Architectural Specifications for the World Wide Web and their Role for Language Resources

L. Quin, F. Sasaki, C.M. Sperberg-McQueen, H.S. Thompson

W3C/MIT, W3C/Keio, W3C/MIT, W3C/ERCIM
liam@w3.org, fsasaki@w3.org, cmsmcq@acm.org, ht@inf.ed.ac.uk

Abstract

This paper describes specifications which have been (or are being) developed within the Architecture Domain of the World Wide Web Consortium. This Domain is responsible for many of the core technologies for the World Wide Web, including XML. We will describe XML-related technologies in five areas: validation, full-text analysis, declarative descriptions of XML processing, layout, and Internationalization, focusing on how they are particularly suited for the representation and processing of language resources. The paper also includes a broad overview of the standardization process which underlies the development of these and other W3C technologies.

1. Introduction: A Brief Overview of W3C and its Process

W3C[1] is an international consortium with the mission to develop Web standards, with contributions from W3C member organizations, the W3C staff, and the public.

W3C is working on a technology stack informally described at <http://www.w3.org/Consortium/techstack-desc.html>. The Work is organized in Activities like the XML Activity or the Internationalization Activity, which are parts of domains (Architecture, Interaction, Technology and Society, Ubiquitous Web, Web Accessibility). Work items are described in charters for Working Groups, Interest Groups, or Incubators. The difference between these is their scope. Working Groups and Interest Groups concentrate on royalty-free specifications for Web technologies ("Recommendations") and guidelines for their use ("Best Practices"); Incubator Groups concentrate on other, experimental topics which may be input to standardization efforts in the future.

In addition to W3C Activities there are the W3C Advisory Board and the Technical Architecture Group (TAG)[2]. The former provides guidance about management, legal matters etc., whereas the latter helps to build consensus on fundamental principles of Web Architecture (Jacobs and Walsh, 2004).

W3C as an organization is formally attached to three hosts: the Massachusetts Institute of Technology (MIT) in the USA, the European Research Consortium for Informatics and Mathematics (ERCIM) in France, and Keio University in Japan. In addition there are W3C offices in many countries to promote adoption of W3C technologies. The W3C membership can propose and drive new work items, has early access to new materials, and uses W3C as a community platform to decide new technology directions.

The development of royalty-free specifications relies on the W3C process (Jacobs, 2005) and the W3C patent policy (Weitzner, 2004). The latter describes licensing and patent disclosure requirements for the participation in W3C work and is an important factor for many organizations to decide about their engagement in W3C.

Two key aspects of W3C Recommendation development are that we aim to reach consensus, within the W3C and the public, about the actual features of new technologies, and it is required to demonstrate several interoperable implementations before publishing a Recommendation. The need for consensus sometimes slows the development process for a Recommendation, but it significantly increases the likelihood that the result will actually be accepted, implemented, and deployed in the community. An example is the XML Query language XQuery 1.0. Between the publication of the first public draft and the final publication of the Recommendation, about six years elapsed. However, during that period a large developer and user community took shape, and the Recommendation was published with around 40 implementations.

2. XML Validation: XSD 1.1

XSD, the XML Schema Definition language, is a metalanguage for defining XML vocabularies. Essentially, the author of an XSD schema provides a (regular right-part) document grammar for documents which use the vocabulary; unlike some other XML schema languages, XSD makes first-class citizens out of the *types* associated with elements and attribute. Types may be defined by restricting or extending other types, so that the class hierarchies usual in object-oriented design have a relatively natural representation in XSD schemas. A typical schema consists primarily of the definition of simple and complex types and the association of elements and attributes with types.

A number of primitive datatypes (or 'simple types') are provided: strings, booleans, decimal numbers, floating-point numbers, date-time stamps of varying precision (date-time, date alone, year, year plus month, time alone, etc.), URIs, and some others. From these, a number of other built-in types are derived by restriction, including integers and various subtypes of integer (long, short, byte, positiveInteger, etc.).

Modularization facilities are also provided for using several vocabularies in conjunction. In the usual case, one or more

[1] A general introduction to W3C can be found at <http://www.w3.org/Consortium/about-w3c.html>.

[2] See <http://www.w3.org/2002/ab/> for a description of the Advisory Board; for the TAG, see <http://www.w3.org/2001/tag/>.

schema documents are used to define the vocabulary associated with a given namespace (see (Bray et al., 2006)), and a schema is constructed by consulting one or more schema documents. The schema thus constructed will often consist of components from several namespaces.

Because they are essentially document grammars with a few additional constraint mechanisms, schemas can vary in many of the same ways that grammars can vary. They can be tight or loose, over- or under-generate vis-a-vis some given body of material. In XSD, they can also be incomplete: wildcards and 'lax validation' can be used to define constraints on some elements and attributes, while leaving others undefined and unconstrained. Schemas are often used for validating XML data as a quality assurance measure, to detect typographic and tagging errors. They are very useful in this role, but schemas can also be used in other ways. Data binding tools read schemas and generate object-oriented code to read documents which conform to the schema and from them create objects of particular classes, or serialize objects of particular classes in XML which conforms to the schema.

Schemas can document the contract between data sources and data recipients: the source typically undertakes to provide only valid data, and the recipient undertakes to accept any and all valid data. In such scenarios, invalid documents are often simply rejected. In other cases, schemas can be used to document a particular understanding of a kind of document, capturing a simple view of the 'standard' realization of the document type without allowing for all variations. In dictionaries, for example, it is a commonplace observation that there are important regularities among entries, but that a small number of entries require rather unusual structures. Grammars which allow for all of the structural variations actually encountered in a dictionary often provide no very clear account at all of the regularities which apply in 99% or more of all cases. Grammars which capture the regularities clearly often do not accommodate the deviant structures which can appear in a small number of cases. (See (Birnbaum and Mundie, 1999) for fuller discussion.) In part to assist in handling such situations, XSD defines validity not solely as a Boolean property of documents, but describes validation as providing a much richer result: each element and each attribute is individually labeled as to validity or partial validity. XSD can thus be used either for conventional prescriptive grammars or for descriptive grammars which focus on capturing the salient regularities of the material; material with the typical structures described by the grammar can be handled in one process, while anomalous structures can be detected automatically (by their failure to be valid against the schema) and handled specially. XSD does not require that applications reject documents which are invalid or only partially valid.

XSD 1.1 (see (Gao et al., 2007) and (Peterson et al., 2006)) offers a number of enhancements to XSD 1.0, most visibly the addition of

- assertions
- conditional type assignment
- open content

Assertions allow the schema author to express constraints using XPath expressions: the assertions associated with any type are evaluated for each instance of the type, and if any assertion fails to evaluate to true, the instance is not valid against the type. The most common use of assertions will be to formulate co-occurrence constraints. When declaring an element with integer-valued attributes named *min* and *max*, for example, a schema author might wish to specify that the value of the one should be less than the value of the other. This is easily accomplished with an appropriate assertion:

```
<xs:assert test="@min le @max"/>
```

Some XML vocabularies specify two or more attributes for a given element with the proviso that at most one of them may occur. This can also be handled conveniently with assertions. To ensure that either attribute *a* or attribute *b* may appear, but not both, one might write:

```
<xs:assert test="not(@a and @b)"/>
```

To require additionally that at least one of the two must appear, one might write:

```
<xs:assert test="
  (@a or @b)
  and not(@a and @b)"/>
```

The assertions of XSD 1.1 are restricted in one important way: they can refer to attributes or descendants of the element being validated, but they cannot refer to its ancestors, to its siblings, or to any elements or attributes outside the element itself. This restriction helps preserve the design invariant that the validity of an element or attribute against a given type can be tested in isolation from the rest of the document. This provides a certain context-independence of type validity, which is useful in transformation contexts like XSLT or XQuery.

Another way to capture co-occurrence constraints is to make the assignment of a given type to an element depend upon conditions to be checked in the instance. XSD 1.1 provides a form of such *conditional type assignment* based on (Marinelli et al., 2004). Here, too, the conditions which govern type assignment are given in the form of XPath expressions. To specify, for example, that the type assigned to a *message* element depends upon its *kind* attribute, given appropriate definitions of *messageType*, *string-message*, *base64-message*, *binary-message*, and *xml-message*, one might write:

```
<xs:element name="message"
    type="messageType">
  <xs:alternative
      test="@kind='string'"
      type="string-message"/>
  <xs:alternative
      test="@kind='base64'"
      type="base64-message"/>
  <xs:alternative
      test="@kind='binary'"
      type="binary-message"/>
```

```
  <xs:alternative
      test="@kind='xml'"
      type="xml-message"/>
  <xs:alternative
      test="@kind='XML'"
      type="xml-message"/>
</xs:element>
```

The third major change in XSD 1.1 to be discussed here is the provision of methods for specifying what is sometimes called 'open content'. In defining a document grammar, one may wish to specify, for example, that a particular element must have an *a*, a *b*, and a *c* among its children, in that order, without forbidding other material to appear before, after, or between these required elements. This can be done in XSD 1.0 with judicious use of wildcards, but experience shows that this method is error-prone and apt to fail for uninteresting technical reasons.[3] XSD 1.1 allows the schema author to specify (on a case-by-case basis) that types have open content; the schema author can specify a wildcard which is notionally inserted everywhere in the content model, or allowed only at the end of the model. The result is that it is much easier using XSD 1.1 to specify vocabularies which allow arbitrary extension by others, and which can accept new material in new versions of the vocabulary without breaking existing infrastructure keyed to earlier versions of the vocabulary. If version 1.0 of the definition of a vocabulary specifies open content everywhere, then any new elements added in later versions will be accepted by 1.0 processors without difficulty (albeit also without any knowledge of their meaning).

For those charged with developing or maintaining language resources, schema languages offer numerous opportunities for finding errors in the XML transcription of material, or distinguishing material with standard, straight-forward structure from material with anomous structures, and for describing explicitly the class of documents which certain processes are allowed to produce and other processes are required to consume without error.

3. Multi-document XML Validation: SML 1.1

SML, the so-called 'service modeling language', is an additional validation technology built on and layered on top of XSD. See (Pandit et al., 2008).

SML was originally developed for checking the validity of models intended to describe complex sets of information-technology services; the name "Service Modeling Language" thus reflects the historical origin of the technology, but in the meantime the name has become a misnomer: SML is a generic mechanism for validation across document boundaries, and has nothing in particular to do with services or their modeling.

An SML model is a set of XML documents, some of them are *model instance documents*, which contain representations of the information being modeled, and others are *definition documents* which define schemas to be used when validating the model instance documents.

In addition to requiring XSD validation of the instance documents, SML provides several additional mechanisms for specifying constraints which can be expressed only awkwardly in XSD, or not at all.

Schematron assertions can be associated with element declarations and type definitions; the asertions are checked for each instance of the element declaration or of the type definition.

The central innovation of SML, however, is its definition of a way to validate references from one document to another. This has a number of aspects.

First, the set of validatable links is not assumed identical to the set of links in the documents: the instance documents may well contain hyperlinks which are not constrained by the SML model and need not be validated. Those inter-document links which *are* to be validated are "SML references", indicated by the presence of the attribute-value pair `sml:ref='true'` (or its equivalent) on the element which constitutes the reference.[4]

The actual form of the link is not constrained: any systematically defined method of pointing from one XML document to an element in another XML document (all targets of SML references must be elements) may be used. Indeed, a single reference may refer to the the target element in multiple ways, each suitable for a different deployment scenario. Of course, SML processors will understand only a particular set of reference schemes; in the interests of interoperability, SML defines one schema, the SML URI reference scheme, which uses URIs to address the target of the link. XPath 1.0 expressions are as fragment identifiers, and XPath 1.0 is augmented by a `deref()` function to allow SML references to be followed. SML processors may support any reference schemes they choose, but all are required to support the SML URI reference scheme.

Having ensured that the set of links to be validated can be reliably and easily determined, SML then allows various constraints to be imposed on links.

- The `targetRequired` constraint requires that the reference resolve to an element in a document within the SML model.
- The `targetElement` constraint requires that if the reference resolves, then it must resolve to an element which bears a particular element name or "generic identifier". (A figure reference in a conventional textual hyperlink might be required, for example, to point to a *figure* element). Other elements declared in the schema as substitutable for the named element may, of course, be substituted.

[3] XSD requires that content models (i.e. the right-hand sides of production rules in the document grammar) be 'deterministic', i.e. that they not require lookahead. Whenever a wildcard is placed between elements *a* and *b* in the content model, if that wildcard also matches elements named *b*, the result is likely to be a violation of the determinism rule. The content model can normally be rewritten to avoid the problem, but this often proves tedious.

[4] As of this writing, SML references are required to be elements; this can make the current version of SML unsuitable for describing existing vocabularies in which a single element has multiple hyperlinks to other locations, expressed for example by different attributes.

- The `targetType` constraint requires that if the reference resolves, then it must resolve to an element governed by a particular XSD type, or by some type substitutable for it.

- The `acyclic` constraint specifies that the references bound to a particular declaration must not form a cycle. If a catalog of university courses, for example, uses a particular form of hyperlink to refer from a course to its pre-requisite courses, then the pre-requisite link should be checked to make sure that no course it (transitively) among its own pre-requisites.

Language resources sometimes are stored in single large documents, and sometimes in many smaller documents, and the choice between monolithic or fragmented representations often depends heavily on external factors rather than upon any logic intrinsic to the material. It is convenient, in such situations, to allow the material to be realized in either form, without losing the ability to validate it. SML's ability to validate across XML document boundaries is a useful way to ensure that relations within the data can be validated whether in a single document or in many.

4. XML Analysis: "XQuery 1.0 and XPath 2.0 Full-Text 1.0"

"XQuery 1.0 and XPath 2.0 Full-Text 1.0" (Amer-Yahia et al., 2007) is a specification that defines full-text search capabilities. It provides various full-text expressions and options to be used from within XQuery 1.0 or XPath 2.0 expressions. The options relate to stemming, thesauri, the use of stop words, and so forth, as well as to distances (for example "within five words") and units (words, sentences, paragraphs). They also support ranking and relative weighting of sub-expressions.

The specification does not dictate specific algorithms for full-text search implementations, but instead describes only the results of operations. As a consequence, one must expect some variation between implementations. From the point of view of linguistic research, this variation means that it is important to determine, whether through documentation or experimentation, the exact facilities provided by any given implementation.

The tokenization algorithm splits the input data into a sequence of tokens, which, conceptually, are then indexed; the same tokenizer is used to parse queries at run-time into sequences of tokens to be matched against the index. The tokenizer is expected to recognise *xml:lang* attributes and to perform multilingual matching as necessary.

At the time of writing, this specification is a Last Call Working Draft; a formal call for implementations is expected in May of 2008, and so although there are already some implementations, there may be changes in the final specification as a result of implementation experience.

Probably the biggest limitation of the current Full Text draft for language research is the lack of introspection: one cannot find out exactly which token or tokens matched the query, and one cannot directly implement match highlighting in the way one might want for a concordance or keyword-in-context index. Some implementations do provide a way to do this, and a future version of the specification may well standardize it, but for now it represents a severe limitation.

The limitation is greatly ameliorated when one considers that XQuery (like XPath 2.0 itself) operates not only on XML files, but on any data that can be represented as XPath and XQuery Data Model (XDM) instances. This includes for example geospatial data, relational data, RDF, and more. As a result, one can perform joins across different types of database, correlating them with efficient text searching. In summary, "XQuery 1.0 and XPath 2.0 Full-Text 1.0" (Amer-Yahia et al., 2007) will be a valuable tool to researchers, including the language resources community.

5. XML Processing: "XProc: An XML Pipeline Language"

Using XML to represent language resources has become the norm. Actually processing language resources for some purpose often consists of a sequence of processing steps which split, merge, restructure, and transform XML. "XProc: An XML Pipeline Language" (Walsh et al., 2007) is a specification which provides an XML vocabulary for specifying just such sequences, together with an inventory of both simple structural manipulations such as renaming, wrapping, deleting, and extracting items in an XML data stream, as well as larger-scale standards-based operations such as validation, transformation, and querying (see above).

Many kinds of XML technology and standards, including XSLT, XML Schema, XInclude, XQuery, and even SOAP and WSDL, can be understood as mapping from one kind of infoset (Cowan and Tobin, 2004) to another. Today most implementations of XML-based language processing applications process XML directly using programming languages and an API such as SAX or DOM. But many XML processing tasks don't *need* to be done at this low level. There are a number of XML Pipeline languages already available which allow you to specify sequences of standards-based XML operations. It is often possible to replace programming-language-based XML processing with short and simple XML Pipeline descriptions, for example

- schema-validate
- then do XInclude
- then transform with a stylesheet
- then send to a server via SOAP
- then validate the result
- then transform with another stylesheet

The W3C's XML Processing Model WG is working to produce an interoperable XML Pipeline language based on existing technology. The work has nearly finished, and the result should provide a language with wide application to language processing tasks.

The XProc language is itself expressed in XML, and has two main parts:

1. A language for specifying the sequence and configuration of processing steps;

2. A collection of built-in step types, including both low-level structure-manipulation and high-level standards-based operations

There is support for more than just straight-through pipelining of operations, with data-flow equivalents of conditions, loops, and exception-handlers.
The low-level manipulations available include

- sub-tree deletion
- element and/or attribute renaming
- sub-tree wrapping and unwrapping

The higher-level operations available include

- XInclude
- XSLT
- http-request

The pipeline paradigm for producing NLP systems has been heavily exploited by the Language Technology Group at the University of Edinburgh. One example, described in (B. Alex, C. Grover *et al.*, 2008), uses a multi-step pipeline to extract named entities, in particular proteins, from biomedical text, classify the extracted terms, and detect relations between terms. Steps in the pipeline range for generic, low-level tasks such as tokenisation and sentence-boundary detection to high-level processes such as relation extraction which involve not only entity-tagged data but also pre-computed statistical models.
The availability of a standardised XML pipeline language offers a real opportunity to improve the principled comparison of alternative approaches, as the modular nature of the pipeline architecture, together with the well-defined interfaces between modules which the XML document structures represent, will make it possible to do properly controlled comparisons of alternative approaches to the key stages in a complex process.

6. Internationalized Formatting: "XSL Formatting Objects (XSL-FO)"

The XSL 1.1 specification (Berglund, 2006) includes facilities for formatting XML, for example into PDF. XSL-FO 2.0 is currently being designed, with increased sophistication and also with increased support for Japanese formatting. The XSL-FO 2.0 requirements document (Bals, 2008) provides more information. XSL-FO is currently the most powerful and most completely internationalized of any widely-used standard for text formatting, with strong support for mixed-language work.
XSL-FO is a fixed XML vocabulary for formatting. In normal use one transforms input XML into the XSL-FO vocabulary using XSLT, and this transformed XML document is then rendered. It is also possible to produce XSL-FO directly, for example using XQuery.
XSl-FO copes with an arbitrary mix of text directions, both in what it calls the inline progression direction (e.g. right-to-left for Hebrew) and in what it calls the block progression direction (e.g. top to bottom for English, or right-to-left for vertical Japanese). It also defines how baselines should be mixed, for example when combining Devanagari and Arabic on the same line. Since language information is available to the formatter, language-specific hyphenation and line-breaking is also generally used.
Currently, XSL-FO is primarily aimed at automatic formatting in a content-driven environment: text flows into page areas, and new pages are created on demand from templates. XSL-FO 2.0 is expected to add support for format-driven processing, in which page areas fetch content as needed, but the 2.0 work is still in the early stages.
Readers interested in the future of XSL-FO are strongly encouraged to inspect the requirements document previously cited and to send comments to the Working Group as instructed in the Status section of that document.

7. XML Internationalization and Localization: "ITS 1.0"

The Internationalization Tag Set (ITS) 1.0 (Lieske and Sasaki, 2007)[5] is a specification which provides an XML vocabulary related to Internationalization and Localization of XML. A prototypical use is specifying which parts of an XML document should be translated or not translated during the localization of XML data. Such information can be expressed with two approaches, which can be used alternatively or complementary. First, *locally*, by adding in an XML document a *translate* attribute to the targeted element node, with the values *yes* or *no*. Second, by describing ITS 1.0 *global rules* which are independent of a specific location and can be applied to several XML documents. Such rules make use of XPath to specify the nodes to which the ITS information should pertain to.
ITS 1.0 specifies for 7 so-called "data categories" a way to express global and local information, defaults, and inheritance behavior (that is, does the information pertain to attributes and / or child elements):

- "Translate" to separate translatable from non-translatable content;
- "Localization Note" to communicate notes to localizers;
- "Terminology" to identify terms and optionally associate them with information, such as definitions;
- "Directionality" to allow the user to specify the base writing direction of blocks, embeddings, and overrides for the Unicode bidirectional algorithm;
- "Ruby" as a run of text which is associated with another run of text (the base text) and used to provide e.g. reading (pronunciation) guidance;

[5]The companion document (Savourel et al., 2008a) describes among others how to apply ITS 1.0 to new and existing XML formats.

- "Language information" to express the language of a piece of content;

- "Elements within Text" to specify the flow characteristics of an element, that is e.g. whether it is part of the flow of its parent, or is nested in a parent element and constitutes an independent flow.

Such information can be applied in many scenarios, for example within localization tools, for the extraction of translatable text, or as a preparation for the localization process. Below are two examples of using ITS 1.0 together with two important standards for XML localization: XLIFF and TBX.

XLIFF (Savourel et al., 2008b) is the "XML Localization Interchange File Format", a interchange file format for localizable content. A prototypical usage scenario is that out of some input data (e.g. an XML document) an XLIFF document is being generated, containing several translation units which wrap markup for the input "source" data and output "target" translations. Translators then create these translations, which are finally integrated into the source data.

The ITS 1.0 data category "Translate" can be used to generate XLIFF documents out of XML input data. Below is an XProc (see sec. 5.) pipeline informally described with the purpose of comparing results of translation tools. It consists of the following XProc steps:

- a step "extract translatable text" for the creation of an XLIFF document out of XML input and ITS 1.0 "Translate" information;

- an automatic translation step, using the XLIFF document and executed with a variety of tools;

- a step for comparing the results of the previous step for the different translation tools.

The use of ITS 1.0 as the first step in the XProc pipeline description helps to "hide" the specifics of input XML data. The automatic translation tools only have to understand the XLIFF format. In this way, the same processing chain can easily be re-used for new translation tools.

The TermBase eXchange (TBX) format is another example where ITS 1.0 helps to generalize processing chains. TBX is used for the representation of terminological information for human consumption or in NLP lexicons. The ITS 1.0 "Terminology" data category helps to identify terms locally or with global rules. In combination with the "Translate" data category, the following processing chain can be envisaged:

- All content which is described as a term via ITS 1.0 information is used to generate a terminological entry in a TBX file.

- The "Terminology" data category allows for adding information to selected terms, e.g. definitions. If such information is given, the information is also added to the terminological entry.

- An optional step relies on the "Translate" data category: for whose terms which are described as translatable, additional, language-specific markup in the terminological entry is generated. The content of this markup has to be filled by the localizer / translators, depending on the target language(s).

8. Outlook: The need for the Integration of Language Resources

The language resources community has struggled for years with the challenge of combining separately developed resources: how to combine your lexicon with mine, or your grammar, corpus etc. This problem is sometimes termed data integration. There are several areas of difficulty in the field of data integration. Some of these have been solved, at least partially, by some of the technologies that have been described in this paper:

- Accessing data from various sources in a uniform manner, so that they can be processed together; XQuery, XPath 2 and XSLT can combine data from (for example) XML documents, relational databases, geospatial databases and more. Current database management systems often allow data to be viewed either as relations or as XML, without requiring any particular effort beyond requesting the XML view. Data from other sources can often be accessed as if it were XML by interposing 'XML lenses' between the data and the consumer. In some cases, URI resolvers are modified to interpose the lens between the data source and the user, so that no active intervention by the user is required.

- Obtaining data (once accessed) in a uniform format, so that it can be processed uniformly; W3C XML Schema can be used to describe XML data, including embedded Internationalization information, and once everything is in XML, XML tools can be used.

- Mapping relationships between hierarchies; this is an unsolved research problem in general, and can be described as the difficulty of combining arguments made from differing and incompatible viewponts. W3C has an Ontology Language, OWL, but this in itself describes any single ontology, not relationships between ontologies. However, when XML documents are marked up with two different vocabularies, it is often possible in practice to write a declarative mapping function as an XSLT styleheet to transform between them.

- Processing; one is sometimes dealing with large amounts of data when handling language corpora. However, what may be voluminous to one observer might be miniscule to another: there are relational databases with petabytes of data that are processed on a daily basis. High volume processing with XQuery is in its infancy, but it is progressing fast. As a practical matter, this means that different XQuery implementations may offer very different performance characteristics, and it will often be useful to experiment with

more than one, to find one that fits a particular deployment scenario.

- Storing results; when the results must be stored to disparate databases, the underlying technology may have to translate formats automatically; the XQuery Update Facility is currently (May 2008) in a call for implementations, but promises to offer this functionality, saving changes both to files and to databases or other sources of structured information.
- Presenting results; here XSL-FO is a strong contender, with a number of implementations.

In all cases, the fact that every XML tool can process any XML document is a major benefit that greatly simplifies work.

9. References

S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, J. Melton, M. Rys, and J. Shanmugasundaram (eds.). 2007. XQuery 1.0 and XPath 2.0 Full-Text 1.0. Technical report, W3C. See <http://www.w3.org/TR/2007/WD-xpath-full-text-10-20070518/>.

B. Alex, C. Grover *et al.* 2008. Assisted curation: Does text mining really help? In *Pacific Symposium on Biocomputing 13*, pages 556–567.

K. Bals (ed.). 2008. Extensible Stylesheet Language (XSL) Requirements Version 2.0. Technical report, W3C. See <http://www.w3.org/TR/xslfo20-req/>.

A. Berglund (ed.). 2006. Extensible Stylesheet Language (XSL) Version 1.1. Technical report, W3C. See <http://www.w3.org/TR/2006/REC-xsl11-20061205/>.

D. J. Birnbaum and D. A. Mundie (eds.). 1999. The problem of anomalous data. *Markup Languages: Theory and Practice*, 1(4):1–19.

Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin (eds.). 2006. Namespaces in xml 1.1 (second edition). Technical report, W3C.

J. Cowan and R. Tobin (eds.). 2004. Xml information set (second edition). Technical report, W3C. See <http://www.w3.org/TR/2004/REC-xml-infoset-20040204/>.

S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson (eds.). 2007. W3C XML Schema Definition Language (XSDL) 1.1 Part 1: Structures. Technical report, W3C. See <http://www.w3.org/TR/2007/WD-xmlschema11-1-20070830/>.

I. Jacobs and N. Walsh (eds.). 2004. Architecture of the World Wide Web, Volume One. Technical report, W3C. See <http://www.w3.org/TR/2004/REC-webarch-20041215/>.

I. Jacobs (eds.). 2005. World Wide Web Consortium Process Document. Technical report, W3C. See <http://www.w3.org/2005/10/Process-20051014/>.

C. Lieske and F. Sasak (eds.). 2007. Internationalization Tag Set (ITS) 1.0. Technical report, W3C. See <http://www.w3.org/TR/2007/REC-its-20070403/>.

Paolo Marinelli, Claudio Sacerdoti Coen, and Fabio Vitali. 2004. Schemapath, a minimal extension to xml schema for conditional constraints. In *Proceedings of the Thirteenth International World Wide Web Conference*, pages 164–174, New York. ACM Press. Available on the Web in the ACM Digital Library; citation at <http://portal.acm.org/citation.cfm?doid=988672.988695>.

Bhalchandra Pandit, Valentina Popescu, and Virginia Smith (eds.). 2008. Service Modeling Language, Version 1.1. Technical report, W3C. See <http://www.w3.org/TR/sml/>.

D. Peterson, P. V. Biron, A. Malhotra, and C. M. Sperberg-McQueen (eds.). 2006. XML Schema 1.1 Part 2: Datatypes. Technical report, W3C. See <http://www.w3.org/TR/2006/WD-xmlschema11-2-20060217/>.

Y. Savourel, J. Kosek, and R. Ishida (eds.). 2008a. Best Practices for XML Internationalization. Technical report, W3C. See <http://www.w3.org/TR/2008/NOTE-xml-i18n-bp-20080213/>.

Y. Savourel, J. Reid, T. Jewtushenko, and R. M. Raya (eds.). 2008b. XLIFF Version 1.2. Technical report, OASIS. See <http://docs.oasis-open.org/xliff/v1.2/os/xliff-core.html>.

N. Walsh, A. Milowski, and H. S. Thompson (eds.). 2007. XProc: An XML Pipeline Language. Technical report, W3C. See <http://www.w3.org/TR/2007/WD-xproc-20071129/>.

D. J. Weitzner (ed.). 2004. W3C Patent Policy. Technical report, W3C. See <http://www.w3.org/Consortium/Patent-Policy-20040205/>.