoset) and to make it immediately useful for data models which go beyond XML's core model.

7.3. XHTML 2.0

The latest version of XHTML, which is *XHTML 2.0* [XHTML2.0], is currently under development. There was a very long (and heated) debate over whether XHTML should start to use XLink syntax. The biggest problems are that XLink's linking model is not sufficient (because XHTML has more sophisticated linking semantics), and that XLink's syntax does not fit into XHTML's markup design. The latest solution to this problem is *HLink* [HLink] (even though it is not clear whether this will be used eventually). HLink basically is a mapping mechanism making it possible to use some of XHTML's established attribute names, and then taking their actual values from other attributes (which may adhere to xlink syntax).

The problem of the XHTML working group illustrates the problem of XLink: it mixes data model and syntax, makes them inseparable, and also does not provide any way to built on it. Using a more modular approach, XHTML might have used the XLink data model (or maybe extend it with things that are XHTML requirements not satisfied by basic XLink), and then define its own syntax for it, thus remaining backwards compatible with older HTML dialects.

On a more personal note: On the scale between either redesigning XLink (which basically is what EXIS tries to achieve) or making XHTML XLink-compatible (which would make it more different from legacy HTML), HLink is probably not a very good solution. It may solve the problem of casting XLink into XHTML-compatible markup in this particular case, but it produces rather ugly markup and is not XLink anyway (because it extends XLink's semantics).

8. Open Issues

This paper describes work in progress. As such, there are lots of open issues and open questions, and there will certainly be more once some of the issues below are tackled. We believe that the work invested in an more modular data model supported by a number of technologies would be rewarded with the opportunity for more software re-use and a cleaner approach to XML data models. However, to make this happen, the following questions have to be answered:

- *Specification of EXIS*: How is a EXIS module specified? It consists of items and/or properties and depends on one or more other modules, so how is that information encoded? It could be done completely informally as the current Infoset specification, or it could be done more formally, for example using a schema for authoring XML documents specifying EXIS modules. The most important part probably would be the datatype information about the properties, because this information is necessary to access actual property values through interface technologies such as DOM and XPath.
- *Datatypes*: At the moment it seems that XML Schema is the most promising candidate for the datatype vocabulary for properties. The *Structures* part of XML Schema is sometimes criticized because of its complexity and its lack of a formal foundation, but the *Datatypes* part of XML Schema seems to enjoy more credibility. However, in order to be really useful, it would be necessary to use type derivation (which is defined in the *Structures* part), so the question is whether this would be a potential acceptance problem.
- *80/20 split*: One question that always looms is "Is it worth the effort?" In the case of EXIS, the answer very much depends on the application area. If some user simply needs to handle elements and attributes, then EXIS certainly is not worth the effort. However, as XML is being used for increasingly complex tasks and data, users that need to go beyond the rather simple XML core data model will certainly be happy to have some built-in support for their needs into important building blocks of XML application scenarios such as DOM, XSLT, and XQuery.

9. Conclusions

In this paper we have presented an approach to make the XML Information Set extensible. The approach presented is work in progress, and will most likely change in one aspect or the other. However, we believe that a lot of benefits could be realized with such an approach, the most important being the opportunity to handle XML and extended XML data models in a more systematic way. It will certainly be a big challenge to have an approach like this being accepted by the XML community, because it reaches deep into the foundation of a number of XML technologies. However, one of the advantages of the approach is that it will be fully backwards compatible, because it does not change anything of XML's core data model. Our work on EXIS is continuing, and any feedback regarding the work presented in this paper is very welcome.

Bibliography

[CSS3Intro] **Eric A. Meyer** and **Bert Bos**. *CSS3 Introduction*. World Wide Web Consortium, Working Draft WD-css3-roadmap-20010523, May 2001.

[DOM3Core] **Arnaud Le Hors**, **Philippe Le Hégaret**, **Lauren Wood**, **Gavin Thomas Nicol**, **Jonathan Robie**, **Mike Champion**, and **Steven Byrne**. *Document Object Model (DOM) Level 3 Core Specification*. World Wide Web Consortium, Working Draft WD-DOM-Level-3-Core-20020409, April 2002.

[HLink] **Steven Pemberton** and **Masayasu Ishikawa**. *HLink: Link recognition for the XHTML Family*. World Wide Web Consortium, Working Draft WD-hlink-20020913, September 2002.

[RELAXNG] **James Clark**. *RELAX NG Specification*. Organization for the Advancement of Structured Information Standards, Committee Specification, December 2001.

[TIKrep148] **Erik Wilde**. *A Proposal for XLink Infoset Contributions*. Technical Report TIK-Report No. 148, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, August 2002.

[XHTML2.0] **Shane McCarron**, **Jonny Axelsson**, **Beth Epperson**, **Ann Navarro**, and **Steven Pemberton**. *XHTML 2.0*. World Wide Web Consortium, Working Draft WD-xhtml2-20020805, August 2002.

[XLink1.0] **Steven J. DeRose**, **Eve Maler**, and **David Orchard**. *XML Linking Language (XLink) Version 1.0*. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[XLinkStyle] **Norman Walsh**. *XML Linking and Style*. World Wide Web Consortium, Note NOTE-xml-link-style-20010605, June 2001.

[XML1.0Sec] **Tim Bray**, **Jean Paoli**, **C. M. Sperberg-McQueen**, and **Eve Maler**. *Extensible Markup Language (XML) 1.0 (Second Edition)*. World Wide Web Consortium, Recommendation REC-xml-20001006, October 2000.

[XMLInfoset] **John Cowan** and **Richard Tobin**. *XML Information Set*. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[XMLNamespaces] **Tim Bray**, **Dave Hollander**, and **Andrew Layman**. *Namespaces in XML*. World Wide Web Consortium, Recommendation REC-xml-names-19990114, January 1999.

[XMLSchema1] **Henry S. Thompson**, **David Beech**, **Murray Maloney**, and **Noah Mendelsohn**. *XML Schema Part 1: Structures*. World Wide Web Consortium, Recommendation REC-xmlschema-1-20010502, May 2001.

[XMLSchema2] **Paul V. Biron** and **Ashok Malhotra**. *XML Schema Part 2: Datatypes*. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502, May 2001.

[XPath1.0] **James Clark** and **Steven J. DeRose**. *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[XPath2.0] **Anders Berglund**, **Scott Boag**, **Don Chamberlin**, **Mary F. Fernández**, **Michael Kay**, **Jonathan Robie**, and **Jérôme Siméon**. *XML Path Language (XPath) 2.0*. World Wide Web Consortium, Working Draft WD-xpath20-20020816, August 2002.

[XPath2.0DataModel] **Mary F. Fernández**, **Jonathan Marsh**, and **Marton Nagy**. *XQuery 1.0 and XPath 2.0 Data Model*. World Wide Web Consortium, Working Draft WD-query-datamodel-20020816, August 2002.

[XQuery1.0] **Scott Boag**, **Don Chamberlin**, **Mary F. Fernández**, **Daniela Florescu**, **Jonathan Robie**, and **Jérôme Siméon**. *XQuery 1.0: An XML Query Language*. World Wide Web Consortium, Working Draft WD-xquery-20020816, August 2002.

[XSLT1.0] **James Clark**. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium, Recommendation REC-xslt-19991116, November 1999.

[XSLT2.0] **Michael Kay**. *XSL Transformations (XSLT) Version 2.0*. World Wide Web Consortium, Working Draft WD-xslt20-20020816, August 2002.